Part VI: Administering

Tables of Contents

Administering

In addition to administering the database server, you can tune performance, replicate data, and archive data.

Quick reference cards that are suitable for printing

- Quick reference card: Configuration parameters in the onconfig.std file (PDF)
- Quick reference card: SQL administration API arguments (PDF)
- Quick reference card: onstat utility commands (PDF)
- Quick reference card: Enterprise Replication (PDF)

Portals that list reference information by functional area and include links

- onconfig portal
- SQL administration API portal
- onstat portal
- Database server utilities
- Environment variable portal
- Limits in Informix®

Main resources

- System administration
  These topics contain concepts, procedures, and reference information for database and database server administrators to use for managing and tuning IBM® Informix database servers.
- Backup and restore
  The backup and restore guides contain information about backing up and restoring data and managing storage devices and media.
- Replication
  The topics in this group contain information about replicating data in IBM Informix databases by using Enterprise Replication.

System administration

These topics contain concepts, procedures, and reference information for database and database server administrators to use for managing and tuning IBM® Informix® database servers.

- External resources for administration
- External resources for performance
- Main resources

External resources for administration

- Customizing Informix Dynamic Server for Your Environment (IBM Redbooks® publication)
  This IBM Redbooks publication provides an overview of some of the capabilities of version 11 of IBM Informix that enable it to be easily customized for your particular environment. The focus of this book is on the areas of ease of administration and application development.
- Recovery of a Down System (Best practice)
  This presentation provides best practices you should follow to bring a down system back into production as soon as possible. It discusses planning for and recovering from a critical situation that impacts your Informix database.
- Knowledge Collection: Informix Data Compression and Storage Optimization (Support document)
  This document helps you find the available resources that are related to the storage optimization feature.
- Using data file abstraction with external tables in Informix Dynamic Server (IBM developerWorks®)
  This article describes how to use the external table feature to easily load and unload data.
- Logical Logfile monitoring using SMI Tables (blog entry)
  This entry provides SQL statements that query system monitoring tables to monitor logging activity, log-backup status, and event alarms for logical logs.
- SYSDBOPEN: A flexible way to change session behavior in Informix (IBM developerWorks)
  Database administrators can use the **sysdbopen()** and **sysdbclose()** procedures to set environments to activate user tracing, handle short-lasting locks on data records, or change the reading behavior of sessions. This article shows how to create a **sysdbopen()** procedure that can be dynamically changed without re-creating the procedure to avoid downtime if session environments must be adjusted.
- Understand the Informix Server V11.70 defragmenter (IBM developerWorks)
  You can defragment a table, a fragment, or an index, including system catalogs. Defragmenting reduces the number of extents in the partition. In this article, you can learn about the defragmenter through usage examples.

External resources for performance tips

- IBM Informix on POWER7® (White paper)
  This technical white paper discusses best practices for using Informix on POWER7 systems to optimize performance for mission critical databases.
- Optimizing Informix database access (IBM developerWorks)
  This article explains how to improve the performance of your Informix database application by interrupting an SQL or connection request that requires more time than expected.
- Tune your Informix database for top performance, Part 1: Tuning the data model and the application (IBM developerWorks)
  This article explains some basic principles of how to tune an IBM Informix database application to get the best possible performance. In Part 1 of the series, see how the data model, the application, and updated statistics affect performance.
- Pushing IBM Informix Innovator-C to its Limits (magazine)
  This article describes performing a TPC-C-based stress test with IBM Informix Innovator-C edition.

Main resources

- administrative utilities and applications
  IBM Informix includes utilities and applications that you can use to perform administrative tasks and capture information about configuration and performance.
- Administrator's Guide
  These topics provide the information required to administer IBM Informix.
- Administrator's Reference
  These topics include comprehensive descriptions of IBM Informix configuration parameters, the system-monitoring interface (SMI) tables in the sysmaster database, the syntax of database server utilities such as onmode and onstat, logical-log records, disk structures, event alarms, and unnumbered error messages.
- DB-Access User's Guide
  This publication describes how to use the DB-Access utility to access, modify, and retrieve information from IBM Informix database servers.
- High-Performance Loader User's Guide
  These topics describe how to use the IBM Informix High-Performance Loader (HPL) to load and unload large quantities of data efficiently to or from an database.
- Performance Guide
  These topics describe how to configure and operate your IBM Informix database server to improve overall system throughput and to improve the performance of SQL queries.
- SNMP Subagent Guide
  These topics describe the Simple Network Management Protocol (SNMP) and the software that you need to use SNMP to monitor and manage IBM Informix database servers and databases.
- InformixHQ

administrative utilities and applications

IBM® Informix® includes utilities and applications that you can use to perform administrative tasks and capture information about configuration and performance.

Table 1. administrative utilities and applications

Utility name	Description
archecker	Verifies backups and performs table-level restores.
chkenv	Checks the validity of shared or private environment-configuration files. The **chkenv** utility validates the names of the environment variables in the file, but not their values.
cdr	Controls Enterprise Replication operations.
ClassGenerator	Generates a Java™ class for a named row type that is defined in the system catalog.
clusterIT_a.exe	Deprecated - Configures on the primary node for use in a cluster environment.
clusterIT_b.exe	Deprecated - Configures on the secondary node for use in a cluster environment.
dbaccess	Provides a user interface for entering, running, and debugging Structured Query Language (SQL) statements and Stored Procedure Language (SPL) routines.
dbexport	Unloads a database into text files for later import into another database and create a schema file.
dbimport	Creates and populates a database from text files. Use the schema file with the **dbimport** utility to re-create the database schema.
dbload	Loads data into databases or tables.
dbschema	Creates a file that contains the SQL statements that are needed to replicate a specified table, view, or database, or view the information schema.
finderr	Looks up a specific error code and displays the corresponding error text.
GenMacKey	Generates MAC key files for encrypting network communications.
genoncfg	Expedites the process of customizing a database server instance's onconfig file to a host environment.
glfiles	Generates a list of GLS-related files on the UNIX operating system.
ifxclone	Creates a snapshot of a database server.
ifxdeploy	Deploys a snapshot or removes a snapshot.
ifxdeployassist	Creates and customizes a snapshot.
ifx_getversion	On the UNIX operating system: outputs the complete version name of an Informix library.
ILOGIN	On the Windows operating system: tests the connection to a database server. The **ILOGIN** utility is in the %INFORMIXDIR%\demo directory.
infoshp	Reports information that is extracted from headers of the .shp, .shx, and .dbf files that make up ESRI shapefiles.
ipload	Manages the **onpload** database and creates the components of High-Performance Loader load and unload jobs through a UNIX GUI.
ixpasswd.exe	On the Windows operating system: Changes the logon password for all services that log on as user **informix**.
ixsu.exe	On the Windows operating system: Opens a command-line window that runs as the specified user.
loadshp	Loads spatial features and associated attributes from an ESRI shapefile into a table in an database.
ntchname.exe	On the Windows operating system: Changes the registry entries for from the old host name to the new host name.
onaudit	Manages audit masks and auditing configurations.
onbar	Backs up and restores storage spaces and logical logs.
oncheck	Checks specified disk structures for inconsistencies, repair inconsistent index structures, and display information about disk structures.
onclean	Forces a shutdown of the database server when normal shutdown from the **onmode** utility fails or when you cannot restart the server.
oncmsm	Starts, stops, installs, or uninstalls a Connection Manager; reloads a Connection Manager configuration file; or converts and earlier format Connection Manager configuration file to a current format configuration file.
ondblog	Changes the logging mode.

Utility name	Description
onconfig_diff	Compares two different onconfig files.
oninit	Starts the database server.
onkstore	Creates and manages keystore files for use with storage space encryption.
onload	Loads data that was created with the **onunload** utility into the database server.
onlog	Displays the contents of logical-log files.
onmode	Changes the operating mode of the database server, and performs various other operations on shared memory, sessions, transactions, parameters, and segments.
onparams	Modifies the configuration of logical logs or physical logs.
onpassword	Encrypts and decrypts password files that are used by Connection Managers or the **CDR** utility.
onperf	Monitors database server performance.
onpladm	Writes scripts and create files that automate data load and unload jobs.
onpload	Manages load and unload jobs directly from the command line.
onpsm	Manages IBM Informix Primary Storage Manager, which controls backup and restore devices for the **ON-Bar** utility.
onrestorept	Restores a server instance back to its original state just before the start of an upgrade.
onsecurity	Checks the security of a file, directory, or path and troubleshoots any existing problems.
onshowaudit	Extracts information from an audit trail.
onsmsync	Synchronizes the **sysutils** database and emergency boot file with the storage-manager catalog.
onspaces	Manages storage spaces.
onstat	Monitors the operation of the database server.
ontape	Logs, backs up, and restores data.
onunload	Unloads data from the database server.
setnet32	On the Windows operating system: Sets or modifies environment variables and network parameters that products use at run time.
SqlhDelete	Deletes the sqlhosts entries from the LDAP server.
SqlhUpload	Loads the sqlhosts entries from a flat ASCII file to the LDAP server in the prescribed format.
syncsqlhosts	Converts the connection information between the sqlhosts file format and the Windows registry format.
unloadshp	Copies spatial features and associated attributes from a table in an IBM Informix database into an ESRI shapefile.

Administrator's Guide

These topics provide the information required to administer IBM® Informix®.

A companion volume, the *IBM Informix Administrator's Reference*, contains reference material for using IBM Informix database servers. If you must tune the performance of your database server and SQL queries, see your *IBM Informix Performance Guide*.

This publication is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Performance engineers
- Programmers in the following categories
  - Application developers
  - DataBlade module developers
  - Authors of user-defined routines

This publication is written with the assumption that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
- Some experience with database server administration, operating-system administration, or network administration

- The database server
- Disk, memory, and process management
- Logging and log administration
- Fault tolerance
- High availability and scalability
  A successful production environment requires database systems that are always available, with minimal if any planned outages, and that can be scaled quickly and easily as business requirements change.
- Distributed data
- Overview of automatic monitoring and corrective actions
  You can use the SQL administration API, the Scheduler, and drill-down queries to manage automatic maintenance, monitoring, and administrative tasks.

The database server

- Overview of database server configuration and administration
  After you install IBM Informix, you configure the database server system and start administering the database server.
- Client/server communication
  These topics explain the concepts and terms that you must understand in order to configure client/server communication.
- Database server initialization
  The database server requires both disk-space initialization and shared-memory initialization.

Overview of database server configuration and administration

After you install IBM® Informix®, you configure the database server system and start administering the database server.

When you install IBM Informix, follow the installation instructions to ensure that all prerequisites are met (for example, the permissions of all key files and directories are set appropriately). The installation instructions are in the *IBM Informix Installation Guide*.

You must have the correct permissions to administer the database server. For most administration tasks, you need the following permissions:

- On UNIX, you must be logged in as user **root**, user **informix**, or the owner of the non-root installation. If role separation is enabled, you must be granted the DBSA role.
- On Windows, you must be a member of the **Informix-Admin** group.

You have various options to choose from when you configure the database server. Configuration includes customizing your environment and the database server. You can control how the database server runs and what function is available.

You must configure connectivity to connect to client administration tools and applications.

You must do some initial administration tasks to finish setting up your database server system. After you configure the database server, your administration responsibilities include a set of routine tasks.

- Database server concepts
  To administer the database server, you must understand key concepts around storage, configuration, logging, CPU use, shared memory use, and automation.
- Environment configuration
  You configure your environment by setting environment variables and creating or modifying files that relate to the environment variables. You can control whether environment variables are set at the environment level, for a specific user, or for a database session. You must set environment variables for the database server environment and for the client environments.
- Database server configuration
  You must customize the database server properties and features by setting configuration parameters, create storage spaces, and configure connectivity. You can automate startup.
- Database server maintenance tasks
  In addition to monitoring the database server for potential problems, regularly perform routine maintenance tasks to keep the server running smoothly and with optimum performance.

Database server concepts

To administer the database server, you must understand key concepts around storage, configuration, logging, CPU use, shared memory use, and automation.

Root dbspace
: The root dbspace is the initial dbspace, or storage space, that the database server creates. The root dbspace contains reserved pages and internal tables that describe and track all physical and logical units of storage. The root dbspace is the default location for logical logs, the physical log, databases, and temporary tables. The database server cannot run without the root dbspace.

Configuration (onconfig) file
: The database server requires a configuration file. Typically, the name of the configuration file is onconfig.*server_name*. The onconfig file contains configuration parameters that control database server properties. The database server reads the onconfig file during startup, shutdown, and for some operations while the server is running. Many configuration parameters can also be set dynamically while the database server is running.

Virtual processors
: A virtual processor runs multiple threads to perform queries and other tasks. The operating system schedules virtual processors as CPU processes. Multiple virtual processors run multiple threads in parallel. Virtual processors are divided into classes where each class is dedicated to processing a specific type of thread.

Logical logs
: The database server contains several logical log files that record data manipulation operations for logged databases, data definition operations for all databases, and administrative information such as checkpoint records and additions and deletions of chunks. A logical log is similar to a transaction log in other relational database management systems.

Physical log
: The physical log stores the before-images of pages. "Before images" are images of pages that are taken before the database server records the changed pages on disk. The unmodified pages are available in case the database server fails or a backup procedure requires the pages to provide an accurate snapshot of the database server data.

Buffer pool
: The buffer pool contains buffers that cache pages from disk in shared memory. Operations on pages that are cached run faster than operations on pages that must be retrieved from disk.

Caches

The database server uses caches to store information in shared memory instead of performing a disk read or another operation to obtain the information. Caching information improves performance for multiple queries that access the same tables.

Scheduler

The Scheduler is a subsystem that runs a set of tasks at predefined times or as determined internally by the server. Tasks are SQL statements can either collect information or run a specific operation. Some tasks are internal to the database server and run automatically. You can enable other tasks, if appropriate. You can also create your own tasks and schedule when they are run.

System databases

The system databases contain information about the database server. The **sysmaster** database contains the system-monitoring interface (SMI) tables. The SMI tables provide information about the state of the database server. The **sysadmin** database contains the tables that contain and organize the Scheduler tasks and sensors, store data that is collected by sensors, and record the results of Scheduler jobs and SQL administration API functions.

Environment configuration

You configure your environment by setting environment variables and creating or modifying files that relate to the environment variables. You can control whether environment variables are set at the environment level, for a specific user, or for a database session. You must set environment variables for the database server environment and for the client environments.

If you choose to create a database server instance during installation, the installation program sets the mandatory environment variables. Otherwise, you must set environment variables before you start the database server. The following environment variables are mandatory:

- The INFORMIXDIR environment variable specifies the directory where you installed the database server.
- The INFORMIXSERVER environment variable specifies the name of the database server.
- The ONCONFIG environment variable specifies the name of the onconfig file in the INFORMIXDIR/etc directory.
- The PATH environment variable must include the INFORMIXDIR/bin directory.

To configure the database server environment, you can set other environment variables:

- If you plan to create an sqlhosts file with a non-default name or location, set the INFORMIXSQLHOSTS environment variable to the name and path of your sqlhosts file.
- If you plan to use the DB-Access utility to run SQL statements, specify terminal properties with the INFORMIXTERM or a similar environment variable.
- If you need Global Language Support (GLS), set GLS environment variables.
- If you want to enable other functionality, set the appropriate environment variables. Some environment variables control functionality that is also controlled by configuration parameters. Environment variables override configuration parameter settings.

To configure client environments, you can set the environment variables that are supported by your client API. For more information, see your client API manual.

You can choose from multiple methods for setting environment variables. For example, you can run the SET ENVIRONMENT statement to set environment variables for the current session. You can add environment variable settings to log in scripts, at the command prompt, or in a configuration file.

Related information:
Environment variables
GLS-related environment variables
Environment variables for clients
Environment variable changes by version

Database server configuration

You must customize the database server properties and features by setting configuration parameters, create storage spaces, and configure connectivity. You can automate startup.

You customize the database server properties by setting or modifying configuration parameters in the onconfig file. The current version of IBM® Informix® does not use some configuration parameters that are used in earlier versions of the server.

If you choose to configure a database server during installation, many configuration parameter and environment variables are set and a set of storage spaces are created automatically. Alternatively, you can manually configure the database server.

When you start the database server for the first time, disk space is initialized and the initial chunk of the root dbspace is created. Any existing data in that disk space is overwritten. Shared memory that the database server requires is also initialized. When you subsequently start the database server, only shared memory is initialized. Although the root dbspace is the default location of log files and databases, you can store log files and databases in other storage spaces to prevent the root dbspace from running out of space.

- Storage space creation and management
  You can create multiple storage spaces to store different types of objects, such as, data, indexes, logs, temporary objects, instead of storing everything in the root dbspace. The way that you distribute the data on disks affects the performance of the database server. You can configure the database server to both automatically minimize the storage space that data requires and automatically expand storage space as needed. You can segregate storage and processing resources among multiple client organization by configuring multitenancy.
- Automatic performance tuning
  You can set configuration parameters and Scheduler tasks to enable the database server to automatically adjust values that affect performance. By default, many automatic tuning configuration parameters and Scheduler tasks are set to solve common performance issues.
- Feature configuration
  You can configure the database server to support the types of optional functionality that you need.
- Connectivity configuration
  The connectivity information allows a client application to connect to the database server on the network. You must prepare the connectivity information even if the

client application and the database server are on the same computer or node.

- Limit session resources
  You can limit the resources available to individual sessions to more evenly distribute system usage, and prevent resource monopolization.
- Automate startup and shutdown on UNIX
  You can modify startup and shutdown scripts on UNIX to automatically start and shut down the database server.
- Automate startup on Windows
  You can automate startup of the database server on Windows.

Related concepts:
Database server initialization
Related information:
Configuration parameter changes by version
Modifying the onconfig file
Database server configuration during installation
Creating a database server after installation

Storage space creation and management

You can create multiple storage spaces to store different types of objects, such as, data, indexes, logs, temporary objects, instead of storing everything in the root dbspace. The way that you distribute the data on disks affects the performance of the database server. You can configure the database server to both automatically minimize the storage space that data requires and automatically expand storage space as needed. You can segregate storage and processing resources among multiple client organization by configuring multitenancy.

A storage space is composed of one or more chunks. The maximum chunk size is 4 TB. You can have up to 32766 chunks in an instance.

After the database server is initialized, you can create storage spaces such as dbspaces and sbspaces. Use the **onspaces** utility to create storage spaces and chunks.

The following storage spaces are the most common:

dbspace
: Stores databases, tables, logical-log files, and the physical log file.
  Temporary dbspaces store temporary tables.

sbspace
: Stores smart large objects. Smart large objects consist of CLOB (character large object) and BLOB (binary large object) data types. User-defined data types can also use sbspaces. Some features of Informix® require sbspaces, such as Enterprise Replication, J/Foundation, spatial data types, and basic text search queries. In some cases, sbspaces are created automatically when needed.
  Temporary sbspaces store temporary smart large objects without logging metadata or user data.

plogspace
: Stores the physical log. If you do not create a plogspace, the physical log is stored in a dbspace.

Other types of storage spaces store specialized types of data.

If you create a server during installation, some storage spaces are created automatically.

Automatically minimizing storage space

You can minimize the amount of space that data needs by configuring automatic data compression and consolidation. You can compress data, consolidate and return free space, and merge extents. You can specify how frequently each of the operations occurs.

You can automatically rotate message logs to limit the amount of space for the logs.

Automatically extending storage space

After you create storage spaces, you can configure the server to automatically extend each storage space as needed. You create a storage pool of entries for available raw devices, cooked files, and directories, and you make sure that the SP_AUTOEXPAND configuration parameter set to the default value of 1. All types of storage spaces except external spaces (extspaces) are automatically expanded.

Automatically managing the location of data

You can automate the process of deciding where to locate databases, tables, and indexes. You can enable the database server to choose the most optimal location for databases, table, and indexes, and to automatically fragment tables. Instead of creating a new database in the root dbspace by default, the database server chooses the location by favoring non-critical spaces, spaces that have the most efficient page size, and other factors. The database server fragments new tables by round-robin and adds more fragments when necessary as the table grows.

You can override the automatic behavior by specifying a location for a database or table.

Multitenancy

You can create multiple tenant databases in a single Informix instance to segregate data, storage space, and processing resources among multiple client organizations.
Related concepts:
Managing automatic location and fragmentation
Data storage
Manage disk space
Automatic space management
Storage optimization

Multitenancy
Related information:
Manage message logs in an embedded environment

Automatic performance tuning

You can set configuration parameters and Scheduler tasks to enable the database server to automatically adjust values that affect performance. By default, many automatic tuning configuration parameters and Scheduler tasks are set to solve common performance issues.

You can configure the database server to adjust resources to improve performance:

- Increase the number of CPU virtual processors (VPs), up to the number of CPU processors or the number that you specify. Set the VPCLASS configuration parameter for the **cpu** class to **autotune=1**.
- Increase the number of AIO VPs. Set the VPCLASS configuration parameter for the **aio** class to **autotune=1**.
- Increase the size of the buffer pool. Set the BUFFERPOOL configuration parameter to enable the automatic extension of the buffer pool.
- Increase or decrease the size of private memory caches for CPU VPs. Set the VP_MEMORY_CACHE_KB configuration parameter to the initial size of the private memory caches.
- Increase the number of logical log files to improve performance. Set the AUTO_LLOG configuration parameter to 1, plus the name of the dbspace in which to add log files, and an optional maximum number of KB for all logical log files.
- Increase the size of the physical log as needed to improve performance. Create the plogspace to store the physical log.

If you created a server during installation, the buffer pool, logical log, and physical log are configured for automatic extension.

The following automatic tuning options are enabled by default. You can control whether the options are enabled.

- Increase the number of CPU virtual processors to half the number of CPU processors to ensure optimum performance. Control with the **auto_tune_cpu_vps** task in the Scheduler.
- Increase the number of AIO virtual processors and page cleaner threads increase I/O capability. Control with the AUTO_TUNE configuration parameter.
- Process read-ahead requests to reduce the time to wait for disk I/O. Control with the AUTO_TUNE configuration parameter.
- Trigger checkpoints as frequently as necessary and add logical log files as needed to avoid the blocking of transactions. Control with the AUTO_TUNE and the DYNAMIC_LOGS configuration parameters.
- Tune LRU flushing to improve transaction throughput. Control with the AUTO_TUNE configuration parameter.
- Reoptimize SPL routines and reprepare prepared objects after the schema of a table is changed to prevent manual processes and errors. Control with the AUTO_TUNE configuration parameter.
- Updates statistics that are stale or missing at scheduled intervals to improve query performance. Control with Auto Update Statistics tasks in the Scheduler and the AUTO_TUNE configuration parameter.
- Run light scans on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data. Control with the BATCHEDREAD_TABLE configuration parameter.
- Fetch a set of keys from an index buffer to reduce the number of times that a buffer is read. Control with the BATCHREAD_INDEX configuration parameter.
- Increase shared memory caches to improve query performance. Control with the DS_POOLSIZE, PC_POOLSIZE, PLCY_POOLSIZE, and USRC_POOLSIZE configuration parameters.

Related reference:
Built-in tasks and sensors
Related information:
Database configuration parameters
onspaces -c -P: Create a plogspace

Feature configuration

You can configure the database server to support the types of optional functionality that you need.

The following features are often enabled:

Parallel database queries
: You can control the resources that the database server uses to perform decision-support queries in parallel. You must balance the requirements of decision-support queries against the requirements of online transaction processing (OLTP) queries. The resources that you must consider include shared memory, threads, temporary table space, and scan bandwidth.

Data replication
: Data replication is the process of representing database objects at more than one distinct site.
High-availability cluster configurations consist of a primary server and one or more secondary servers that contain the same data as the primary server. High-availability clusters can provide redundancy, failover, workload balancing, and scalability. You can direct connections from applications to cluster servers with Connection Manager.
Enterprise Replication replicates all or a specified subset of the data between geographically distributed database servers. You can define set of replication servers as a grid to administer and run queries across the servers. You can combine a high-availability cluster and Enterprise Replication on the same database server.

Auditing
: If you enabled role separation when you installed the database server, you can audit selected activities. To use database server auditing, you must specify where audit records are stored, how to handle error conditions, and other configuration options. You also might want to change how users are audited if you suspect that they are abusing their access privileges.

Security
: You can keep your data secure by preventing unauthorized viewing and altering of data or other database objects. Use network encryption to encrypt data that is transmitted between servers and clients, and between servers. You can use column-level encryption to store sensitive data in an encrypted format. You create secure connections to the database server with authentication and authorization processes. You can encrypt storage spaces. Discretionary access control verifies

whether the user who is attempting to perform an operation is granted the required privileges to perform that operation. You can use label-based access control (LBAC) to control who has read access and who has write access to individual rows and columns of data.

Distributed queries
: You can use the database server to query and update multiple databases across multiple database servers or within the same database server instance. IBM® Informix® uses a two phase commit protocol to ensure that distributed queries are uniformly committed or rolled back across multiple database servers.

Disk mirroring
: When you use disk mirroring, the database server writes data to two locations. Mirroring eliminates data loss due to storage device failures. If mirrored data becomes unavailable for any reason, the mirror of the data is available immediately and transparently to users.

Warehousing
: You can create data warehouse applications and optimize your data warehouse queries. Informix Warehouse Accelerator is an in-memory database that boosts performance for analytic queries on operational and historical data. Informix Warehouse Accelerator uses a columnar, in-memory approach to accelerate complex warehouse and operational queries without application changes or tuning.

Related concepts:
High availability and scalability
Mirroring
Related information:
Parallel database query (PDQ)
Auditing data security
Securing data
Distributed queries
Overview of Informix Warehouse Accelerator
Storage space encryption
IBMInformix Enterprise Replication technical overview

Connectivity configuration

The connectivity information allows a client application to connect to the database server on the network. You must prepare the connectivity information even if the client application and the database server are on the same computer or node.

Informix® client/server connectivity information, the sqlhosts information, includes the database server name, the type of connection that a client can use to connect to the database server, the host name of the computer or node on which the database server runs, and the service name by which it is known. You do not need to specify all possible network connections in the sqlhosts information before you start the database server. However, to make a new connection available you must shut down the database server and then restart it.

The sqlhosts file contains connectivity information. You might also need to modify other connectivity and security files, depending on your needs.

When the database server is online, you can connect client applications and begin to create databases. Before you can access information in a database, the client application must connect to the database server environment. To connect to and disconnect from a database server, you can issue SQL statements from the client programs that are included in the , such as DB-Access, or API drivers.

Related reference:
Connectivity files

Limit session resources

You can limit the resources available to individual sessions to more evenly distribute system usage, and prevent resource monopolization.

Set the following configuration parameters to impose limits on sessions:

- SESSION_LIMIT_LOCKS limits the number of locks a tenant session can acquire.
- SESSION_LIMIT_MEMORY limits the amount of memory that can be allocated for a session.
- SESSION_LIMIT_TEMPSPACE limits the amount of temporary table space that can be allocated for a session.
- SESSION_LIMIT_LOGSPACE limits the size of transactions within a session, based on the amount of log space that an individual transaction would fill.
- SESSION_LIMIT_TXN_TIME limits the amount of time that a transaction is allowed to run within a session.

You can set the IFX_SESSION_LIMIT_LOCKS environment option in the session, to specify a lower lock limit than the SESSION_LIMIT_LOCKS configuration parameter value.

Session limits do not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Transactions and sessions that exceed a set limit are terminated by the **session_mgr** thread. The **session_mgr** thread starts when the database server starts, and remains inactive until a session limit is exceeded.

Related information:
SESSION_LIMIT_LOCKS configuration parameter
SESSION_LIMIT_MEMORY configuration parameter
SESSION_LIMIT_TEMPSPACE configuration parameter
SESSION_LIMIT_LOGSPACE configuration parameter
SESSION_LIMIT_TXN_TIME configuration parameter
TENANT_LIMIT_SPACE configuration parameter
onstat -g ses command: Print session-related information

Automate startup and shutdown on UNIX

You can modify startup and shutdown scripts on UNIX to automatically start and shut down the database server.

UNIX startup script

Modify the UNIX startup script to start the database server automatically when your computer enters multiuser mode.

1. Add UNIX and database server utility commands to the UNIX startup script so that the script performs the following actions:
   - Sets the INFORMIXDIR environment variable to the full path name of the directory in which the database server is installed.
   - Sets the PATH environment variable to include the $INFORMIXDIR/bin directory.
   - Sets the INFORMIXSERVER environment variable so that the **sysmaster** database can be updated (or created, if necessary).
   - Runs the **oninit** command, which starts the database server and leaves it in online mode.
2. If you plan to start multiple versions of the database server (multiple residency), you must add commands in the script to set the ONCONFIG and INFORMIXSERVER environment variables and run the **oninit** command for each instance of the database server.
3. If different versions of the database server are installed in different directories, you must add commands to the script to set the INFORMIXDIR environment variable and repeat the preceding steps for each version.

UNIX shutdown script

Modify your UNIX shutdown script to shut down the database server in a controlled manner whenever UNIX shuts down. The database server shutdown commands run after all client applications complete transactions and exit.

1. Add UNIX and database server utility commands to the UNIX shutdown script so that the script performs the following tasks:
   - Sets the INFORMIXDIR environment variable to the full path name of the directory in which the database server is installed.
   - Sets the PATH environment variable to include the $INFORMIXDIR/bin directory.
   - Sets the ONCONFIG environment variable to the appropriate configuration file.
   - Runs the **onmode -ky** command, which initiates an immediate shutdown and takes the database server offline.
2. If you are running multiple versions of the database server (multiple residency), you must add commands in the script to set the ONCONFIG environment variable and run the **onmode -ky** command for each instance.
3. If different versions of the database server are installed in different directories, you must add commands to the script to set the INFORMIXDIR environment variable and repeat the preceding steps for each version.

Related information:
Database configuration parameters
The oninit utility
Environment variables in Informix products

Automate startup on Windows

You can automate startup of the database server on Windows.

To start the database server automatically when Windows starts:

1. From the Service control application window, select the IBM® Informix® service and click Startup.
2. Select Automatic in the Status Type dialog box.
3. In the Log On As dialog box, select This Account and verify that `informix` is in the text box.

To stop automatic startup, clear the Automatic property.

Database server maintenance tasks

In addition to monitoring the database server for potential problems, regularly perform routine maintenance tasks to keep the server running smoothly and with optimum performance.

You can use the Informix® command-line utilities to perform the following tasks. Not all of the following tasks are appropriate for every installation.

Backup data and logical log files
: To ensure that you can recover your databases in the event of a failure, make frequent backups of your storage spaces and logical logs. You can create backups with the ON-Bar utility or the **ontape** utility.

Check data for consistency
: To ensure that data is consistent, perform occasional checks.

Manage logical logs
: To ensure database server performance, perform logical-log administration tasks, such as, backing up logical-log files, adding, freeing, and resizing logical-log files, and specifying high-watermarks. The database server dynamically allocates logical-log files while online to prevent long transactions from blocking user transactions.

Manage the physical log

To ensure database server performance, make sure that you allocate enough space for the physical log. You can change the size and location of the physical log. When the database server starts, it checks whether the physical log is empty because that implies that the server shut down in a controlled fashion. If the physical log is not empty, the database server automatically performs a *fast recovery*. Fast recovery automatically restores the databases to a state of physical and logical consistency after a system failure that might have left one or more transactions uncommitted.

Manage shared memory

To ensure that the database server has the appropriate amount of shared memory to maintain performance goals, perform the following tasks:

- Changing the size or number of buffers (by changing the size of the logical-log or physical-log buffer, or changing the number of buffers in the shared-memory buffer pool)
- Changing shared-memory parameter values
- Changing forced residency (on or off, temporarily or for a session)
- Tuning checkpoint intervals
- Adding segments to the virtual portion of shared memory
- Configuring the SQL statement cache to reduce memory usage and preparation time for queries

Manage virtual processors

To ensure database server performance, configure enough virtual processors (VPs). The configuration and management of VPs has a direct affect on the performance of a database server. The optimal number and mix of VPs for your database server depends on your hardware and on the types of applications that your database server supports.

Manage the database server message log

To ensure that message log space does not fill, monitor the size of the database server message log. The database server appends new entries to this file. You can enable the automatic rotating of the database server message log to limit the total size of the log files.

Related concepts:
Consistency checking
Logical log
Physical logging, checkpoints, and fast recovery
Shared memory
Virtual processors and threads
Related information:
Overview of backup and restore
Tasks that automatically rotate message log files

Client/server communication

These topics explain the concepts and terms that you must understand in order to configure client/server communication.

- Client/server architecture
  IBM® Informix® products conform to the *client/server* software-design model.
- Connections that the database server supports
  The database server supports different types of connections with client application.
- Local connections
  A *local connection* is a connection between a client and the database server on the same computer.
- Communication support services
  Communication support services include connectivity-related services
- Connectivity files
  The *connectivity files* contain the information that enables client/server communication and enable a database server to communicate with another database server.
- The sqlhosts information
  The sqlhosts information contains connectivity information for each database server and definitions for groups. The database server looks up the connectivity information when you start the database server, when a client application connects to a database server, or when a database server connects to another database server.
- Informix support for IPv6 addresses
  On all platforms, IBM Informix recognizes Internet Protocol Version 6 (IPv6) addresses, which are 128 bits long, and Internet Protocol Version 4 (IPv4) addresses, which are 32 bits long.
- Configuration parameters related to connectivity
  Some of the configuration parameters in the onconfig file specify information related to connectivity.
- Environment variables for network connections
  The INFORMIXCONTIME (connect time) and INFORMIXCONRETRY (connect retry) environment variables affect the behavior of the client when it is trying to connect to a database server. Use these environment variables to minimize connection errors caused by busy network traffic.
- Automatically terminating idle connections
  You can automatically terminate sessions with clients that have been idle for a specified time by enabling the **idle_user_timeout** Scheduler task.
- Distributed Relational Database Architecture (DRDA) communications
  DRDA is a set of protocols that enables multiple database systems and application programs to work together.
- Examples of client/server configurations
  The next several sections show the correct sqlhosts entries for several client/server connections.
- IBM Informix MaxConnect
  IBM Informix MaxConnect is a networking product for IBM Informix database server environments on UNIX. Informix MaxConnect manages large numbers (from several hundred to tens of thousands) of client/server connections.

Related tasks:
Changing client connectivity information

Client/server architecture

IBM® Informix® products conform to the *client/server* software-design model.

Application or clients can be on the computer housing the database server or on a different computer. Client applications issue requests for services and data from the database server. The database server responds by providing the services and data that the client requested.

You use a *network protocol* together with a *network programming interface* to connect and transfer data between the client and the database server.

- Network protocol
  A *network protocol* is a set of rules that govern how data is transferred between applications and, in this context, between a client and a database server. The rules of a protocol are implemented in a *network driver*. A network driver contains the code that formats the data when it is sent from client to database server and from database server to client.
- Network programming interface
  A *network programming interface* is an application programming interface (API) that contains a set of communications routines or system calls. An application can call these routines to communicate with another application that is on the same or on different computers.
- Windows network domain
  Windows network technology enables you to create network *domains*. A domain is a group of connected Windows computers that share user account information and a security policy.
- Database server connections
  A client application establishes a connection to a database server with either the CONNECT or DATABASE SQL statement.
- Supporting multiplexed connections
  A *multiplexed connection* uses a single network connection between the database server and a client to handle multiple database connections from the client.

Network protocol

A *network protocol* is a set of rules that govern how data is transferred between applications and, in this context, between a client and a database server. The rules of a protocol are implemented in a *network driver*. A network driver contains the code that formats the data when it is sent from client to database server and from database server to client.

Clients and database servers gain access to a network driver by way of a *network programming interface*. A network programming interface contains system calls or library routines that provide access to network-communications facilities. An example of a network programming interface for UNIX is TLI (Transport Layer Interface). An example of a network programming interface for Windows is WINSOCK (sockets programming interface).

The power of a network protocol lies in its ability to enable client/server communication even though the client and database server are on different computers with different architectures and operating systems.

You can configure the database server to support more than one protocol, but consider this option only if some clients use TCP/IP.

Related concepts:
The sqlhosts file and the SQLHOSTS registry key
Database server connections
Related tasks:
Connections that the database server supports
Related reference:
Network-configuration files

Network programming interface

A *network programming interface* is an application programming interface (API) that contains a set of communications routines or system calls. An application can call these routines to communicate with another application that is on the same or on different computers.

In the context of this explanation, the client and the database server are the applications that call the routines in the TLI or sockets API. Clients and database servers both use network programming interfaces to send and receive the data according to a communications protocol.

Both client and database server environments must be configured with the same protocol if client/server communication is to succeed. However, some network protocols can be accessed through more than one network programming interface. For example, TCP/IP can be accessed through either TLI or sockets, depending on which programming interface is available on the operating-system platform.

Related concepts:
Communication support services
Network security files
Related reference:
A network connection

Windows network domain

Windows network technology enables you to create network *domains*. A domain is a group of connected Windows computers that share user account information and a security policy.

A *domain controller* manages the user account information for all domain members. The domain controller facilitates network administration. By managing one account list for all domain members, the domain controller relieves the network administrator of the requirement to synchronize the account lists on each of the domain computers. In other words, the network administrator who creates or changes a user account must update only the account list on the domain controller rather than the account lists on each of the computers in the domain.

To log in to a Windows database server, a user on another Windows computer must belong to either the same domain or a *trusted domain*. A trusted domain is one that establishes a *trust relationship* with another domain. In a trust relationship, user accounts are only in the trusted domain.

A user who attempts to log in to a Windows computer that is a member of a domain can do so either by using a local login and profile or a domain login and profile. However, if the user is listed as a trusted user or the computer from which the user attempts to log in is listed as a trusted host, the user can be granted login access without a profile.
Important: A client application can connect to the database server only if there is an account for the user ID in the Windows domain in which the database server runs. This rule also applies to trusted domains.
If you specify a user identifier but no domain name for a connection to a workstation that expects both a domain name and a user name (domain\user), the database server checks only the local workstation and the primary domain for the user account. If you explicitly specify a domain name, that domain is used to search for the user account. The attempted connection fails with error -951 if no matching domain\user account is found on the local workstation.

Use the CHECKALLDOMAINSFORUSER configuration parameter to configure how the database server searches for user names in a networked Windows environment.

Table 1. Locations searches for user names specified either alone or with a domain name.

	Domain and user specified	**User name only specified**
CHECKALLDOMAINSFORUSER is unset	Searches in the specified domain only	Searches on the local host only
`CHECKALLDOMAINSFORUSER=0`	Searches in the specified domain only	Searches on the local host only
`CHECKALLDOMAINSFORUSER=1`	Searches in the specified domain only	Searches in all domains

Important: The database server's trusted client mechanism is unrelated to the trust relationship that you can establish between Windows domains. Therefore, even if a client connects from a trusted Windows domain, the user must have an account in the domain on which the database server is running.
Related information:
CHECKALLDOMAINSFORUSER configuration parameter

Database server connections

A client application establishes a connection to a database server with either the CONNECT or DATABASE SQL statement.

An application might contain the following CONNECT statement to connect to the database server named **my_server**:

```
CONNECT TO '@my_server'
```

Tip: The database server's internal communications facility is called Association Services Facility (ASF). If you see an error message that includes a reference to ASF, you have a problem with your connection.
Related reference:
Network protocol
Related information:
CONNECT statement
DATABASE statement

Supporting multiplexed connections

A *multiplexed connection* uses a single network connection between the database server and a client to handle multiple database connections from the client.

Client applications can establish multiple connections to a database server to access more than one database on behalf of a single user. If the connections are not multiplexed, each database connection establishes a separate network connection to the database server. Each additional network connection uses additional computer memory and processor time, even for connections that are not active. Multiplexed connections enable the database server to create multiple database connections without using up the additional computer resources that are required for additional network connections.

To configure the database server to support multiplexed connections:

1. Define an alias using the DBSERVERALIASES configuration parameter. For example, specify:

   ```
   DBSERVERALIASES ifx_mux
   ```

2. Add an sqlhosts file entry for the alias using `onsqlmux` as the **nettype** entry. The **hostname** and **servicename**, must have entries, but the entries are ignored. Dashes (-) can be used as entries. For example:

   ```
   #dbservername   nettype    hostname   servicename   options
   ifx_mux         onsqlmux   -          -
   ```

3. Enable multiplexing for the selected connection types by specifying `m=1` in the sqlhosts entry that the client uses for the database server connection. For example:

   ```
   #dbservername   nettype    hostname   servicename   options
   menlo           ontlitcp   valley     jfkl          m=1
   ```

4. On Windows platforms, you must also set the IFX_SESSION_MUX environment variable.

The following example shows both onconfig file and sqlhosts file entries.

onconfig file:

```
DBSERVERNAME web_tli
DBSERVERALIASES web_mux
```

sqlhosts file:

```
#dbservername   nettype    hostname   servicename   options
web_tli         ontlitcp   node5      svc5          m=1
web_mux         onsqlmux   -          -
```

You are not required to change the sqlhosts information that the database server uses. The client program does not require any special SQL calls to enable connections multiplexing. Connection multiplexing is enabled automatically when the onconfig file and the sqlhosts entries are configured appropriately and the database server starts.

Multiplexed connections do not support:

- Multithreaded client connections
- Shared-memory connections
- Connections to subordinate database servers (for distributed queries or data replication, for example)

If any of these conditions exist when an application attempts to establish a connection, the database server establishes a standard connection. The database server does not return an SQL error.
The Informix® ESQL/C **sqlbreak()** function is not supported during a multiplexed connection.

Related concepts:
The sqlhosts file and the SQLHOSTS registry key
Related reference:
sqlhosts file and SQLHOSTS registry key options
Related information:
DBSERVERNAME configuration parameter
DBSERVERALIASES configuration parameter
Multiplexed connections and CPU utilization

Connections that the database server supports

The database server supports different types of connections with client application.

The following types of connections are supported by the database server.

Connection type	Windows	UNIX	Local	Network
Sockets	X	X	X	X
TLI (TCP/IP)		X	X	X
Shared memory		X	X	
Secure Sockets Layer (SSL)	X	X		X
Stream pipe		X	X	
Named pipe	X		X	

Secure Sockets Layer (SSL) connections use encryption for data communication between two points over a network.

When configuring connectivity, consider setting the LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTION configuration parameters. These parameters enable you to reduce the risk of a hostile denial-of-service (DOS) attack by making it more difficult to overwhelm the Listener VP that handles connections.

UNIX only: On many UNIX platforms, the database server supports multiple network programming interfaces. The machine notes show the interface/protocol combinations that the database server supports for your operating system.
To set up a client connection:

1. Specify connectivity and connection configuration parameters in your onconfig file.
2. Set up appropriate entries in the connectivity files on your platform.
3. Specify connectivity environment variables in your UNIX start-up scripts or the local and domain-wide Windows registries.
4. Add an sqlhosts entry to define a dbserver group for your database server.

Related concepts:
The sqlhosts information
Related reference:
Network protocol
Connectivity files
Configuration parameters related to connectivity
Environment variables for network connections
sqlhosts connectivity information
Related information:
NETTYPE configuration parameter
Secure sockets layer protocol
LISTEN_TIMEOUT configuration parameter

MAX_INCOMPLETE_CONNECTIONS configuration parameter
Limiting denial-of-service flood attacks

Local connections

A *local connection* is a connection between a client and the database server on the same computer.

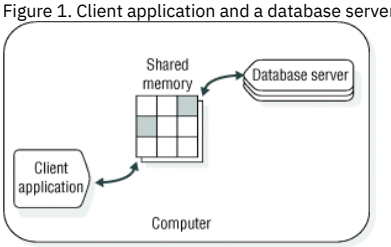
The following topics describe different types of local connections.

- Shared-memory connections (UNIX)
  A *shared-memory connection* uses an area of shared-memory as the channel through which the client and database server communicate with each other. A client cannot have more than one shared-memory connection to a database server.
- Stream-pipe connections (UNIX and Linux)
  A *stream pipe* is a UNIX interprocess communication (IPC) facility that allows processes on the same computer to communicate with each other.
- Named-pipe connections (Windows)
  Named pipes are application programming interfaces (APIs) for bidirectional interprocess communication (IPC) on Windows.
- Local-loopback connections
  A network connection between a client application and a database server on the same computer is called a *local-loopback* connection.

Shared-memory connections (UNIX)

A *shared-memory connection* uses an area of shared-memory as the channel through which the client and database server communicate with each other. A client cannot have more than one shared-memory connection to a database server.

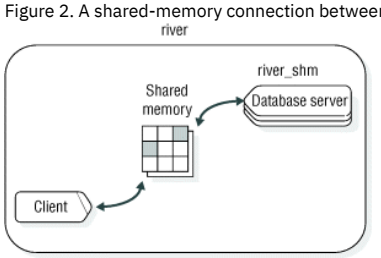
The following figure illustrates a shared-memory connection.
Figure 1. Client application and a database server communication through a shared-memory connection.



Shared memory provides fast access to a database server, but it poses some security risks. Errant or malicious applications might delete or view message buffers of their own or of other local users. Shared-memory communication is also vulnerable to programming errors if the client application performs explicit memory addressing or over-indexes data arrays. Such errors do not affect the database server if you use network communication or stream pipes.

Example of a shared-memory connection

The following figure shows a shared-memory connection on the computer named **river**.
Figure 2. A shared-memory connection between a client application and a database server named river_shm.



The onconfig file for this installation includes the following line:

```
DBSERVERNAME river_shm
```

The sqlhosts file for this installation includes the following lines:

```
#dbservername   nettype   hostname   servicename options
 river_shm      onipcshm  river      rivershm
```

The client application connects to this database server using the following statement:

```
CONNECT TO '@river_shm'
```

For a shared-memory connection, no entries in network configuration files are required. Use arbitrary values for the `hostname` and `servicename` fields of the sqlhosts file.

Related concepts:
Communications portion of shared memory (UNIX)
How a client attaches to the communications portion (UNIX)

Related reference:
Examples of client/server configurations

Stream-pipe connections (UNIX and Linux)

A *stream pipe* is a UNIX interprocess communication (IPC) facility that allows processes on the same computer to communicate with each other.

Stream-pipe connections have the following advantages:

- Unlike shared-memory connections, stream pipes do not pose the security risk of being overwritten or read by other programs that explicitly access the same portion of shared memory.
- Unlike shared-memory connections, stream-pipe connections allow distributed transactions between database servers that are on the same computer.

Stream-pipe connections have the following disadvantages:

- Stream-pipe connections might be slower than shared-memory connections on some computers.
- Stream pipes are not available on all platforms.
- When you use shared memory or stream pipes for client/server communications, the **hostname** entry is ignored.

Related reference:
sqlhosts connectivity information

Named-pipe connections (Windows)

Named pipes are application programming interfaces (APIs) for bidirectional interprocess communication (IPC) on Windows.

Named-pipe connections provide a high-level interface to network software by making transport-layer operations transparent. Named pipes store data in memory and retrieve it when requested, in a way that is similar to reading from and writing to a file system.

Local-loopback connections

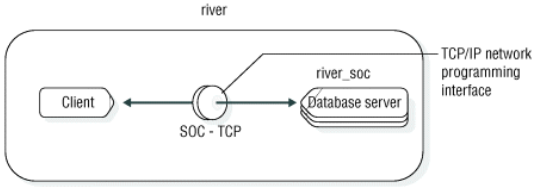
A network connection between a client application and a database server on the same computer is called a *local-loopback* connection.

The networking facilities used are the same as if the client application and the database server were on different computers. You can make a local-loopback connection provided your computer is equipped to process network transactions. Local-loopback connections are not as fast as shared-memory connections, but they do not pose the security risks of shared memory.

In a local-loopback connection, data seems to pass from the client application, out to the network, and then back in again to the database server. The internal connection processes send the information directly between the client and the database server and do not put the information out on the network.

An example of a local-loopback connection

The following figure shows a local-loopback connection that uses sockets and TCP/IP.
Figure 1. A local-loopback connection between a client and a database server named **river_soc** on a computer named **river**.



The sqlhosts file for this installation includes the following lines:

```
#dbservername   nettype   hostname   servicename   options
 river_soc      onsoctcp  river      riverol
```

If the network connection uses TLI instead of sockets, only the **nettype** entry in this example changes. In that case, the **nettype** entry is `ontlitcp` instead of `onsoctcp`.
The onconfig file for this installation includes the following lines:

```
DBSERVERNAME river_soc
```

This example assumes that an entry for **river** is in the hosts file and an entry for **riverol** is in the services file.

Related reference:
Examples of client/server configurations

Communication support services

Communication support services include connectivity-related services

Some connectivity-related services are:

- **Authentication**, which is the process of verifying the identity of a user or an application. The most common form of authentication is to require the user to enter a name and password to obtain access to a computer or an application.
- **Message integrity**, which ensures that communication messages are intact and unaltered when they arrive at their destination.
- **Message confidentiality**, which protects messages from unauthorized viewing, usually through encryption and decryption, during transmission.

Communication support services can also include other processing such as data compression or traffic-based accounting.

The database server provides extra security-related communication support services through plug-in software modules called Communication Support Modules (CSM). The database server uses the default authentication policy when you do not specify a communications support module.

Related concepts:
Network programming interface
Network security files
Related information:
Communication support modules for data transmission encryption
Single sign-on authentication
Simple password encryption

Connectivity files

The *connectivity files* contain the information that enables client/server communication and enable a database server to communicate with another database server.

The connectivity configuration files can be divided into three groups:

- Network-configuration files
- Network security files
- The sqlhosts file

Windows: On the database server, the connectivity information is stored in the sqlhosts file; however, on clients the connectivity information is stored in the SQLHOSTS registry.

- Network-configuration files
  These topics identify and explain the use of network-configuration files on TCP/IP networks.
- Network security files
  IBM® Informix® products follow standard security procedures that are governed by information contained in the network security files.
- The sqlhosts file and the SQLHOSTS registry key
  IBM Informix client/server connectivity information, the *sqlhosts information*, contains information that enables a client application to find and connect to any IBM Informix database server on the network.

Related tasks:
Connections that the database server supports
Related reference:
Connectivity configuration

Network-configuration files

These topics identify and explain the use of network-configuration files on TCP/IP networks.

- TCP/IP connectivity files
  When you configure the database server to use the TCP/IP network protocol, you use information from the hosts and services files to prepare the sqlhosts information.
- Multiple TCP/IP ports
  You can modify the services file to take advantage of having multiple Ethernet cards.

Related reference:
Network protocol

TCP/IP connectivity files

When you configure the database server to use the TCP/IP network protocol, you use information from the hosts and services files to prepare the sqlhosts information.

The hosts file requires a single entry for each network-controller card that connects a computer running IBM® Informix® client/server products on the network. Each entry in the file contains the IP address (or ethernet card address) and host name. You can also include the host alias. Although the length of the host name is not limited in the hosts file, the IBM Informix database server limits the host name to 256 bytes.

The following example has two entries.

```
#address        hostname  alias
98.555.43.21    odyssey
12.34.56.555    illiad    sales
```

The services file contains an entry for each service available through TCP/IP. Each entry is a single line that contains the following information:

- Service name
  IBM Informix products use this name to determine the port number and protocol for making client/server connections. The service name is limited to 128 bytes.
- Port number and connection protocol, separated by a forward slash (/) character
  The port number is the computer port, and the protocol for TCP/IP is `tcp`.

  The operating system imposes restrictions on the port number. User **informix** must use a port number equal to or greater than `1024`. Only **root** users are allowed to use a port number lower than `1024`.
- Host Aliases (optional)

The service name and port number are arbitrary. However, they must be unique within the context of the file and must be identical on all the computers running IBM Informix client/server products. The following example has one entry:

```
#servicename    port/protocol
server2         1526/tcp
```

This entry makes `server2` known as the service name for TCP port `1526`. A database server can then use this port to service connection requests.

Important: For database servers that communicate with other database servers, you must define either a TCP/IP connection or an IPCSTR (interprocess communications stream pipe) connection for the DBSERVERNAME configuration parameter. You can also define at least one DBSERVERALIASES configuration parameter setting with the appropriate connection protocol for connectivity between the coordinator and the subordinate servers. For cross-server transactions, each participating server must support a TCP/IP or an IPCSTR connection with the coordinator, even if both database server instances are on the same workstation.
You typically include a separate NETTYPE parameter for each connection type that is associated with a dbserver name. You list dbserver name entries in the DBSERVERNAME and DBSERVERALIASES configuration parameters. You associate connection types with dbserver names through entries in the sqlhosts file or registry. The hosts and services files must be available to each computer that runs IBM Informix client/server products.

UNIX:

- The hosts and services files are in the /etc directory.
- On systems that use NIS, the hosts and services files are maintained on the NIS server. The hosts and services files that are on your local computer might not be used and might not be up to date. To view the contents of the NIS files, enter the following commands on the command line:

  ```
  ypcat hosts
  ypcat services
  ```

Windows:

- The hosts and services files are in %WINDIR%\system32\drivers\etc\.
- You might want to configure TCP/IP to use the Domain Name Service (DNS) for host name resolutions.
- The Dynamic Host Configuration Product (DHCP) dynamically assigns IP addresses from a pool of addresses instead of using IP addresses that are explicitly assigned to each workstation. If your system uses DHCP, Windows Internet Name Service (WINS) is required. DHCP is transparent to the database server.
- Client and server actions when a TCP/IP connection is opened
  When a TCP/IP connection is opened, information is read on both the client side and server side.

Related reference:
sqlhosts connectivity information

Client and server actions when a TCP/IP connection is opened

When a TCP/IP connection is opened, information is read on both the client side and server side.

The following information is read on the client side:

- The INFORMIXSERVER environment variable.
- The hosts file information (INFORMIXSQLHOSTS environment variable, $INFORMIXDIR/etc/sqlhosts file and services file information
- Other environment variables
- Resource files

The following information is read on the server side:

- The DBSERVERNAME configuration parameter
- The DBSERVERALIASES configuration parameter
- Server environment variables and configuration parameters, including any NETTYPE configuration parameter setting that manages TCP/IP connections.

Related information:
NETTYPE configuration parameter

Multiple TCP/IP ports

You can modify the services file to take advantage of having multiple Ethernet cards.

To take advantage of multiple Ethernet cards:

- Make an entry in the services file for each port the database server uses, as in the following example:

```
#servicename    port/protocol    alias
soc1            21/tcp
soc2            22/tcp
```

  Each port of a single IP address must be unique. Separate ethernet cards can use unique or shared port numbers. You might want to use the same port number on ethernet cards connecting to the same database server. (In this scenario, the service name is the same.)

- Put one entry per ethernet card in the hosts file with a separate IP address, as in the following example:

```
#address          hostname    alias
192.147.104.19    svc8
192.147.104.20    svc81
```

- In the onconfig file, set DBSERVERNAME configuration parameter for one of the ethernet cards and the DBSERVERALIASES configuration parameter for the other ethernet card. The following lines show sample entries in the onconfig file:

```
DBSERVERNAME chicago1
DBSERVERALIASES chicago2
```

- Add one sqlhosts entry for each ethernet card. That is, make an entry for the DBSERVERNAME and another entry for the DBSERVERALIASES.

```
#dbservername    nettype    hostname    servicename    options
chicago1         onsoctcp   svc8        soc1
chicago2         onsoctcp   svc81       soc2
```

After this configuration is in place, the application communicates through the ethernet card assigned to the dbserver name that the INFORMIXSERVER environment variable provides.

Related information:
INFORMIXSERVER environment variable

Network security files

IBM® Informix® products follow standard security procedures that are governed by information contained in the network security files.

For a client application to connect to a database server on a remote computer, the user of the client application must have a valid user ID on the remote computer.

- Trusted-host information
  Users on trusted hosts are allowed to access the local system without supplying a password. You can include an optional user name to limit the authentication to a specific user on a specific host.
- Trusted-user information
  A user can list hosts from which they can connect as a trusted user in their .rhosts file.
- The netrc information
  The netrc information is optional information that specifies identity data. A user who does not have authorization to access the database server or is not on a computer that is trusted by the database server can use this file to supply a name and password that are trusted. A user who has a different user account and password on a remote computer can also provide this information.

Related concepts:
Network programming interface
Communication support services
Related information:
REMOTE_SERVER_CFG configuration parameter
REMOTE_USERS_CFG configuration parameter
S6_USE_REMOTE_SERVER_CFG configuration parameter

Trusted-host information

Users on trusted hosts are allowed to access the local system without supplying a password. You can include an optional user name to limit the authentication to a specific user on a specific host.

Use one of the following *trusted-hosts files* to specify remote hosts for **rlogin**, **rsh**, **rcp**, and **rcmd** remote-authentication:

- hosts.equiv
- The file that is specified by a database server's REMOTE_SERVER_CFG configuration parameter

Use trusted-hosts information only for client applications that do not supply a user account or password. If a client application supplies an invalid account name and password, the database server rejects the connection even if the trusted-host information contains an entry for the client computer.

To use trusted-host information for authentication, specify the `s=1` or `s=3` options in sqlhosts file entries. If you do not specify an `s` option, `s=3` is the default.

On Windows, the trusted-host file is in the \%WINDIR%\system32\drivers\etc directory.

On Linux and UNIX systems, the trusted-host file is in the $INFORMIXDIR/etc/ directory.

The hosts.equiv file has the following requirements:

- It must be owned by user **informix**
- It belong to group **informix**
- Permissions on the file must be restricted so that only user **informix** can modify the file. Using octal permissions, one of the following values is appropriate:
  - 644
  - 640
  - 444
  - 440

If you are using the hosts.equiv file and you use the **rlogind** daemon, you can execute the following statement on the client computer to determine whether the client is trusted:

```
rlogin hostname
```

If you log-in successfully without receiving a password prompt, the client is trusted. This method of determining if a client is trusted does not work when the file specified by the REMOTE_SERVER_CFG configuration parameter is used

Trusted-host file entries

To avoid an extra DNS lookup, specify the host name both with and without the domain name. For example, if the trusted host is named **host1** and it is in the domain **example.com**, then add the following entries to the trusted-host file:

```
#trustedhost        username
host1               informix
host1.example.com   informix
```

On some networks, the host name that a remote host uses to connect to a particular computer might not be the same as the host name that the computer uses to refer to itself. For example, the network host with the fully qualified domain name (FQDN) **host2.example.com** might refer to itself with the local host name **viking**. If this situation occurs, specify both host-name formats:

```
#trustedhost
host2.example.com
viking
```

Using the file specified by the REMOTE_SERVER_CFG configuration parameter instead of the hosts.equiv file

In the following situations, use the REMOTE_SERVER_CFG configuration parameter and the file that the parameter specifies:

- You need different trusted hosts for the database server than those listed for the OS.
- The security policies at your installation do not allow the use of hosts.equiv.
- You are a user of a non-root server instance and need to control which hosts are trusted.

To add entries to the file specified by the REMOTE_SERVER_CFG configuration parameter, you can manually enter the information or you can run the **admin()** or **task()** function with the cdr add trustedhost argument. If you run cdr add trustedhost argument with the **admin()** or **task()** function on a server in a high-availability cluster, the trusted-host information is added to the trusted-host files of all database servers in the cluster. Do not run the **admin()** or **task()** function with the cdr list trustedhost argument if you have manually entered trusted-host information on any of the database servers in a high-availability cluster or Enterprise Replication domain.

Related concepts:
Redirecting clients with the connectivity information
Related tasks:
Configuring secure connections for high-availability clusters
Related reference:
sqlhosts file and SQLHOSTS registry key options
Related information:
Creating sqlhost group entries for replication servers
INFORMIXSERVER environment variable
DBPATH environment variable
REMOTE_SERVER_CFG configuration parameter
S6_USE_REMOTE_SERVER_CFG configuration parameter

Trusted-user information

A user can list hosts from which they can connect as a trusted user in their .rhosts file.

The .rhosts file is located in the user's home directory on the computer housing the database server. To enable the trusted user authentication, specify `s=2` or `s=3` in the options in the sqlhosts entry. If you do not specify an `s` option, `s=3` is the default.

There may be reasons why a user's .rhosts file cannot be used. For example, a non-root installation might not have read access to a specific user's .rhosts file. You can specify an alternate filename by setting the REMOTE_USERS_CFG configuration parameter. If you set this parameter, the database server only has a single trusted-user file for all users.

Each line of the .rhosts file is a host from which the user can connect. You must specify server names both with and without domain names to avoid performing an extra DNS lookup. For example:

```
#trustedusers
xxx.example.com
xxx

yyy.example.com
yyy
```

The file specified by the REMOTE_USERS_CFG configuration parameter must be a combination of individual .rhosts files. Each single-line entry of the file has the following format:

```
hostname username
```

For example, suppose the following two .rhosts files existed for users **John** and **Fred**:
~john/.rhosts

```
#trustedhosts
xxx.example.com
xxx

yyy.example.com
yyy
```

~fred/.rhosts

```
#trustedhosts
xxx.example.com
xxx

zzz.example.com
zzz
```

John does not trust **zzz.example.com** or **zzz**, and **Fred** does not trust **yyy.example.com** or **yyy**.
The .rhosts files could be combined into a single file with the following format:

```
#trustedhost       username
xxx.example.com    john
xxx                john

yyy.example.com    john
yyy                john

xxx.example.com    fred
xxx                fred

zzz.example.com    fred
zzz                fred
```

Windows: A home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the **User Manager** application.

Related concepts:
Redirecting clients with the connectivity information
Related reference:
sqlhosts file and SQLHOSTS registry key options
Related information:
Creating sqlhost group entries for replication servers
INFORMIXSERVER environment variable
DBPATH environment variable
REMOTE_USERS_CFG configuration parameter

The netrc information

The netrc information is optional information that specifies identity data. A user who does not have authorization to access the database server or is not on a computer that is trusted by the database server can use this file to supply a name and password that are trusted. A user who has a different user account and password on a remote computer can also provide this information.

UNIX: The netrc information is in the .netrc file in the user's home directory. Use any standard text editor to prepare the .netrc file. The format of a netrc entry is:

```
machine machine_name login user_name password user_password
```

Windows: Use the Host Information tab of **setnet32** to edit the netrc information.

If you do not explicitly provide the user password in an application for a remote server (that is, through the USER clause of the CONNECT statement or the user name and password prompts in DB-Access), the client application looks for the user name and password in the netrc information. If the user explicitly specified the password in the application, or if the database server is not remote, the netrc information is not consulted.

The database server uses the netrc information regardless of whether it uses the default authentication policy or a communications support module.

For information about the specific content of this file, see your operating system documentation.

Windows only: On Windows, a home directory is not automatically assigned when the Windows administrator creates a user identity. The administrator can add a home directory to a user's profile with the User Manager application

- User impersonation
  The database server must impersonate the client to run a process or program on behalf of the client for certain client queries or operations.

Related reference:
sqlhosts file and SQLHOSTS registry key options

User impersonation

The database server must impersonate the client to run a process or program on behalf of the client for certain client queries or operations.

In order to impersonate the client, the database server must receive a password for each client connection. Clients can provide a user ID and password through the CONNECT statement or netrc information.

The following examples show how you can provide a password to impersonate a client.

netrc

```
machine trngpc3 login bruce password im4golf
```

CONNECT statement

```
CONNECT TO ol_trngpc3 USER bruce USING "im4golf"
```


The sqlhosts file and the SQLHOSTS registry key

IBM® Informix® client/server connectivity information, the *sqlhosts information*, contains information that enables a client application to find and connect to any IBM Informix database server on the network.

The default location of the sqlhosts file is:

UNIX:
    $INFORMIXDIR/etc/sqlhosts
Windows:
    %INFORMIXDIR%\etc\sqlhosts.%INFORMIXSERVER%

If you store the information in another location, you must set the INFORMIXSQLHOSTS environment variable.

If you set up several database servers to use distributed queries, use one of the following ways to store the sqlhosts information for all the databases:

- In one sqlhosts file, pointed to by the INFORMIXSQLHOSTS environment variable
- In separate sqlhosts files in each database server directory

- Creating the sqlhosts file with a text editor
  Each computer that hosts a database server or a client must have an sqlhosts file.
- Setting up the SQLHOSTS registry key with Setnet32 (Windows)
  A client application connects to the Informix database server that is running on a computer that can be reached through the network. To establish the connection, use **Setnet32** to specify the location of the Informix database server on the network and the network communications protocol to use. You must obtain this information from the administrator of the database server you want to use.

Related concepts:
Redirecting clients with the connectivity information
Related tasks:
Supporting multiplexed connections
Related reference:
Network protocol

Creating the sqlhosts file with a text editor

Each computer that hosts a database server or a client must have an sqlhosts file.

The sqlhosts file is located, by default, in the $INFORMIXDIR/etc directory. As an alternative, you can set the INFORMIXSQLHOSTS environment variable to the full path name and file name of a file that contains the sqlhosts information.

Open any standard text editor to create the sqlhosts file.

Note:

- Use white space (spaces, tabs, or both) to separate the fields.
- Do not include any spaces or tabs within a field.
- To put comments in the sqlhosts file, start a line with the comment character (#). You can also leave lines blank for readability.

Sample sqlhosts file

The following code block shows a sample sqlhosts file.

```
#dbservername    nettype        hostname       servicename       options
 menlo           onipcshm       valley         menlo
 newyork         ontlitcp       hill           dynsrvr2          s=2,b=5120
 payroll         onsoctcp       dewar          py1
 asia            group           -              -                e=asia.3
 asia.1          ontlitcp       node6          svc8              g=asia
 asia.2          onsoctcp       node0          svc1              g=asia
 portland        drsocssl       dewar          portland_serv
```


Setting up the SQLHOSTS registry key with Setnet32 (Windows)

A client application connects to the Informix® database server that is running on a computer that can be reached through the network. To establish the connection, use **Setnet32** to specify the location of the Informix database server on the network and the network communications protocol to use. You must obtain this information from the administrator of the database server you want to use.

If you specify a shared SQLHOSTS registry key, you must set the INFORMIXSQLHOSTS environment variable on your local computer to the name of the Windows computer that stores the registry. The database server first looks for the SQLHOSTS registry key on the INFORMIXSQLHOSTS computer. If the database server does not find an SQLHOSTS registry key on the INFORMIXSQLHOSTS computer, or if INFORMIXSQLHOSTS is not set, the database server looks for an SQLHOSTS registry key on the local computer.

You must comply with Windows network-access conventions and file permissions to ensure that the local computer has access to the shared SQLHOSTS registry key. For information about network-access conventions and file permissions, see your Windows documentation.

1. Double-click **Setnet32** in the folder that contains the Client SDK products.
   The Informix **Setnet32** window opens.
2. Click the Server Information tab to display the Server Information page, which has the following elements:
   - **Informix Server**
     Select an existing Informix database server or type the name of a new database server.
   - Host Name
     Select the host computer with the database server that you want to use or type the name of a new host computer.
   - Protocol Name
     Select a network protocol from a list of protocols that the installation procedure provides.
   - Service Name
     Specify the service name that is associated with a specific database server. Type either the service name or the port number that is assigned to the database server on the host computer. You must obtain this information from the database server administrator.

     Requirement: If you enter a service name, it must be defined on the client computer in the services file in the Windows installation directory. This file is in system32\drivers\etc\services. The service definition must match the definition on the database server host computer.
   - Options
     Enter options specific to the database server. For more information, see the *IBM Informix Administrator's Guide*.
   - Make Default Server
     Sets the INFORMIXSERVER environment variable to the name of the current database server to make it the default database server.
   - Delete Server
     Deletes the definition of a database server from the Windows registry. It also deletes the host name, protocol name, and service name associated with that database server.
3. Click OK to save the values.

The sqlhosts information

The sqlhosts information contains connectivity information for each database server and definitions for groups. The database server looks up the connectivity information when you start the database server, when a client application connects to a database server, or when a database server connects to another database server.

Each computer that hosts a database server or a client must include connectivity information.

In the sqlhosts file, each row contains the connectivity information for one database server, or the definition for one group.

- The connectivity information for each database server includes four fields of required information and one field for options.
- The group definition contains information in only three of the fields.

In the registry, the database server name is assigned to a key in the SQLHOSTS registry key, and the other fields are values of that key.

The following table summarizes the fields that are used for the SQLHOSTS information.

Field name in the sqlhosts file	Field name in the SQLHOSTS registry key	Description of connectivity information	Description of group information
dbservername	Database server name key or database server group key	Database server name	Database server group name
nettype	PROTOCOL	Connection type	The keyword `group`
hostname	HOST	Host computer for the database server	No information. Use a dash as a placeholder in this field.
servicename	SERVICE	Alias for the port number	No information. Use a dash as a placeholder in this field.
options	OPTIONS	Options that describe or limit the connection	Group options

UNIX: If you install IBM® Informix® Enterprise Gateway with DRDA in the same directory as the database server, your sqlhosts file also contains entries for the Gateway and non-Informix database servers. However, this section covers only the entries for the database server. For information about other entries in the sqlhosts file, see the *IBM Informix Enterprise Gateway with DRDA User Manual*.

- IANA standard service names and port numbers in the sqlhosts.std file
  The Internet Assigned Numbers Authority (IANA) assigns service names and port numbers for IBM Informix database servers.
- sqlhosts connectivity information
  Fields in the sqlhosts file or SQLHOSTS registry key describe connectivity information.
- Group information
  You define server groups in the sqlhosts file or SQLHOSTS registry key. When you create a server group, you can treat multiple related database server or Connection Manager SLA entries as a single entity for client connections to simplify connection redirection to database servers or Connection Managers. You must create group entries for database servers that participate in Enterprise Replication.
- Alternatives for TCP/IP connections
  The following topic describes some ways to bypass port and IP address lookups for TCP/IP connections.

Related concepts:
Strategies for increasing availability with Connection Managers
Related tasks:
Defining sqlhosts information for connection management
Configuring connection management
Connections that the database server supports
Configuring connectivity between Informix database servers and IBM Data Server clients
Defining sqlhosts information for connection management of high-availability clusters
Defining sqlhosts information for connection management of high-availability clusters that use secure ports
Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA)
Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA) and secure ports
Defining sqlhosts information for connection management of grids and replicate sets
Defining sqlhosts information for connection management of grids and replicate sets that use secure ports
Defining sqlhosts information for connection management high-availability replication systems
Defining sqlhosts information for connection management of high-availability replication systems that use secure ports
Defining sqlhosts information for connection management of server sets
Related information:
The syncsqlhosts utility

IANA standard service names and port numbers in the sqlhosts.std file

The Internet Assigned Numbers Authority (IANA) assigns service names and port numbers for IBM® Informix® database servers.

The services names and port numbers for database servers are:

Port/service	IANA code	Description
sqlexec	9088/tcp	SQL Interface
sqlexec-ssl	9089/tcp	SQL Interface - Encrypted

These service names are created in the sqlhosts.std file of . You are not required to change installed systems, because they continue to work with existing port numbers and service names. (Also, there is no guarantee that some other system is not already using the service names or port numbers assigned to .)

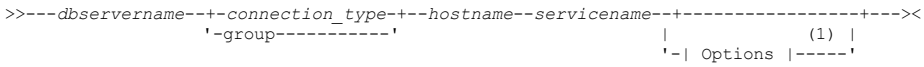
Organizations that have policies for following standards can use these service names and port numbers if they want the database server to be in compliance with the IANA standard. If another application that is installed on the same workstation already uses one of the service names or port numbers, you can ask the publisher of the non-compliant application to register for an IANA port number assignment to avoid the conflict. When applications are noncompliant, you can run using non-standard ports.

For more information, see the IANA organization website.

sqlhosts connectivity information

Fields in the sqlhosts file or SQLHOSTS registry key describe connectivity information.

Syntax



Notes:

1. See sqlhosts file and SQLHOSTS registry key options

Element	Purpose	Restrictions
dbservername	Names the database server for which the connectivity information is being specified. If specified with the group keyword instead of the connection type, names a group to treat multiple, related database server entries as one logical entry. You can use groups to establish or change client/server connections, or to simplify the redirection of connections to database servers.	The name must begin with a lowercase letter, and can contain lowercase letters, numbers, and underscore (_) symbols. The field length is limited to 128 bytes. The database server must exist. Its name must be specified by the DBSERVERNAME or DBSERVERALIASES configuration parameter in the onconfig file. A database server group cannot be nested inside another database server group. Database servers can be members of one group.
connection_type	Describes the type of connection that is made between the database server and the client application or another database server.	
hostname	Specifies the computer where the database server is located.	The field length is limited to 256 bytes. If the group keyword is specified, must be null (-).
servicename	Specifies the alias for the port number. The interpretation of the service name field depends on the type of connection in the connection-type field.	The field length is limited to 128 bytes. If the group keyword is specified, must be null (-).

dbservername field

Each database server across all of your associated networks must have a unique database server name.

If an sqlhosts file has multiple entries with the same dbservername, only the first one is used.

Connection-type field

The connection-type field is called `nettype` in the sqlhosts file and PROTOCOL in the SQLHOSTS registry key.

The following table summarizes the possible connection-type values for database server connections on different operating systems.

Table 1. Summary of connection-types

Values for UNIX	Values for Windows	Description	Connection type
drsocssl	drsocssl	Secured Sockets Layer (SSL) protocol for DRDA. You must configure a new server alias in the sqlhosts file or SQLHOSTS registry that uses drsoctcp connection protocol.	Network
drsoctcp	drsoctcp	Distributed Relational Database Architecture™ (DRDA) - connection for IBM Data Server Client. You must configure a new server alias in the sqlhosts file or SQLHOSTS registry that uses drsoctcp connection protocol.	Network
drtlitcp	drtlitcp	Distributed Relational Database Architecture (DRDA) - connection for IBM Data Server Client. You must configure a new server alias in the sqlhosts file or SQLHOSTS registry that uses drtlitcp connection protocol.	Network
onipcshm		Shared-memory communication. Requires the cfd option in the sqlhosts file if used for a non-root installation where the server and client are in different locations.	IPC
onipcstr		Stream-pipe communication. Requires the cfd option in the sqlhosts file if used for a non-root installation where the server and client are in different locations.	IPC
	onipcnmp	Named-pipe communication	IPC
ontlitcp		TLI with TCP/IP protocol	Network
onsocssl	onsocssl	Secured Sockets Layer (SSL) protocol	Network
onsoctcp	onsoctcp	Sockets with TCP/IP protocol	Network
onsocimc		Sockets with TCP/IP protocol for communication with Informix® MaxConnect	Network
ontliimc		TLI with TCP/IP protocol for communication with Informix MaxConnect	Network
onsqlmux	onsqlmux	Multiplexed connection	Network

Note: The connection-type values that begin with "on" can use "ol" in the place of "on". For example, either onipcshm or olipcshm specify shared-memory connections if used in the sqlhosts information.

Host name field

The host name is entered in the `hostname` field in the sqlhosts file, and in the HOST registry key.

If the connection type is `onsqlmux`, the `hostname` field must not be empty, but any specific value entered in it is ignored.

Following is an explanation of how client applications derive the values that are used in the host name field.

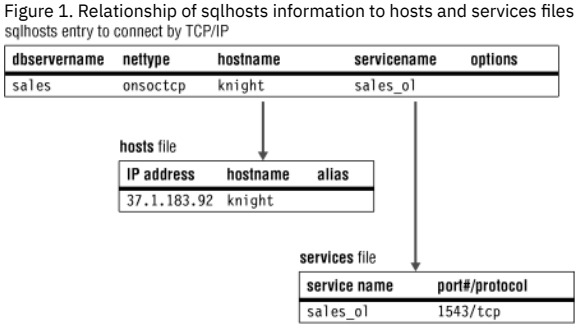
Network communication with TCP/IP
: When you use the TCP/IP connection protocol, the `host name` field is a key to the hosts file, which provides the network address of the computer. The name that you use in the `hostname` field must correspond to the name in the hosts file. In most cases, the host name in the hosts file is the same as the name of the computer.
In some situations, you might want to use the actual Internet IP address in the host name field.

UNIX: Shared-memory and stream-pipe communication
: When you use shared memory or stream pipes for client/server communications, the `hostname` field must contain the actual host name of the computer on which the database server is located.

Multiplexed connections
: When you use `onsqlmux` as the connection type, the `hostname` field must have an entry, but the entry is ignored. Dashes (–) can be used as entries.

Service name field

Network communication with TCP/IP
: The service name field is called `servicename` on the UNIX operating system and SERVICE on the Windows operating system. When you use the TCP/IP connection protocol, the service name entry must correspond with the name in the services file. The port number in the services file tells the network software how to find the database server on the specified host.
The following figure shows the relationship between the sqlhosts information and the hosts file, and the relationship of sqlhosts information to the services file.
Figure 1. Relationship of sqlhosts information to hosts and services files



In some cases, you might use the actual TCP listen-port number in the service name field.

Windows: Named-pipe communication
: For a named-pipe connection (`onipcnmp`), the SERVICE entry can be any short group of letters that is unique in the environment of the host computer where the database server is located.

UNIX: Shared-memory and stream-pipe communication
: For a shared-memory connection (`onipcshm`) or a stream-pipe connection (`onipcstr`), the database server uses the value in the *servicename* entry internally to create a file that supports the connection. For both `onipcshm` and `onipcstr` connections, the *servicename* can be any short group of letters that is unique in the environment of the host computer where the database server is located.
Tip: Use the *dbservername* as the *servicename* for stream-pipe connections.

Multiplexed connections
: For multiplexed connections (`onsqlmux`), the `hostname` field must have an entry, but the entry is ignored. Dashes (-) can be used as entries.

- sqlhosts file and SQLHOSTS registry key options
  You can include server options and group options in the sqlhosts file or SQLHOSTS registry key.

Related tasks:
Configuring secure connections for high-availability clusters
Changing client connectivity information
Connections that the database server supports
Related reference:
Configuration parameters related to connectivity
Group information
Stream-pipe connections (UNIX and Linux)
Specifying Network Connections
IBM Informix MaxConnect
TCP/IP connectivity files
Alternatives for TCP/IP connections
Related information:
Connectivity protocols

sqlhosts file and SQLHOSTS registry key options

You can include server options and group options in the sqlhosts file or SQLHOSTS registry key.

The following syntax fragments show the server options. The syntax fragment for group options is described in a section after the server options.

Important: Options must be separated by commas, but the first option that is listed in each sqlhosts entry must not have a comma before it.

Server options

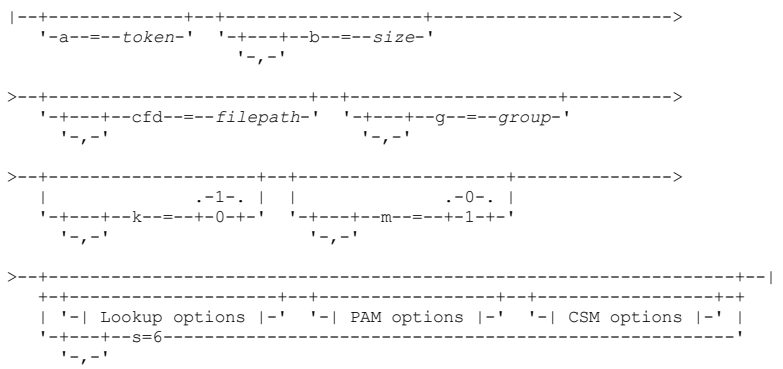


Table 1. Server options in the sqlhosts file and SQLHOSTS registry key.

Element	Purpose	Restrictions
a	Stores the authentication token that is required for connecting to the Informix® Warehouse Accelerator. This entry is created by Informix during Informix Warehouse Accelerator connection setup.	Important: Do not manually change this option.
b	Specifies, in bytes, the size of the communications buffer space for TCP/IP connections.	The maximum buffer size supported is 32 KB.
c	Enables confidentiality service for the Generic Security Services CSM, which supports single sign-on (SSO). Data transmitted to and from the SSO-authenticated user is encrypted and can be viewed only by the user that is logged in with the authorized credentials.	• **c=1** enables the service (default) • **c=0** disables the service
cfd	Indicates the storage location for communication files that are used in shared-memory and stream-pipe connections.	The length of the **cfd** path is restricted to 70 bytes. Relative-path byte lengths include $INFORMIXDIR.
csm	Describes the communication support module (CSM) for each database server that uses a CSM.	CSM entries must be specified in the concsm.cfg file.
ENCCSM	The name of the encryption communication support module.	The ENCCSM must be specified in the concsm.cfg file. You cannot use an ENCCSM with • Enterprise Replication and high-availability clusters • A multiplexed connection • A simple password CSM (SPWDCSM)
g	Specifies the name of the group to which the database server belongs.	The group must be defined.
GSSCSM	The name of the generic security services communications support module for single sign-on (SSO) authentication.	The GSSCSM must be specified in the concsm.cfg file. Cannot be used for Enterprise Replication and high-availability clusters.

Element	Purpose	Restrictions
i	Enables the integrity service for the Generic Security Services CSM, which supports single sign-on (SSO).	• **i=1** enables the service (default) • **i=0** disables the service
k	Enables the network service to check periodically whether the connection between the client and server is still active. If the connection is found to be broken the network service frees resources.	Only available for TCP/IP connections.
m	Enables the database server to create multiple database connections without using up the additional computer resources that are required for more network connections.	• Multithreaded client connections, shared-memory connections, and connections to subordinate database servers are not supported. • The Informix ESQL/C sqlbreak() function is not supported. • Cannot be used with a CSM.
p	Enables and disables the simple password CSM, which provides password encryption.	• **p=0** password is not required (default) • **p=1** password is required
r	Enables the control of operating-system security-file lookups to control the way that a client (user) gains access to a database server. The **s** option identifies database server-side settings, and the **r** option identifies client-side settings.	The database server ignores **r** settings.
s	Enables the control of operating-system security-file lookups to control the way that a client (user) gains access to a database server. The **s** option identifies database server-side settings, and the **r** option identifies client-side settings.	A client ignores **s** settings.
pam_serv	Gives the name of a PAM service that a database is using.	Must be used with **s=4** option.
pamauth	Describes the authorization method that is used by the PAM service.	Must be used with **s=4** option.
SPWDCSM	The name of the simple password communication support module	The SPWDCSM must be specified in the concsm.cfg file. You cannot use an SPWDCSM with • Enterprise Replication and high-availability clusters • A multiplexed connection • An encryption CSM (ENCCSM)

The following syntax fragment shows the group options in the sqlhosts file.

```
Group options

|--+-------------+--+---------------------+--------------------->
   |       .-0-. |  '-+---+--e--=--server-'
   '-c--=--+-1-+-'    '-,-'

>--+-------------------------+---------------------------------|
   '-+---+--i--=--identifier-'
     '-,-'
```

Table 2. Group options in the sqlhosts file and SQLHOSTS registry key.

Element	Purpose	Restrictions
c	Controls connection redirection. Indicates the order in which client applications choose database servers, or the aliases within a database server group.	
e	Specifies a database server name that marks the end of a database server group.	
i	Assigns an identifying number to a database server group.	The identifier must be a positive integer from 1 through 32767 and must be unique within your network environment. The **i** option is required for Enterprise Replication.

Usage

When you change option values in an sqlhosts entry, those changes affect the next connection that a client application makes. The server automatically recognizes any changes that are made.

The database server evaluates the options entries as a series of columns. A comma or white space in an options entry represents an end of a column. Client and database server applications check each column to determine whether the option is supported.

You can combine multiple options in each entry, and you can include them in any order. The maximum length for an options entry is 256 bytes.

Attention: Unsupported or incorrect options do not trigger a notification.

Buffer option (b)

The **b** option applies only to connections that use the TCP/IP connection protocol. Other types of connections ignore the **b** option.

You can adjust the buffer size to use system and network resources more efficiently; however, if the buffer size is set too high, the user receives a connection-reject error because no memory can be allocated. For example, if you set **b=16000** on a system that has 1000 users, the system might require 16 megabytes of memory for the communications buffers. This setting might exhaust the memory resources of the computer. The default buffer size for the database server for TCP/IP is 4096 bytes.

If your network includes several different types of computers, be careful when you change the size of the communications buffer.
Tip: Use the default size for the communications buffer. If you choose to set the buffer size to a different value, set the client-side communications buffer and the database server-side communications buffer to the same size.

Group connection-redirection option (c)

The **c** option is valid only for servers that are assigned to a server group.

Use the **c** option to:

- Balance the load across multiple database server instances.
- Set High-Availability Data Replication (HDR) to transfer over to a backup database server in the event of a failure.

Table 3. Settings for the connection-redirection option.

Setting	Result
c=0	This is the default setting. Client applications connect to the first database server instance listed in the server group in the sqlhosts information. If the client cannot connect to the first instance, it attempts to connect to the second instance, and so on.
c=1	Client applications choose a random starting point from which to connect to a database server instance in a server group.

Communication files directory option (cfd)

You can use the communication files directory option to store shared-memory or stream-pipe connection communication files in a new location. Specifying the communication files directory option for non-root installations of Informix is necessary if the server and client are in different locations, and increases system performance if the server and client are in the same location.

The **cfd** option can define an absolute path or a path relative to $INFORMIXDIR for storing communication files:

- **cfd=/***location* defines an absolute path
- **cfd=***location* defines a path relative to $INFORMIXDIR

The length of the **cfd** path is restricted to 70 bytes. Relative-path byte lengths include $INFORMIXDIR.

Non-root installations of Informix do not have permission to write to the /INFORMIXTMP directory, so shared-memory and stream-pipe connection communication files are written to the $INFORMIXDIR/etc directory if no communication files directory is specified as an option in the sqlhosts information.

Important: This option must be defined for non-root installations of Informix, where the server and client are in different locations, or the connection fails.

Communication support module option (csm)

The format of the CSM option is **csm=(***name***,***options***)**.

The value of *name* must match a name entry in the concsm.cfg file.

CSM options that are defined in the sqlhosts file override options that are specified in the concsm.cfg file. CSM encryption options cannot be specified in the sqlhosts information.

If you do not specify the **csm** option, the database server uses the default authentication policy for that database server.

Note: The **s=7** option is deprecated and not required for the Single Sign-On (SSO) CSM.

End of group option (e)

If no **e** option is specified for a group, but all sqlhosts entries specify either groups or group members, the network must scan the entire file. You can use the **e** option to specify the end of a server group, and improve system performance. The network layer scans the sqlhosts file until the entry specified by the **e** option is read.

If no end-of-group option is specified for a group, the group members are assumed to be contiguous. The end of the group is determined when an entry is reached that does not belong to the group, or at the end of file, whichever comes first.

In the following example, the **e** option specifies entry lx3, so entry lx4 is not scanned by the network layer.

```
#dbservername   nettype   hostname   servicename   options
g_x1            group     -          -             i=10,e=lx3
lx1             onsoctcp  apollo11   9810          g=g_x1
lx2             onsoctcp  apollo12   9820          g=g_x1
lx3             onsoctcp  apollo13   9830          g=g_x1

lx4             onsoctcp  apollo14   9840
```

Keep-alive option (k)

This option enables the network service to check periodically whether the connection between the client and server is still active. If the receiving end of the connection does not respond within the time that is specified by the parameters of your operating system, the network service immediately detects the broken connection and frees resources.

Table 4. Settings for the keep-alive option

Setting	Result
k=0	Disables this service
k=1	Enables this service (default)

Multiplex option (m)

This option enables the database server to create multiple database connections to client applications without using up the additional computer resources that are required for more network connections. You must restart the server after you enable this service.

Table 5. Settings for the multiplex option

Setting	Result
m=0	Disables this service (default)
m=1	Enables this service

PAM options (pam_serv, pam_auth, s=4)

The database server provides an interface to use pluggable authentication modules (PAM) for session authentication. To configure this interface, supply the PAM service name and the authentication method. Authentication can be the connection password or a user challenge that requires the user to answer a question. Distributed Relational Database Architecture (DRDA) connections for IBM Data Server clients can use password authentication, but not challenge authentication.

Informix PAM authentication calls the **pam_authenticate()** and **pam_acct_mgmt()** functions.

Table 6. Settings for PAM services

Option	Description	Settings
pam_serv	The name of the PAM service that the database server is using.	PAM services typically are in the /usr/lib/security directory and parameters are listed in the **/etc/pam.conf** file. In Linux, the **/etc/pam.conf** file can be replaced with a directory called /etc/pam.d, where there is a file for each PAM service. If /etc/pam.d exists, /etc/pam.conf is ignored by Linux.
pamauth	The method of authentication that is used by the PAM service. An application must be designed to respond to the challenge prompt correctly before connecting to the database server.	**pamauth=password** uses the connection request password for authentication. **pamauth=challenge** authentication requires a correct user reply to a question or prompt
s=4	Enables PAM authentication.	

Trusted-host and trusted-user lookup options (s)

With these security options, you can specifically enable or disable the use of either or both files.

Table 7. Settings for trusted-host and trusted-user lookup.

Setting	Result
s=0	Disables trusted-hosts lookup in hosts.equiv or the file specified by the REMOTE_SERVER_CFG configuration parameter. Disables trusted-user lookup in rhosts files or the file specified by the REMOTE_USERS_CFG configuration parameter. Only incoming connections with passwords are accepted. Cannot be used for distributed database operations.
s=1	Enables trusted-hosts lookup in hosts.equiv or the file specified by the REMOTE_SERVER_CFG configuration parameter. Disables trusted-user lookup in rhosts files or the file specified by the REMOTE_USERS_CFG configuration parameter.
s=2	Disables trusted-hosts lookup in hosts.equiv or the file specified by the REMOTE_SERVER_CFG configuration parameter. Enables trusted-user lookup in rhosts files or the file specified by the REMOTE_USERS_CFG configuration parameter. Cannot be used for distributed database operations.
s=3	Enables trusted-hosts lookup in hosts.equiv or the file specified by the REMOTE_SERVER_CFG configuration parameter. Enables trusted-user lookup in rhosts files or the file specified by the REMOTE_USERS_CFG configuration parameter. (default)

Secure connections for clusters option (s=6)

The **s=6** option in the sqlhosts information ensures that the connections between cluster servers are trusted. Secure ports that are listed in the sqlhosts information can be used only for cluster communication. Client applications cannot connect to secure ports.

Table 8. The secure connection option for clusters.

Setting	Result
s=6	Configures Enterprise Replication and High Availability Connection Security. Cannot be used with any other **s** option.

netrc lookup options (r)

With r options, you can enable or disable netrc lookup.

Table 9. Settings for netrc lookup options.

Setting	Result
r=0	netrc lookup is disabled.
r=1	netrc lookup is enabled (default)

Related concepts:
Trusted-host information
Trusted-user information
Related tasks:
Supporting multiplexed connections
Configuring secure connections for high-availability clusters
Related reference:
Group information
The netrc information
Related information:
Pluggable authentication modules (UNIX or Linux)
Communication support modules for data transmission encryption
Configuring the IBMInformix instance for SSO
Simple password encryption
Configuring secure ports for connections between replication servers

Group information

You define server groups in the sqlhosts file or SQLHOSTS registry key. When you create a server group, you can treat multiple related database server or Connection Manager SLA entries as a single entity for client connections to simplify connection redirection to database servers or Connection Managers. You must create group entries for database servers that participate in Enterprise Replication.

You can use the name of a group instead of the database server name in the following environment variables, or in the SQL CONNECT command:

- The value of the INFORMIXSERVER environment variable for a client application can be the name of a group. However, you cannot use a group name as the value of the INFORMIXSERVER environment variable for a database server or database server utility.
- The value of the DBPATH environment variable can contain the names of groups.

Use a dash (-) character (ASCII 45) for hostname and server/port values when you specify a connection information for a group.

High-availability cluster groups

A high-availability cluster groups sqlhosts have the following format:

```
#dbservername   nettype   hostname     servicename          options
 group_name     group     -            -                    c=1,e=member_name_n
 member_name_1  protocol  host_name_1  service_or_port_1    g=group_name
 member_name_2  protocol  host_name_2  service_or_port_2    g=group_name
 member_name_n  protocol  host_name_n  service_or_port_n    g=group_name
```

`c=1` is optional, and specifies that a random starting point in the list of group members is used for connection attempts. `e=member_name` is optional, and specifies the final entry for group members, so that the entire file is not scanned. The `g=group_name` option is required for group members, and specifies the group that the member belongs to.

Enterprise Replication server groups

All database servers that participate in replication must be a member of a database server group. Each database server in the enterprise must have a unique identifier. Enterprise Replication node groups have the following sqlhosts format:

```
#dbservername   nettype   hostname     servicename          options
 group_name_1   group     -            -                    i=identifier_1,e=member_name_1
 member_name_1  protocol  host_name_1  service_or_port_1    g=group_name_1

 group_name_2   -         -                                 i=identifier_2,e=member_name_2
 member_name_2  protocol  host_name_2  service_or_port_2    g=group_name_2

 group_name_n   -         -                                 i=identifier_n,e=member_name_n
 member_name_n  protocol  host_name_n  service_or_port_n    g=group_name_n
```

The `i=identifier` is required for Enterprise Replication. `e=member_name` is optional, and specifies the final entry for group members, so that the entire file is not scanned. The `g=group_name` option is required for group members, and specifies the group that the member belongs to.

Connection Manager service-level agreement groups

Connection Manager SLA groups have the following sqlhosts format:

```
#dbservername          nettype   hostname   servicename              options
SLA_1_group_name       group     -          -                        c=1,e=SLA_name_1_from_CM_n
SLA_name_1_from_CM_1   protocol  CM_1_host  CM_1_port_or_service_1   g=SLA_1_group_name
SLA_name_1_from_CM_2   protocol  CM_2_host  CM_2_port_or_service_1   g=SLA_1_group_name
SLA_name_1_from_CM_n   protocol  CM_n_host  CM_n_port_or_service_1   g=SLA_1_group_name

SLA_2_group_name       group     -          -                        c=1,e=SLA_name_2_from_CM_n
SLA_name_2_from_CM_1   protocol  CM_1_host  CM_1_port_or_service_2   g=SLA_2_group_name
SLA_name_2_from_CM_2   protocol  CM_2_host  CM_2_port_or_service_2   g=SLA_2_group_name
SLA_name_2_from_CM_n   protocol  CM_n_host  CM_n_port_or_service_2   g=SLA_2_group_name

SLA_n_group_name       group     -          -                        c=1,e=SLA_name_n_from_CM_n
SLA_name_n_from_CM_1   protocol  CM_1_host  CM_1_port_or_service_n   g=SLA_n_group_name
SLA_name_n_from_CM_2   protocol  CM_2_host  CM_2_port_or_service_n   g=SLA_n_group_name
SLA_name_n_from_CM_n   protocol  CM_n_host  CM_n_port_or_service_n   g=SLA_n_group_name
```

`C=1` is optional, and specifies that a random starting point in the list of group members is used for connection attempts. `e=member_name` is optional, and specifies the final entry for group members, so that the entire file is not scanned. The `g=group_name` option is required for group members, and specifies the group that the member belongs to.

- Creating a group in the sqlhosts file
  You can define a group and the members of the group by adding entries to the sqlhosts file.

Related reference:
sqlhosts connectivity information
sqlhosts file and SQLHOSTS registry key options

Creating a group in the sqlhosts file

You can define a group and the members of the group by adding entries to the sqlhosts file.

To create a database server group in the sqlhosts file:

1. Add an entry to define the database server group:

   dbservername
   : The name of the group. The name must begin with a lowercase letter, and can contain lowercase letters, numbers, and underscore (_) symbols.

   nettype
   : The word group.

   hostname
   : A dash (-) character (ASCII 45), to indicate that the field value is null.

   servicename
   : A dash (-) character (ASCII 45), to indicate that the field value is null.

   options
   : The c, e, or i options, as appropriate.

2. Add one or more entries for members of the group. Include the g=*group* option.

Example

The following example shows definition of a group named **g_asia**. The group contains four members.

```
#dbservername  nettype   hostname  servicename  options
g_asia         group     -         -            c=1,e=manilla
tokyo          onsoctcp  node_1    service_1    g=g_asia
beijing        onsoctcp  node_2    service_2    g=g_asia
seoul          onsoctcp  node_3    service_4    g=g_asia
manilla        onsoctcp  node_4    service_5    g=g_asia
```


Alternatives for TCP/IP connections

The following topic describes some ways to bypass port and IP address lookups for TCP/IP connections.

IP addresses for TCP/IP connections

For TCP/IP connections (both TLI and sockets), you can use the actual IP address in the **hostname** field instead of the host name or alias found in the hosts file. The following example shows sample IP addresses and hosts from a hosts file.

```
#address       hostname  alias
555.12.12.12   smoke
```

```
98.555.43.21    odyssey
12.34.56.555    knight    sales
```

Using the IP address for **knight** from the table, the following two sqlhosts entries are equivalent:

```
#dbservername    nettype    hostname        servicename    options
sales            ontlitcp   12.34.56.789    sales_ol
```

```
#dbservername    nettype    hostname        servicename    options
sales            ontlitcp   knight          sales_ol
```

Using an IP address might speed up connection time in some circumstances. However, because computers are usually known by their host name, using IP addresses in the host name field makes it less convenient to identify the computer with which an entry is associated.

UNIX: You can find the IP address in the net address field of the hosts file, or you can use the UNIX **arp** or **ypmatch** command.
Windows: You can configure Windows to use either of the following mechanisms to resolve a domain to an IP address:

- Windows Internet Name Service
- Domain Name Server

Wildcard addressing for TCP/IP connections

You can use wildcard addressing in the **hostname** field of the hosts file when both of the following conditions are met:

- You are using TCP/IP connections.
- The computer where the database server is located uses multiple network-interface cards (NICs).

If the preceding conditions are met, you can use an asterisk (*) as a *wildcard* in the **hostname** field that the database server uses. When you enter a wildcard in the **hostname** field, the database server can accept connections at any valid IP address on its host computer.

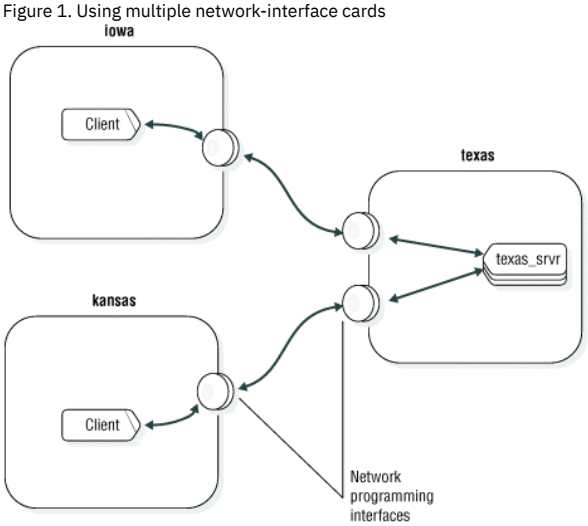
Each IP address is associated with a unique host name. When a computer uses multiple NICs, as in the following table, the hosts file must have an entry for each interface card. For example, the hosts file for the **texas** computer with two NICs might include these entries.

```
#address        hostname   alias
123.45.67.81    texas1
123.45.67.82    texas2
```

If the client application and database server share the sqlhosts information, you can specify both the wildcard and a host name or IP address in the **hostname** field (for example, `*texas1` or `*123.45.67.81`). The client application ignores the wildcard and uses the host name (or IP address) to make the connection, and the database server uses the wildcard to accept a connection from any IP address.

The wildcard format allows the listen thread of the database server to wait for a client connection using the same service port number on each of the valid network-interface cards. However, waiting for connections at multiple IP addresses might require more processor time than waiting for connections with a specific host name or IP address.

The following figure shows a database server on a computer named **texas** that has two network-interface cards. The two client sites use different network cards to communicate with the database server.
Figure 1. Using multiple network-interface cards



The following examples show potential sqlhosts connectivity information for the **texas_srvr** database server.

```
#dbservername    nettype    hostname        servicename    options
texas_srvr       ontlitcp   *texas1         pd1_on
```

```
#dbservername    nettype    hostname        servicename    options
texas_srvr       ontlitcp   *123.45.67.81   pd1_on
```

```
#dbservername    nettype    hostname        servicename    options
texas_srvr       ontlitcp   *texas2         pd1_on
```

```
#dbservername    nettype    hostname        servicename    options
texas_srvr       ontlitcp   *123.45.67.82   pd1_on
```

```
#dbservername    nettype    hostname        servicename    options
texas_srvr       ontlitcp   *               pd1_on
```

If the connectivity information corresponds to any of the preceding lines, the **texas_srvr** database server can accept client connections from either of the network cards. The database server finds the wildcard in the **hostname** field and ignores the explicit host name.
Tip: For clarity and ease of maintenance, include a host name when you use the wildcard in the host name field (that is, use `*host` instead of `*`).
The connectivity information used by a client application must contain an explicit host name or IP address. The client applications on **iowa** can use any of the following host names: `texas1`, `*texas1`, `123.45.67.81`, or `*123.45.67.81`. If there is a wildcard (*) in the `hostname` field, the client application ignores it.

The client application on **kansas** can use any of the following host names: `texas2`, `*texas2`, `123.45.67.82`, or `*123.45.67.82`.

Port numbers for TCP/IP connections

For the TCP/IP network protocol, you can use the actual TCP listen port number in the service name field.

For example, if the port number for the **sales** database server in the services file is `1543`, you can write an entry in the sqlhosts file as follows:

```
#dbservername   nettype    hostname    servicename    options
sales           ontlitcp   knight      1543
```

Using the actual port number might save time when you make a connection in some circumstances. However, as with the IP address in the `hostname` field, using the actual port number might make administration of the connectivity information less convenient.

Related reference:
sqlhosts connectivity information

Informix support for IPv6 addresses

On all platforms, IBM® Informix® recognizes Internet Protocol Version 6 (IPv6) addresses, which are 128 bits long, and Internet Protocol Version 4 (IPv4) addresses, which are 32 bits long.

Beginning with 10.00.xC4 and Client SDK 2.90.xC4, the database server checks, on startup, whether IPv6 is supported in the underlying operating system. If IPv6 is supported it is used. If the underlying operating system does not support IPv6, the IPv4 address is used. and Client SDK retrieve the IP address from the name service.

You can treat that runs on a host with both IPv4 and IPv6 addresses the same way you treat a server running on a multi-homed host. You can configure on a host with both IPv4 and IPv6 addresses in either of the following ways:

- Create aliases (using the DBSERVERALIASES configuration parameter) and assign an IPv6 address to one of them and an IPv4 address to the other.
- Instruct the to listen on all the IP addresses configured on the host by using a wild-carded hostname in the sqlhosts file.
  For example:

  ```
  #dbservername   nettype    hostname    servicename    options
  olserver1       onsoctcp   *myhost     onservice1
  ```

  Starting with Version 10.0, the host name entry in the SQLHOSTS file maps to an IPv6 address if the host has a configured IPv6 address. If the host does not have a configured IPv6 address, the hostname entry maps to an IPv4 address.

Disabling IPv6 Support

also provides a way to disable IPv6 support when working in IPv4 environments.

To disable IPv6 support for all database instances and client applications:

- Create an empty file $INFORMIXDIR/etc/IFX_DISABLE_IPV6.

The file must have read permission for user **informix**. The file is not read from or written to, and is not required to contain any data.
To disable IPv6 support for a single database instance or for a single client application:

- On the database server instance, or on the workstation on which applications are run, create an environment variable named IFX_DISABLE_IPV6 and set its value to **yes**, as in:

  ```
  IFX_DISABLE_IPV6=yes
  ```


Configuration parameters related to connectivity

Some of the configuration parameters in the onconfig file specify information related to connectivity.

When you restart the database server, the restart procedure uses the values that you set in these configuration parameters.

The following configuration parameters are related to connectivity:

- DBSERVERNAME
- DBSERVERALIASES
- LIMITNUMSESSIONS
- NETTYPE
- NS_CACHE
- NUMFDSERVER

- HA_ALIAS

UNIX: When you configure connectivity, also consider setting the LISTEN_TIMEOUT and MAX_INCOMPLETE_CONNECTIONS configuration parameters. These parameters can reduce the risk of a hostile denial-of-service (DOS) attack by making it more difficult to overwhelm the Listener VP that handles connections. For more information, see the *IBM Informix Security Guide*.

- Connection information set in the DBSERVERNAME configuration parameter
  When a client application connects to a database server, it must specify the name for the database server. The sqlhosts information that is associated with the specified database server name describes the type of connection between the application and the database server.
- Connection information set in the DBSERVERALIASES configuration parameter
  The DBSERVERALIASES configuration parameter lets you assign additional dbserver names to the same database server.
- Connection information set in the LIMITNUMSESSIONS configuration parameter
  The LIMITNUMSESSIONS configuration parameter is an optional parameter that specifies the maximum number of sessions that you want connected to IBM® Informix®. If you specify a maximum number, you can also specify whether you want to print messages to the online.log file when the number of sessions approaches the maximum number.
- Connection information set in the NETTYPE configuration parameter
  The NETTYPE configuration parameter lets you adjust the number and type of virtual processors the database server uses for communication. Each type of network connection (for example, ipcshm or soctcp) can have a separate NETTYPE entry in the configuration file.
- Name service maximum retention time set in the NS_CACHE configuration parameter
  The NS_CACHE configuration parameter defines the maximum retention time for an individual entry in the host name/IP address cache, the service cache, the user cache, and the group cache. If you specify maximum retention times, the database server gets host, service, user, and group database server information from the cache.
- Connection information set in the NUMFDSERVERS configuration parameter
  For network connections on UNIX, use the NUMFDSERVERS configuration parameter to specify the maximum number of poll threads to handle network connections migrating between Informix virtual processors (VPs).
- Connection information set in the HA_ALIAS configuration parameter
  The HA_ALIAS configuration parameter defines a network alias that is used for server-to-server communication in a high-availability cluster. The specified network alias is also used by Connection Managers, the **ifxclone** utility, and **onmode -d** commands.

Related tasks:
Connections that the database server supports
Related reference:
sqlhosts connectivity information

Connection information set in the DBSERVERNAME configuration parameter

When a client application connects to a database server, it must specify the name for the database server. The sqlhosts information that is associated with the specified database server name describes the type of connection between the application and the database server.

For example, to assign the name **nyc_research** to a database server, set the DBSERVERNAME value in the onconfig file or Windows registry key:

```
DBSERVERNAME nyc_research
```

Client applications specify the name of the database server in one of the following places:

- In the INFORMIXSERVER environment variable
- In SQL statements such as CONNECT, DATABASE, CREATE TABLE, and ALTER TABLE, which specify a database environment
- In the DBPATH environment variable

The DBSERVERNAME must specify either the database server name or one of the database server aliases. The name must begin with a lowercase letter and can contain other lowercase letters, digits, and underscores. The name must not include uppercase characters, a field delimiter (space or tab), or a new line character. Other characters from the basic ASCII code set are not necessarily reliable. For example, a hyphen or minus sign can create problems and a colon might not work reliably. The @ character is reserved to separate the database from the server (as in dbase@server).

For `onimcsoc` or `onsoctcp` protocols, you can update the DBSERVERNAME configuration parameter to include the number of multiple listen threads for the database server aliases in your sqlhosts information, as follows:

```
DBSERVERNAME name-number_of_multiple_listen_threads
```

You can configure DBSERVERALIASES connections as SSL connections, and you can have a mix of SSL and non-SSL connections.

Related information:
DBSERVERNAME configuration parameter

Connection information set in the DBSERVERALIASES configuration parameter

The DBSERVERALIASES configuration parameter lets you assign additional dbserver names to the same database server.

The maximum number of aliases is 32. The following example shows entries in an onconfig configuration file that assign three dbserver names to the same database server instance.

```
DBSERVERNAME          sockets_srvr
DBSERVERALIASES       ipx_srvr,shm_srvr
```

Because each dbserver name has a corresponding sqlhosts entry, you can associate multiple connection types with one database server.

```
shm_srvr        onipcshm    my_host           my_shm
sockets_srvr    onsoctcp    my_host           port1
ipx_srvr        ontlispx    nw_file_server    ipx_srvr
```

Using the sqlhosts file shown in the previous example, a client application uses the following statement to connect to the database server using shared-memory communication:

```
CONNECT TO '@shm_srvr'
```

A client application can initiate a TCP/IP sockets connection to the *same* database server using the following statement:

```
CONNECT TO '@sockets_srvr'
```

DBSERVERALIASES must begin with a lowercase letter and can contain other lowercase letters, digits, and underscores. DBSERVERALIASES must not include uppercase characters, a field delimiter (space or tab), or a new line character. Other characters from the basic ASCII code set are not necessarily reliable. For example, a hyphen or minus sign can create problems and a colon might not work reliably. The @ character is reserved to separate the database from the server (as in dbase@server).

In the previous examples, the `@shm_srvr` statement connects to an unidentified database at that server; alternatively, you can connect to `dbase1@shm_srvr`.

For **onimcsoc** or **onsoctcp** protocols, you can update the DBSERVERALIASES configuration parameter to include the number of multiple listen threads for the database server aliases in your **sqlhosts** information, as follows:

```
DBSERVERALIASESname-number,name-number
```

You can configure DBSERVERALIASES connections as SSL connections, and you can have a mix of SSL and non-SSL connections.

Related information:
DBSERVERALIASES configuration parameter

Connection information set in the LIMITNUMSESSIONS configuration parameter

The LIMITNUMSESSIONS configuration parameter is an optional parameter that specifies the maximum number of sessions that you want connected to IBM® Informix®. If you specify a maximum number, you can also specify whether you want to print messages to the online.log file when the number of sessions approaches the maximum number.

Distributed queries against a server are counted against the limit.

You might be required to dynamically increase or temporarily turn off the LIMITNUMSESSIONS configuration parameter to allow administrative utilities to run if the database server is reaching the limit. Use **onmode -wf** or **onmode -wm** to dynamically increase or turn off LIMITNUMSESSIONS.

If the LIMITNUMSESSIONS configuration parameter is enabled and sessions are restricted because of this limit, both regular user threads and DBSA user threads connecting to any database count against the limit. However, a DBSA user is allowed to connect to the server even after the limit is reached.

The LIMITNUMSESSIONS configuration parameter is not intended to be used as a means to adhere to license agreements.

Example

The following example specifies that you want a maximum of 100 sessions to connect to the database server and you want to print a warning message when the number of connected sessions approaches 100: `LIMITNUMSESSIONS 100,1`

Connection information set in the NETTYPE configuration parameter

The NETTYPE configuration parameter lets you adjust the number and type of virtual processors the database server uses for communication. Each type of network connection (for example, ipcshm or soctcp) can have a separate NETTYPE entry in the configuration file.

Recommendation: Although the NETTYPE parameter is not a required parameter, you must set NETTYPE if you use two or more connection types. After the database server is running for some time, you can use the NETTYPE configuration parameter to tune the database server for better performance.
For more information about NETTYPE, see Network virtual processors. For information about the NETTYPE configuration parameter, see the *IBM® Informix® Administrator's Reference*.

Name service maximum retention time set in the NS_CACHE configuration parameter

The NS_CACHE configuration parameter defines the maximum retention time for an individual entry in the host name/IP address cache, the service cache, the user cache, and the group cache. If you specify maximum retention times, the database server gets host, service, user, and group database server information from the cache.

Each cache entry expires either after the time configured for the specific cache or when the time is reconfigured.

Usually the network name service provider (for example, DNS) is on a remote computer. To avoid spending the time required to return information from the network name service provider, you can use the NS_CACHE configuration parameter to specify the maximum retention times for obtaining information from one of the internal caches. Then Informix® looks for information in the cache. If the information is not there, the database server queries the operating system for the information.

You can avoid many of these operating system lookups by using the Informix name service caching mechanism, which can keep and reuse each retrieved piece of information for a configurable amount of time.

The server can get information from the cache faster than it does when querying the operating system. However, if you disable one or more of these caches by setting the retention time to 0, the database server queries the operating system for the host, service, user, or group information.

As a DBA, you might want to modify the NS_CACHE configuration parameter settings if the network name service provider runs on a remote computer or the MSC VP is running with a large amount of processor usage.

For example, you can run the **onstat -g glo** command to check the `msc` VP usage in the `Individual virtual processors` portion of the output. In the following ouput sample, the `msc` processor usage, shown in the `usercpu` and `syscpu` columns is high. If you suspect the usage is high because the DNS call takes too much time, you can confirm the high usage with an operating system command and then modify the NS_CACHE configuration parameter settings.

```
Individual virtual processors:
vp    pid       class       usercpu   syscpu    total     Thread    Eff
1     2036      cpu         76.95     7.14      84.09     99.08     84%
2     2149      adm         0.00      0.00      0.00      0.00       0%
3     2151      LIC         0.00      0.00      0.00      0.00       0%
4     2260      lio         0.00      0.00      0.00      0.03       0%
5     2442      pio         0.00      0.00      0.00      0.00       0%
6     2443      aio         0.00      0.01      0.01      0.11       8%
7     2444      msc         14.18     14.64     28.82     199.91    14%
8     2446      fifo        0.00      0.00      0.00      0.00       0%
```

You might also want to specify NS_CACHE information, if your operating system does not have a name service (NS) cache or if you disabled the operating system NS cache.

Example

To define the maximum retention time for your host and service connections as 600 seconds, and to disable the maximum retention limit for your user and group database server connections, specify:

```
NS_CACHE host=600,service=600,user=0,group=0
```


Connection information set in the NUMFDSERVERS configuration parameter

For network connections on UNIX, use the NUMFDSERVERS configuration parameter to specify the maximum number of poll threads to handle network connections migrating between Informix® virtual processors (VPs).

Specifying NUMFDSERVERS information is useful if Informix has a high rate of new connect and disconnect requests or if you find a high amount of contention between network shared file (NSF) locks.

Related information:
NUMFDSERVERS configuration parameter

Connection information set in the HA_ALIAS configuration parameter

The HA_ALIAS configuration parameter defines a network alias that is used for server-to-server communication in a high-availability cluster. The specified network alias is also used by Connection Managers, the **ifxclone** utility, and **onmode -d** commands.

Set the HA_ALIAS configuration parameter for each server in a high-availability cluster. The HA_ALIAS configuration parameter is required for high-availability cluster servers that use shared-memory connections.

The HA_ALIAS configuration parameter value must match a DBSERVERNAME or DBSERVERALIASES configuration parameter value that is associated with a TCP sqlhosts file entry.

Related information:
HA_ALIAS configuration parameter
onmode -d: Set data-replication types
DBSERVERALIASES configuration parameter
DBSERVERNAME configuration parameter

Environment variables for network connections

The INFORMIXCONTIME (connect time) and INFORMIXCONRETRY (connect retry) environment variables affect the behavior of the client when it is trying to connect to a database server. Use these environment variables to minimize connection errors caused by busy network traffic.

If the client application explicitly attaches to shared-memory segments, you might be required to set INFORMIXSHMBASE (shared-memory base).

You can use the INFORMIXSERVER environment variable to specify a default dbserver name to which your clients connect.

Related concepts:
How a client attaches to the communications portion (UNIX)
Related tasks:
Connections that the database server supports
Related information:
INFORMIXCONTIME environment variable
INFORMIXCONRETRY environment variable
INFORMIXSHMBASE environment variable (UNIX)
INFORMIXSERVER environment variable

Automatically terminating idle connections

You can automatically terminate sessions with clients that have been idle for a specified time by enabling the **idle_user_timeout** Scheduler task.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
To enable the **idle_user_timeout** task, run the following statement:

```
UPDATE ph_task
   SET tk_enable = 't'
   WHERE tk_name = 'idle_user_timeout';
```

By default, the **idle_user_timeout** task terminates user sessions that are idle for longer than 60 minutes. Sessions owned by user **informix** are not terminated. The **idle_user_timeout** task starts checking for idle sessions after two hours, which is the default frequency for the task.

Tip: When the system time changes on the database server computer, the amount of time user sessions have been idle is no longer accurate. For example, if a user session last did work at 3:14 PM and at 3:15 PM the system clock is moved forward by one hour, then to the database server, the user session has been idle for over an hour.
To change the idle timeout period, update the frequency of running the task and the value of the threshold. The shortest idle timeout period allowed is 5 minutes. For example, to change the timeout period to 5 minutes, run the following statements:

```
UPDATE ph_task
  SET tk_frequency = INTERVAL (5) MINUTE TO MINUTE
  WHERE tk_name = 'idle_user_timeout';

UPDATE ph_threshold
   SET value = '5'
   WHERE task_name = 'idle_user_timeout';
```


Distributed Relational Database Architecture (DRDA) communications

DRDA is a set of protocols that enables multiple database systems and application programs to work together.

This section contains information about how to configure IBM® Informix® to use the Distributed Relational Database Architecture™ (DRDA).

- Overview of DRDA
  Distributed Relational Database Architecture (DRDA) is a set of protocols that enable communication between applications and database systems on disparate platforms, and enables relational data to be distributed among multiple platforms.
- Configuring connectivity between Informix database servers and IBM Data Server clients
  To connect to with an IBM Data Server client, you must follow certain configuration steps.
- Allocating poll threads for an interface/protocol combination with the NETTYPE configuration parameter
  The NETTYPE configuration parameter configures poll threads for each connection type that your instance of the database server supports. You can use this configuration parameter to allocate more than one poll thread for an interface/protocol combination.
- Specify the size of the DRDA communication buffer with the DRDA_COMMBUFFSIZE configuration parameter
  Use the DRDA_COMMBUFFSIZE configuration parameter to specify the size of the DRDA communications buffer. The minimum size is 4 KB, the maximum size is 2 megabytes, and the default value is 32 KB.
- The DRDAEXEC thread and queries from clients
  For every DRDA client, IBM Informix creates a session and a DRDAEXEC thread, which is the equivalent of an SQLEXEC thread, to process and run the queries. This thread also formats the results of the queries in the DRDA protocol format and sends the results back to the client computer.
- SQL and supported and unsupported data types
  When using DRDA, IBM Informix syntax is supported over the common API. When using DRDA connections, rounds decimal and money values to 32-digit precision for all data retrieval operations on decimal or money data types.
- Display DRDA connection information
  Use **onstat** and **onmode** commands to display information that includes the DRDA thread name and an indicator that distinguishes SQLI and DRDA sessions
- Display DRDA session information
  Use the **syssesappinfo** table in the **sysmaster** database to view DRDA client session information.

Overview of DRDA

Distributed Relational Database Architecture™ (DRDA) is a set of protocols that enable communication between applications and database systems on disparate platforms, and enables relational data to be distributed among multiple platforms.

Any combination of relational database management products that use DRDA can be connected to form a distributed relational database management system. DRDA coordinates communication between systems by defining what is exchanged and the exchange method.

You can configure the database server to use DRDA to respond to requests from a common API, such as the IBM Data Server JDBC Driver or the IBM Data Server .NET Provider.

Connection Managers support DRDA, so you can use connection management to redirect client connection requests to appropriate database servers. Connection Managers can also provide automatic failover for high-availability clusters using DRDA.

Enterprise Replication, data replication, and utilities, such as DB-Access, require standard connections. Enterprise Replication utilities do not operate over DRDA connections. However, Enterprise Replication connections can coexist with DRDA connections.

When you use DRDA with ANSI-compliant databases, unbuffered logging and implicit transactions are enforced. If you migrate an application that is based on a non-ANSI-compliant database to a DRDA environment, the connection must handle application logic for statements that need transactions. For example, a BEGIN WORK statement is required before a concatenation operator in a stored procedure.

You can secure DRDA connections between a common client API and in the following ways:

- Encrypted password security or an encrypted user ID and encrypted password security
- Secure Sockets Layer (SSL) protocol to encrypt data in end-to-end
- Password authentication through a pluggable authentication module

Related concepts:
Connection management through the Connection Manager
Related information:
Transactions
BEGIN WORK statement
Secure sockets layer protocol
Configuring a connection to use PAM

Configuring connectivity between Informix® database servers and IBM Data Server clients

To connect to with an IBM Data Server client, you must follow certain configuration steps.

IBM Data Server Client and an applicable driver must be installed.

1. On each Connection Manager and database server host, add sqlhosts file entries for each server: For example:

```
#dbservername    nettype     hostname    servicename   options
 server_1        onsoctcp    host_1      port_1
 server_2        onsoctcp    host_2      port_2
 server_3        onsoctcp    host_3      port_3
```

2. In each database server's onconfig file, set the DBSERVERALIASES parameter to specify an alias for the server.
   The onconfig file entry for **server_1**:

   ```
   DBSERVERALIASES drda_1
   ```

   The onconfig file entry for **server_2**:

   ```
   DBSERVERALIASES drda_2
   ```

   The onconfig file entry for **server_3**:

   ```
   DBSERVERALIASES drda_3
   ```

3. On each Connection Manager's host, add sqlhosts file entries for DRDA aliases. Specify a **drtlitcp** or **drsoctcp** protocol and specify a port for DRDA communication. For example:

```
#dbservername  nettype    hostname   servicename   options
 server_1      onsoctcp   host_1     port_1
 server_2      onsoctcp   host_2     port_2
 server_3      onsoctcp   host_3     port_3

 drda_1        drsoctcp   host_1     drda_port_1
 drda_2        drsoctcp   host_2     drda_port_2
 drda_3        drsoctcp   host_3     drda_port_3
```

4. On the host of each Connection Manager, add a group entry for the group of database server and add a group entry for the group of DRDA aliases. Add group options to the database server and DRDA alias entries. Use the `c=1` group-entry option so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     group      -          -             c=1,e=server_3
 server_1       onsoctcp   host_1     port_1        g=my_servers
 server_2       onsoctcp   host_2     port_2        g=my_servers
 server_3       onsoctcp   host_3     port_3        g=my_servers

 drda_aliases   group      -          -             c=1,e=drda_3
 drda_1         drsoctcp   host_1     port_4        g=drda_aliases
 drda_2         drsoctcp   host_2     port_5        g=drda_aliases
 drda_3         drsoctcp   host_3     port_6        g=drda_aliases
```

5. Add the DRDA service-level agreements to your Connection Managers' configuration files. For example:
   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_1          DBSERVERS=PRI
   SLA sla_primary_drda_1     DBSERVERS=PRI
   SLA sla_secondaries_1      DBSERVERS=SDS,HDR
   SLA sla_secondaries_drda_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

   The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_2          DBSERVERS=PRI
   SLA sla_primary_drda_2     DBSERVERS=PRI
   SLA sla_secondaries_2      DBSERVERS=SDS,HDR
   SLA sla_secondaries_drda_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

6. On the host of each IBM Data Server client, create sqlhosts file entries for each service-level agreement (SLA) in each Connection Manager configuration file. Create group entries for each group of SLA entries, and add group options to the SLA entries.
   For example:

```
#dbservername            nettype    hostname   servicename   options
 g_primary               group      -          -             c=1,e=sla_primary_2
 sla_primary_1           onsoctcp   cm_host_1  cm_port_1     g=g_primary
 sla_primary_2           onsoctcp   cm_host_2  cm_port_2     g=g_primary

 g_secondaries           group      -          -             c=1,e=sla_secondaries_2
 sla_secondaries_2       onsoctcp   cm_host_1  cm_port_3     g=g_secondaries
 sla_secondaries_2       onsoctcp   cm_host_2  cm_port_4     g=g_secondaries

 g_primary_drda          group      -          -             c=1,e=sla_primary_2_drda
 sla_primary_1_drda      drsoctcp   cm_host_1  cm_port_5     g=g_primary_drda
 sla_primary_2_drda      drsoctcp   cm_host_2  cm_port_6     g=g_primary_drda

 g_secondaries_drda      group      -          -             c=1,e=sla_secondaries_2_drda
 sla_secondaries_2_drda  drsoctcp   cm_host_1  cm_port_7     g=g_secondaries_drda
 sla_secondaries_2_drda  drsoctcp   cm_host_2  cm_port_8     g=g_secondaries_drda
```

In the previous example, IBM Data Server client connection requests to **@g_primary_drda** are sent by **drsoctcp** protocol to one of the Connection Managers. The Connection Manager that receives the request uses an SLA to provide the client application with connection information for the primary server.

If you receive error -23104 when accessing the server through the DRDA protocol, the client application might be trying to bind a value that has an encoding different from the code page or code set of the database locale. Set the GL_USEGLU environment variable to `1` before you start the instance. This setting enables the server to initialize the appropriate Unicode converters that are required to handle the code set conversions.

Related concepts:
The sqlhosts information
Related tasks:
Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA)
Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA) and secure ports
Related information:
GL_USEGLU environment variable

Allocating poll threads for an interface/protocol combination with the NETTYPE configuration parameter

The NETTYPE configuration parameter configures poll threads for each connection type that your instance of the database server supports. You can use this configuration parameter to allocate more than one poll thread for an interface/protocol combination.

Set the NETTYPE configuration parameter as follows:

1. Specify **SQLI**, **drtlitcp**, or **drsoctcp** as the connection protocol.
2. Add information about the number of poll threads, the number of connections, and the virtual processor class.

For example, specify:

```
NETTYPE drtlitcp,3,2,CPU
```

A NETTYPE entry can handle multiple database server aliases on the same protocol type. Thus, when DRDA is in use, the network listener thread (NETTYPE **drtlitcp** or **drsoctcp**) typically has at least two sockets open and listening for connections. One socket is open for SQLI connections and another is open for DRDA connections.

Additional sockets might be open if many separate server aliases are configured.

Related information:
NETTYPE configuration parameter

Specify the size of the DRDA communication buffer with the DRDA_COMMBUFFSIZE configuration parameter

Use the DRDA_COMMBUFFSIZE configuration parameter to specify the size of the DRDA communications buffer. The minimum size is 4 KB, the maximum size is 2 megabytes, and the default value is 32 KB.

You can specify a one megabyte buffer as `1M`, `1m`, `1024K`, `1024k`, or `1024`. IBM® Informix® automatically resets values that are less than 4 KB as 32 KB.

When a DRDA session is established, the session allocates a communication buffer of the current buffer size.

You can use the **isgetdrdacommbuffsize()** function to return the current value of DRDA_COMMBUFFSIZE.

You cannot use the **onmode -wm** command to change the setting while the database server is running.

The DRDAEXEC thread and queries from clients

For every DRDA client, IBM® Informix® creates a session and a DRDAEXEC thread, which is the equivalent of an SQLEXEC thread, to process and run the queries. This thread also formats the results of the queries in the DRDA protocol format and sends the results back to the client computer.

Queries issued from a DRDA client run in parallel if PDQPRIORITY is set and the query can run in parallel. Queries run from DRDAEXEC threads can also run in parallel.

SQL and supported and unsupported data types

When using DRDA, IBM® Informix® syntax is supported over the common API. When using DRDA connections, rounds decimal and money values to 32-digit precision for all data retrieval operations on decimal or money data types.

The following data types are supported over the common API:

- BIGINT
- BIGSERIAL
- BLOB
- BOOLEAN
- BSON
- BYTE
- CHAR(32k)
- CLOB
- DATE
- DATETIME
- DECIMAL
- FLOAT
- INT
- INT8
- INTERVAL
- JSON
- LVARCHAR(32k)
- MONEY
- NCHAR(32k)
- NVARCHAR(255)
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- TEXT
- VARCHAR(255)

DATETIME values are mapped to DATE, TIME, or TIMESTAMP values.

The following data types are supported for use with database server host variables:

- CHAR
- DATE
- INT

- SMALLINT
- VCHAR

Display DRDA connection information

Use **onstat** and **onmode** commands to display information that includes the DRDA thread name and an indicator that distinguishes SQLI and DRDA sessions

The following commands display information about the thread name and session:

- **onstat -g ses**
- **onstat -g sql**
- **onstat -g ath**
- **onstat -g stk**
- **onstat -u**
- **onstat -x**
- **onstat -G**
- **onstat -g ddr**
- **onstat -g env**
- **onstat -g stm**
- **onstat -g ssc**
- **onmode -D**
- **onmode -Z**

For example, the **onstat** output might show "drdaexec" as the threadname.

Display DRDA session information

Use the **syssesappinfo** table in the **sysmaster** database to view DRDA client session information.

The table shows the client session ID, session application name, and a session value in the **sesapp_sid**, **sesapp_name**, and **sesapp_value** columns.

For example, the table might show the following information:

- **sesapp_sid**: `6`
- **sesapp_name**: `Accting`
- **sesapp_value**: `db2jcc_application`

You can also display client session information using the **onstat -g ses** command.

Related information:
The sysmaster Database
syssesappinfo
onstat -g ses command: Print session-related information

Examples of client/server configurations

The next several sections show the correct sqlhosts entries for several client/server connections.

You can assume that the network-configuration files hosts and services have been correctly prepared even if they are not explicitly mentioned. The following examples are included:

- Using a network connection
- Using multiple connection types
- Accessing multiple database servers

Examples of shared-memory and local-loopback connections can be found with the explanation of shared memory and local-loopback connections.

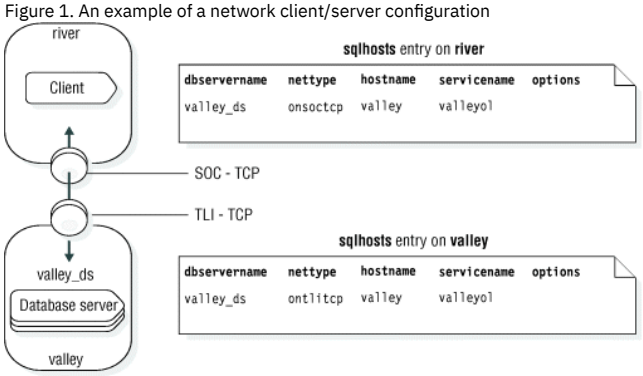
- A network connection
  The following figure shows a network-connection configuration for two database servers.
- Multiple connection types
  A single instance of the database server can provide more than one type of connection.
- Accessing multiple database servers
  When more than one database server is active on a single computer, it is known as *multiple residency*.

Related reference:
Shared-memory connections (UNIX)
Local-loopback connections

A network connection

The following figure shows a network-connection configuration for two database servers.

The client application is on host **river** and the database server is on host **valley**.
Figure 1. An example of a network client/server configuration



An sqlhosts entry for the **valley_ds** database server is defined on both computers.

Both computers are on the same TCP/IP network, but the host **river** uses sockets for its network programming interface, while the host **valley** uses TLI for its network programming interface. The **nettype** field must reflect the type of network programming interface used by the computer on which sqlhosts is located. In this example, the **nettype** field for the **valley_ds** database server on host **river** is `onsoctcp`, and the **nettype** field for the **valley_ds** database server on host **valley** is `ontlitcp`.

Related concepts:
Network programming interface

Multiple connection types

A single instance of the database server can provide more than one type of connection.

The following figure illustrates a configuration with more than one type of connection. The database server is on host **river**. Client A connects to the database server with a shared-memory connection because shared memory is fast. Client B must use a network connection because the client and server are on different computers.

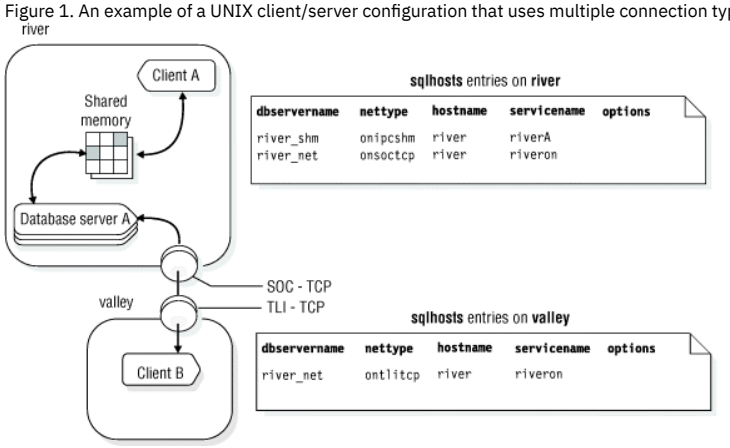
When you want the database server to accept more than one type of connection, you must take the following actions:

- Add DBSERVERNAME and DBSERVERALIASES entries in the onconfig file.
- Add an sqlhosts entry for each database server/connection type pair.

For the configuration in the following figure, the database server has two dbserver names: **river_net** and **river_shm**. The onconfig file includes the following entries:

```
DBSERVERNAME            river_net
DBSERVERALIASES         river_shm
```

Figure 1. An example of a UNIX client/server configuration that uses multiple connection types



The dbserver name used by a client application determines the type of connection that is used. Client A uses the following statement to connect to the database server:

```
CONNECT TO '@river_shm'
```

In the sqlhosts file, the **nettype** associated with the name **river_shm** specifies a shared-memory connection, so this connection is a shared-memory connection.

Client B uses the following statement to connect to the database server:

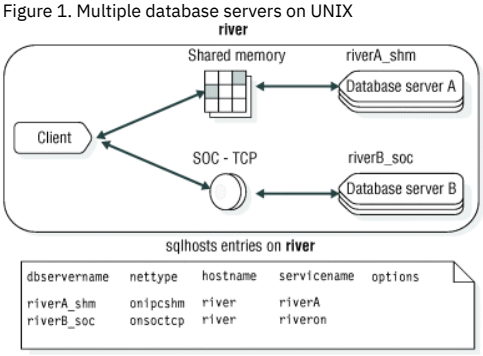
```
CONNECT TO '@river_net'
```

In the sqlhosts file, the **nettype** value associated with **river_net** specifies a network (TCP/IP) connection, so Client B uses a network connection.

Accessing multiple database servers

When more than one database server is active on a single computer, it is known as *multiple residency*.

The following figure shows a configuration with two database servers on host **river**.
Figure 1. Multiple database servers on UNIX



For the configuration in previous example, you must prepare an onconfig file for database server **A** and another one for database server **B**. The sqlhosts file includes the connectivity information for both database servers.

The onconfig file for database server **A** includes the following line:

```
DBSERVERNAME            riverA_shm
```

The onconfig file for database server **B** includes the following line:

```
DBSERVERNAME riverB_soc
```

Related information:
Creating multiresident database servers (Windows)
Creating multiresident database servers (UNIX, Linux, Mac OS X)

IBM Informix MaxConnect

IBM® Informix® MaxConnect is a networking product for IBM Informix database server environments on UNIX. Informix MaxConnect manages large numbers (from several hundred to tens of thousands) of client/server connections.

Informix MaxConnect multiplexes connections so that the ratio of client connections to database connections can be 200:1 or higher. Informix MaxConnect increases system scalability to many thousands of connections and saves system resources, reducing response times and processor requirements. Informix MaxConnect is best for OLTP data transfers and should not be used for large multimedia data transfers.

Install Informix MaxConnect separately from your IBM Informix database server and client applications. For maximum performance benefit, install Informix MaxConnect either on a separate computer to which IBM Informix clients connect or on the client application server. You can install Informix MaxConnect in the following configurations:

- On a dedicated server to which IBM Informix clients connect
- On the client application server
- On the database server computer

Two protocols for multiplexing connections, **ontliimc** and **onsocimc**, are available for Informix MaxConnect users. You can use the **ontliimc** and **onsocimc** protocols in the following two configurations:

- To connect Informix MaxConnect to the database server.
  In this configuration, the client connections are multiplexed and use packet aggregation.
- To connect the client applications directly to the database server without going through Informix MaxConnect.
  In this configuration, the client does not get the benefits of connection multiplexing or packet aggregation. Choose this configuration when the client application is transferring simple- or smart-large-object data, because a direct connection to the database server is best.

For more information about how to configure Informix MaxConnect and monitor it with the **onstat -g imc** and **imcadmin** commands, see the *IBM Informix MaxConnect User's Guide*.
Important: Informix MaxConnect and the *IBM Informix MaxConnect User's Guide* ship separately from the IBM Informix database server.
Related reference:
sqlhosts connectivity information

Database server initialization

The database server requires both disk-space initialization and shared-memory initialization.

Shared-memory initialization
: *Shared-memory initialization*, or starting the server, establishes the contents of database server shared memory as follows: internal tables, buffers, and the shared-memory communication area. Shared memory is initialized every time the database server starts. You use the **oninit** utility from the command line to initialize database server shared memory and bring the database server online.

Disk-space initialization
: *Disk-space initialization* uses the values that are stored in the configuration file to create the initial chunk of the root dbspace on disk. You use the **oninit -i** command from the command line to initialize disk space. When you initialize disk space, the database server automatically initializes shared memory as part of the process. Disk space is initialized the first time the database server starts.
Warning: When you initialize disk space, you overwrite whatever is on that disk space. If you reinitialize disk space for an existing database server, all the data in the earlier database server is deleted.
You can prevent accidental disk initialization by setting the FULL_DISK_INIT configuration parameter to 0. When this configuration parameter is set to 0, the **oninit -i** command fails if the root dbspace exists.

The key difference between shared-memory initialization and disk-space initialization is that shared-memory initialization has no effect on disk-space allocation or layout. No data is deleted.

- Initialization process
  When you start the database server or initialize disk space, the database server performs a set of steps. You can see the results of each step in the message log.
- Database server operating modes
  You can determine the current database server mode by running the **onstat** utility from the command line. The **onstat** header displays the mode.

Related concepts:
Database server configuration
Related information:
The oninit utility
onmode -k, -m, -s, -u, -j: Change database server mode
FULL_DISK_INIT configuration parameter

Initialization process

When you start the database server or initialize disk space, the database server performs a set of steps. You can see the results of each step in the message log.

Disk-space initialization always includes the initialization of shared memory. However, some activities that normally take place during shared-memory initialization, such as recording configuration changes, are not required during disk initialization because those activities are not relevant with a newly initialized disk.

The following main tasks are completed during the two types of initialization:

- Process the configuration file
- Create shared-memory segments
- Initialize shared-memory structures
- Initialize disk space, if necessary
- Start all required virtual processors
- Make necessary conversions
- Start fast recovery and checkpoint, if necessary
- Document configuration changes
- Update the oncfg_servername.servernum file
- Change to quiescent mode
- Drop temporary tblspaces (optional)
- Set forced residency, if requested
- Change to online mode and return control to user
- Create or update SMI tables as necessary
- Monitor maximum number of user connections at each checkpoint

Process the configuration file

The database server uses configuration parameters to allocate shared-memory segments during initialization and restart. If you modify a shared-memory configuration parameter, you must shut down and restart the database server for the change to take effect.

The ONCONFIG environment variable, which specifies the onconfig file that contains your configuration parameters, must be set before you initialize or restart the database server.

During initialization, the database server looks for configuration values in the following files:

- If the ONCONFIG environment variable is set, the database server reads values from the onconfig file.
  If the ONCONFIG environment variable is set, but the database server cannot access the specified onconfig file, the server returns an error message.
- If the ONCONFIG environment variable is not set, the database server reads the values from the onconfig file.

If you omit any configuration parameters in your onconfig file, the database server uses the default values that are built in the server.

The restart process compares the values in the current configuration file with the previous values, if any, that are stored in the root dbspace reserved page, PAGE_CONFIG. When differences exist, the database server uses the values from the current onconfig configuration file when the database server is restarted.

Create shared-memory segments

The database server uses the configuration values to calculate the required size of the database server resident shared memory. In addition, the database server computes additional configuration requirements from internal values. Space requirements are calculated and stored.

To create shared memory, the database server acquires the shared-memory space from the operating system for three different types of memory:

- Resident portion, which is used for data buffers and internal tables
- Virtual portion, used for most system and user-session memory requirements
- IPC communication portion, which is used for IPC communication
  The database server allocates this portion of shared memory only if you configure an IPC shared-memory connection.

Next, the database server attaches shared-memory segments to its virtual address space and initializes shared-memory structures. For more information about shared-memory structures, see Virtual portion of shared memory.

After initialization is complete and the database server is running, it can create additional shared-memory segments as necessary. The database server creates segments in increments of the page size.

Initialize shared-memory structures

After the database server attaches to shared memory, it clears the shared-memory space of uninitialized data. Next the database server lays out the shared-memory header information and initializes data in the shared-memory structures. The database server lays out the space that is required for the logical-log buffer, initializes the structures, and links together the three individual buffers that form the logical-log buffer.

After the database server remaps the shared-memory space, it registers the new starting addresses and sizes of each structure in the new shared-memory header.

During shared-memory initialization, disk structures and disk layout are not affected. The database server reads essential address information, such as the locations of the logical and physical logs, from disk and uses this information to update pointers in shared memory.

Initialize disk space, if necessary

Disk space is initialized only when you start the server for the first time or when you run the **oninit -i** command. Disk space is not initialized when the database server is restarted. After shared-memory structures are initialized, the database server begins initializing the disk. The database server initializes all the reserved pages that it maintains in the root dbspace on disk and writes page zero control information to the disk.

If the DISK_ENCRYPTION configuration parameter is set, the root dbspace is encrypted.

The FULL_DISK_INIT configuration parameter specifies whether **oninit -i** can run on your instance when a page zero exists at the root path location (at the first page of the first chunk location). Use this configuration parameter to prevent an accidental disk reinitialization of an existing server instance.

The default setting of the FULL_DISK_INIT configuration parameter is `0`. When the configuration parameter is set to`0`, the **oninit -i** command runs only if there is not a page zero at the root path location.

If a page zero exists at the root path location, initialization occurs only if the FULL_DISK_INIT configuration parameter is set to `1`. The database server automatically resets the FULL_DISK_INIT configuration parameter to`0` after the initialization.

Start all required virtual processors

The database server starts all the virtual processors that it requires.

The parameters in the onconfig file influence what processors are started. For example, the NETTYPE parameter can influence the number and type of processors that are started for making connections. For more information about virtual processors, see Virtual processors.

Make necessary conversions

The database server checks its internal files. If the files are from an earlier version, it updates these files to the current format.

Start fast recovery and checkpoint, if necessary

The database server checks if fast recovery is required and, if so, starts it. Fast recovery is not performed during disk-space initialization because there is not yet anything to recover.

For more information about fast recovery, see Fast recovery.

After fast recovery completes, the database server runs a checkpoint to verify that all recovered transactions are flushed to disk so the transactions are not repeated.

As part of the checkpoint procedure, the database server writes a checkpoint-complete message in the message log. For more information about checkpoints, see Checkpoints.

Document configuration changes

The database server compares the current values that are stored in the configuration file with the values previously stored in the root dbspace reserved page PAGE_CONFIG. When differences exist, the database server notes both values (old and new) in a message to the message log.

Update the oncfg_servername.servernum file

The database server creates the oncfg_*servername.servernum* file and updates it every time that you add or delete a dbspace, blobspace, logical-log file, or chunk.

You are not required to manipulate this file in any way, but you can see it listed in your $INFORMIXDIR/etc directory on UNIX or in your %INFORMIXDIR%\etc directory on Windows. The database server uses the oncfg_*servername.servernum* file during a full-system restore for salvaging the logical log.

Change to quiescent mode

The database server now moves to quiescent mode or online mode, depending on how you started the initialization or database-server restart process.

Drop temporary tblspaces (optional)

Temporary tblspaces, if any, are tblspaces left by user processes that died prematurely and were unable to perform appropriate cleanup. The database server deletes any temporary tblspaces and reclaims the disk space. For more information about temporary tblspaces, see Temporary tables.

This task is performed when the database server is restarted; it is not performed during disk-space initialization.

If you use the **-p** option of **oninit** to initialize the database server, the database server skips this step.

Set forced residency, if requested

If the value of the RESIDENT configuration parameter is `-1` or a number greater than 0, the database server tries to enforce residency of shared memory.

If the host computer system does not support forced residency, the initialization procedure continues. Residency is not enforced, and the database server sends an error message to the message log.

Change to online mode and return control to user

Control returns to the user when the database server writes the `IBM® Informix® Dynamic Server initialized - complete disk initialized` message in the message log only if initialization, not database-server restart, occurred and dynamically allocates a virtual shared-memory segment.

The server returns control to the user. Any error messages that are generated by the initialization procedure are displayed in the following locations:

- The command line
- The database server message log file, which is specified by the MSGPATH configuration parameter

You can use the **oninit -w** utility to force the server to return to a command prompt within a configurable timeout. The **oninit -w** utility is useful for troubleshooting initialization failures.

Create or update SMI tables as necessary

The database server creates the system-monitoring interface (SMI) tables and other system databases if they do not exist.

If the SMI tables are not current, the database server updates the tables. If the SMI tables are not present, as is the case when the disk is initialized, the database server creates the tables. After the database server builds the SMI tables, it puts the message `sysmaster database built successfully` in the message log. The database server also re-creates the **sysmaster** database during conversion and reversion. For more information about SMI tables, see the chapter on the **sysmaster** database in the *IBM Informix Administrator's Reference*.

If you shut down the database server before it finishes building the SMI tables, the process of building the tables stops. This condition does not damage the database server. The database server builds the SMI tables the next time that you bring the database server online. However, if you do not allow the SMI tables to finish building, you cannot run any queries against those tables, and you cannot use ON-Bar for backups.

The database server drops and re-creates the **sysutils** database during disk initialization, conversion, or reversion. ON-Bar stores backup and restore information in the **sysutils** database. Wait until the message `sysutils database built successfully` displays in the message log.

The database server creates the **sysuser** database during initialization. The **sysuser** database is used for Pluggable Authentication Module (PAM) authentication in IBM Informix server to server communication.

The database server creates the **sysadmin** database during initialization. The **sysadmin** database provides remote administration and scheduler API features in IBM Informix.

After the SMI tables and system databases are created, the database server is ready for use. The database server runs until you stop it or, possibly, until a malfunction.

Recommendation: Do not try to stop the database server by stopping a virtual processor or ending another database server process. For more information, see Start and stop virtual processors.

Monitor maximum number of user connections at each checkpoint

The database server prints the maximum number of user connections in the message log at each checkpoint in the following format: `maximum server connections` *`number`*. You can monitor the number of users who connected to the database server since the last restart or disk initialization.

The number that is displayed is reset when the customer reinitializes the database server.

Related information:
DISK_ENCRYPTION configuration parameter

Database server operating modes

You can determine the current database server mode by running the **onstat** utility from the command line. The **onstat** header displays the mode.

The table shows the principal modes of operation of the database server.

Table 1. Operating modes

Operating mode	Description	Users allowed access
Offline mode	The database server is not running. Shared memory is not allocated.	Only the administrator (user **informix**) can change from this mode to another mode.
Quiescent mode	Database-server processes are running and shared-memory resources are allocated. Administrators use this mode to perform maintenance functions that do not require the execution of SQL and DDL statements.	Only the administrator (user **informix**) can access the database server. Other users can view database-server status information, but they cannot access the database server.
Administration mode	This mode is an intermediary mode between Quiescent mode and Online mode. Administrators use this mode to perform any maintenance task, including tasks requiring the execution of SQL and DDL statements. Administrators can also perform all other functions available in Online mode.	The following users can connect to the database server in administration mode: • User **informix** • Users who have the DBSA role Set the ADMIN_USER_MODE_WITH_DBSA configuration parameter to `1` if you want users who are members of the DBSA group (in addition to user **informix**) to connect to the database server in administration mode. • One or more users who have administration mode access User **informix** or a DBSA can dynamically give one or more specific users the ability to connect to the database server in administration mode through the **onmode -j** command, the **oninit -U** command, or the ADMIN_MODE_USERS configuration parameter. Other users can view database-server status information, but they cannot access the database server.
Online mode	This is the normal operating mode of the database server.	Any authorized user can connect with the database server and perform all database activities. User **informix** or user **root** can use the command-line utilities to change many database server ONCONFIG parameter values.

In addition, the database server can also be in one of the following modes:

- *Read-only mode* is used by the secondary database server in a data replication environment. An application can query a secondary database server that is in read-only mode, but the application cannot write to a read-only database.
- *Recovery mode* is transitory. It occurs when the database server performs fast recovery or recovers from a system archive or system restore. Recovery occurs during the change from offline to quiescent mode.
- *Shutdown mode* is transitory. It occurs when the database server is moving from online to quiescent mode or from online (or quiescent) to offline mode. For the current users access the system, but no new users are allowed access.
  After shutdown mode is initiated, it cannot be canceled.

- Users permitted to change modes
- Changing database server operating modes
  Use the **oninit** and **onmode** utilities to change from one database server operating mode to another. Use the ADMIN_MODE_USERS configuration parameter to specify which users can connect to the server in administration mode.

Users permitted to change modes

UNIX only

Users who are logged in as **root** or **informix** and members of the DBSA group can change the operating mode of the database server.

If you want users with the DBSA group to connect to the database server in administration mode, set the ADMIN_USER_MODE_WITH_DBSA configuration parameter to `1`. If this parameter is set to zero, then access is restricted to user **informix** only. If the parameter is missing from $ONCONFIG, it is treated as `0`.

User **informix** or a DBSA can dynamically give one or more specific users the ability to connect to the database server in administration mode, using the **onmode** utility, the **oninit** utility, or the ADMIN_MODE_USERS configuration parameter.
Note: For a member of the DBSA group, the permissions on $INFORMIXDIR/bin/oninit must be changed to allow **public execute permission - root:informix:6755** in a standard IBM® Informix® installation.

Windows only

Table 1 shows which users can change the operating mode of the database server in Windows. Apache as user **informix**. The Apache server runs as a member of the **Informix-Admin** group.

Table 1. Changing operating modes in windows

Changing operating mode	Administrators group	Informix-Admin group
command-line utilities such as **starts**		X
services control panel	X	

Changing database server operating modes

Use the **oninit** and **onmode** utilities to change from one database server operating mode to another. Use the ADMIN_MODE_USERS configuration parameter to specify which users can connect to the server in administration mode.

Windows only: In Windows, the database server runs as a service. Windows provides a service control application (also called the Services tool) to start, stop, and pause a service. The service control application is located in the Control Panel program group. The service name for the database server includes the database server name (the value of DBSERVERNAME in the onconfig file). For example, the IBM® Informix® service for the `newyork` database server is:

```
IBM Informix Database Server - newyork
```

To change mode with the Services tool, start the tool and select the database server service. Then choose the appropriate option in the Services window. The tables shown later in these topics explain which option you select for each mode.

To start and stop the database server, you can use other Windows tools, such as the NET command and the Server Manager tool. For more information about these methods, consult your Windows operating-system documentation.

Tip: After you change the mode of your database server, run the **onstat** command to verify the current server status.
You can change database server modes in the following ways:

Change from offline to quiescent mode
: When the database server changes from offline mode to quiescent mode, the database server initializes shared memory. Only administrators can access the database server to perform maintenance functions that do not involve the execution of SQL and DDL statements.
UNIX: Run the **oninit -s** command.
Windows: On the command line, use the **starts** *dbservername* **-s** command.

Change from offline to online mode
: When you move the database server from offline to online mode, the database server initializes shared memory and is available for all user sessions.
UNIX: Run the **oninit** command.
Windows: In the Services tool, select the database server service and click Start. Alternatively, on the command line, use the **starts *dbservername*** command.

Change from offline to administration mode
: When you move the database server from offline to administration mode, you move the server into a mode that only administrators can use to perform database server functions and maintenance functions, including those involving the execution of SQL and DDL statements.
Run the **oninit -j** command.

Change from quiescent to online mode
: When you take the database server from quiescent mode to online mode, all sessions gain access. If you have already taken the database server from online mode to quiescent mode and you are now returning the database server to online mode, any users who were interrupted in earlier processing must reselect their database and redeclare their cursors.
Run the **onmode -m** command.
Windows: In the Services tool, choose the database server service and click Continue.

Change gracefully from online to quiescent mode
: Take the database server gracefully from online mode to quiescent mode to restrict access to the database server without interrupting current processing. After you perform this task, the database server sets a flag that prevents new sessions from gaining access to the database server. The current sessions are allowed to finish processing. After you initiate the mode change, it cannot be canceled. During the mode change from online to quiescent, the database server is considered to be in Shutdown mode.
Run the **onmode -s** or the **onmode -sy** command.
Windows: In the Services tool, choose the database server service and click Pause.

Change immediately from online to quiescent mode
: Take the database server immediately from online mode to quiescent mode to restrict access to the database server as soon as possible. Work in progress can be lost. A prompt asks for confirmation of the immediate shutdown. If you confirm, the database server sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates the session. The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated. The database server cleans up all sessions that the database server terminated. Active transactions are rolled back.
Run the **onmode -u** or the **onmode -uy** command.

Change from quiescent or online to administration mode
: When you move the database server from quiescent or online to administration mode, you move the server into a mode that only administrators can use. If you begin in online mode, the database server automatically disconnects any users who are connected with any user ID that is not user **informix** and the users receive an error message. If a connection is terminated during a transaction, the database server rolls back the transaction. Change to administration mode when you want to run SQL and DLL commands when no other users are connected. Also see Specifying administration mode users.
Run the **onmode -j** command.

Change from administration to online mode
: When you move the database server from administration to online mode, all users can access the database server.
Run the **onmode -m** command.

Change from administration to quiescent mode
: When you move the database server from administration to quiescent mode, you move the server into a mode that only administrators can use to perform maintenance functions that do not involve the execution of SQL and DDL statements.
Run the **onmode -s** command.

Change from any mode immediately to offline mode
: You can take the database server immediately from any mode to offline mode. A prompt asks for confirmation to go offline. If you confirm, the database server initiates a checkpoint request and sends a disconnect signal to all sessions that are attached to shared memory. If a session does not receive the disconnect signal or is not able to comply automatically within 10 seconds, the database server terminates this session. The database server users receive either error message -459 indicating that the database server was shut down or error message -457 indicating that their session was unexpectedly terminated. After you take the database server to offline mode, restart the database server in quiescent, administration, or online mode. When you restart the database server, it performs a fast recovery to ensure that the data is logically consistent. The database server cleans up all sessions that were terminated by the database server. Active transactions are rolled back.
Run the **onmode -k** or the **onmode -ky** command.
Windows: In the Services tool, choose the database server service and click Stop.

If the **onmode** command fails to shut down the database server, you can use the **onclean** utility to force an immediate shutdown.

- Specifying administration mode users
  You can specify which users can connect to the database server in administration mode.

Related information:
The oninit utility
onmode -k, -m, -s, -u, -j: Change database server mode

Specifying administration mode users

You can specify which users can connect to the database server in administration mode.

Temporary administration mode users

User **informix** or a DBSA can use the **onmode -j -U** or the **oninit -U** command to grant individual users access to the database server in administration mode for a session.

For example, run the following command to enable three individual users to connect to the database server and have database server access until the database server mode changes to offline, quiescent or online mode:

```
onmode -j -U mark,ajay,carol
```

After connecting, these individual users can run any SQL or DDL commands. When the server is changed to administration mode, all sessions for users not identified in the **onmode -j -U** command lose their database server connection.

After initially running the **onmode -j -U** command, you can remove individuals by running **onmode -j -U** and removing individual user names from the new list of names in the command, for example, by running:

```
onmode -j -U mark,carol
```

Run the **oninit -U** command with a blank space instead of a name to remove all users in the list, as shown in this example:

```
oninit -U " "
```

Permanent administration mode users

Unlike the **oninit** and **onmode** commands that enable you to specify administration mode users until the server changes to offline, quiescent, or online mode, the ADMIN_MODE_USERS configuration parameter preserves a list of administration mode users indefinitely.

To create a list of administration mode users that is preserved in the onconfig file, specify a comma-separated list of users as ADMIN_MODE_USERS configuration parameter values, for example, `mark,ajay,carol`.

To override ADMIN_MODE_USERS during a session, use the **onmode -wf** command, as shown in this example:

```
onmode -wf ADMIN_MODE_USERS=sharon,kalpana
```

The effect of the ADMIN_MODE_USERS configuration parameter is to add to the list of people permitted to access the server in administration mode. Those people listed in the **onmode** command line override those listed in the onconfig file.
Related information:
The oninit utility
onmode -k, -m, -s, -u, -j: Change database server mode
ADMIN_MODE_USERS configuration parameter

Disk, memory, and process management

- Virtual processors and threads
  These topics describe virtual processors, explain how threads run within the virtual processors, and explain how the database server uses virtual processors and threads to improve performance.
- Manage virtual processors
  These topics describe how to set the configuration parameters that affect database server virtual processors, and how to start and stop virtual processors.
- Shared memory
  These topics describe the content of database server shared memory, the factors that determine the sizes of shared-memory areas, and how data moves into and out of shared memory.
- Manage shared memory
- Data storage
  The database server uses physical units of storage to allocate disk space. It stores data in logical units. Unlike the logical units of storage whose size fluctuates, each of the physical units has a fixed or assigned size that is determined by the disk architecture.
- Manage disk space
  You can use several utilities and tools to manage disk spaces and the data that the database server controls.
- Moving data with external tables
  You can use external tables to load and unload database data.
- Storage space encryption
  You can encrypt storage spaces (dbspaces, blobspaces and smart blobspaces) with Informix Dynamic Server. The data in encrypted storage spaces is unintelligible

without the encryption key. Encrypting storage spaces is an effective way to protect sensitive information that is stored on disk.

Virtual processors and threads

These topics describe virtual processors, explain how threads run within the virtual processors, and explain how the database server uses virtual processors and threads to improve performance.

- Virtual processors
  A virtual processor is a process that the operating system schedules for processing.
- How virtual processors service threads
  A virtual processor services multiple threads concurrently by switching between them.
- Virtual processor classes
  Each class of virtual processor is dedicated to processing certain types of threads.

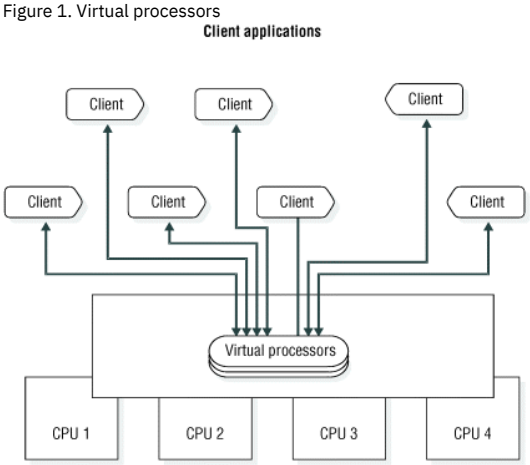
Related reference:
Database server maintenance tasks

Virtual processors

A virtual processor is a process that the operating system schedules for processing.

Database server processes are called *virtual processors* because the way they function is similar to the way that a CPU functions in a computer. Just as a CPU runs multiple operating-system processes to service multiple users, a database server virtual processor runs multiple threads to service multiple SQL client applications.

The following figure illustrates the relationship of client applications to virtual proc
Figure 1. Virtual processors



essors. A small number of virtual processors serve a much larger number of client applications or queries.

- Threads
  A thread is a task for a virtual processor in the same way that the virtual processor is a task for the CPU.
- Advantages of virtual processors
  A virtual processor provides a number of advantages.

Threads

A thread is a task for a virtual processor in the same way that the virtual processor is a task for the CPU.

The virtual processor is a task that the operating system schedules for execution on the CPU; a database server thread is a task that the virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes, but they make fewer demands on the operating system.

Database server virtual processors are multithreaded because they run multiple concurrent threads.

The nature of threads is as follows.

Operating system	Action
UNIX	A thread is a task that the virtual processor schedules internally for processing.

Operating system	Action
Windows	A thread is a task that the virtual processor schedules internally for processing. Because the virtual processor is implemented as a Windows thread, database server threads run within Windows threads.

Important: Throughout these topics, all references to *thread* refer to the threads created, scheduled, and deleted by the database server. All references to "Windows threads" refer to the threads created, scheduled, and deleted by Windows.
A virtual processor runs threads on behalf of SQL client applications (*session* threads) and also to satisfy internal requirements (*internal* threads). In most cases, for each connection by a client application, the database server runs one session thread. The database server runs internal threads to accomplish, among other things, database I/O, logging I/O, page cleaning, and administrative tasks. For cases in which the database server runs multiple session threads for a single client, see Parallel processing.

A *user thread* is a database server thread that services requests from client applications. User threads include session threads, called **sqlexec** threads, which are the primary threads that the database server runs to service client applications.

User threads also include a thread to service requests from the **onmode** utility, threads for recovery, B-tree scanner threads, and page-cleaner threads.

To display active user threads, use **onstat -u**. For more information about monitoring sessions and threads, see *IBM® Informix® Performance Guide*.

Advantages of virtual processors

A virtual processor provides a number of advantages.

Compared to a database server process that services a single client application, the dynamic, multithreaded nature of a database server virtual processor provides the following advantages:

- Virtual processors can share processing.
- Virtual processors save memory and resources.
- Virtual processors can perform parallel processing.
- You can start additional virtual processors and terminate active CPU virtual processors while the database server is running.
- You can bind virtual processors to CPUs.

The following topics describe these advantages.

- Shared processing
  Virtual processors in the same class have identical code and share access to both data and processing queues in memory. Any virtual processor in a class can run any thread that belongs to that class.
- Save memory and resources
  The database server is able to service many clients with a small number of server processes compared to architectures that have one client process to one server process by running a thread, rather than a process, for each client.
- Parallel processing
  When a client initiates index building, sorting, or logical recovery, the database server creates multiple threads to work on the task in parallel, using as much of the computer resources as possible. While one thread is waiting for I/O, another can be working.
- Add and drop virtual processors in online mode
  You can add virtual processors to meet increasing demands for service while the database server is running.
- Bind virtual processors to CPUs
  You can use some multiprocessor systems to bind a process to a particular CPU. This feature is called *processor affinity*.

Shared processing

Virtual processors in the same class have identical code and share access to both data and processing queues in memory. Any virtual processor in a class can run any thread that belongs to that class.

Generally, the database server tries to keep a thread running on the same virtual processor because moving it to a different virtual processor can require some data from the memory of the processor to be transferred on the bus. When a thread is waiting to run, however, the database server can migrate the thread to another virtual processor because the benefit of balancing the processing load outweighs the amount of overhead incurred in transferring the data.

Shared processing within a class of virtual processors occurs automatically and is transparent to the database user.

Save memory and resources

The database server is able to service many clients with a small number of server processes compared to architectures that have one client process to one server process by running a thread, rather than a process, for each client.

Multithreading permits more efficient use of the operating-system resources because threads share the resources allocated to the virtual processor. All threads that a virtual processor runs have the same access to the virtual-processor memory, communication ports, and files. The virtual processor coordinates access to resources by the threads. Individual processes, though, each have a distinct set of resources, and when multiple processes require access to the same resources, the operating system must coordinate the access.

Generally, a virtual processor can switch from one thread to another faster than the operating system can switch from one process to another. When the operating system switches between processes, it must stop one process from running on the processor, save its current processing state (or context), and start another process. Both processes must enter and exit the operating-system kernel, and the contents of portions of physical memory might require replacement. Threads, though, share the same virtual memory and file descriptors. When a virtual processor switches from one thread to another, the switch is from one path of execution to another. The virtual processor, which is a process, continues to run on the CPU without interruption. For a description of how a virtual processor switches from one thread to another, see Context switching.

Parallel processing

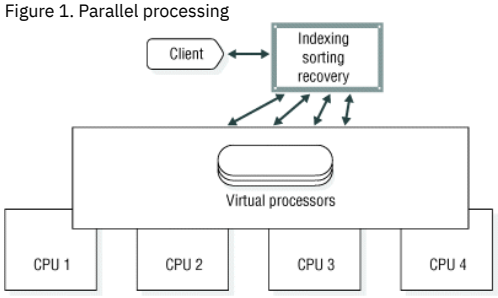
When a client initiates index building, sorting, or logical recovery, the database server creates multiple threads to work on the task in parallel, using as much of the computer resources as possible. While one thread is waiting for I/O, another can be working.

In the following cases, virtual processors of the CPU class can run multiple session threads, working in parallel, for a single client:

- Index building
- Sorting
- Recovery
- Scanning
- Joining
- Aggregation
- Grouping
- User-defined-routine (UDR) execution

For more information about parallel UDR execution, see *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

The following figure illustrates parallel processing.
Figure 1. Parallel processing



Add and drop virtual processors in online mode

You can add virtual processors to meet increasing demands for service while the database server is running.

If the virtual processors of a class become compute bound or I/O bound (meaning that CPU work or I/O requests are accumulating faster than the current number of virtual processors can process them), you can start additional virtual processors for that class to distribute the processing load further.

For more information, see Add virtual processors in online mode.

Windows only: In Windows, you cannot drop a virtual processor of any class.
While the database server is running, you can drop virtual processors of the CPU or a user-defined class. For more information, see Set virtual-processor configuration parameters.

Bind virtual processors to CPUs

You can use some multiprocessor systems to bind a process to a particular CPU. This feature is called *processor affinity*.

On multiprocessor computers for which the database server supports processor affinity, you can bind CPU virtual processors to specific CPUs in the computer. When you bind a CPU virtual processor to a CPU, the virtual processor runs exclusively on that CPU. This operation improves the performance of the virtual processor because it reduces the amount of switching between processes that the operating system must do. Binding CPU virtual processors to specific CPUs also enables you to isolate database work on specific processors on the computer, leaving the remaining processors free for other work. Only CPU virtual processors can be bound to CPUs.

For information about how to assign CPU virtual processors to hardware processors, see Processor affinity.

How virtual processors service threads

A virtual processor services multiple threads concurrently by switching between them.

At a given time, a virtual processor can run only one thread. A virtual processor runs a thread until it yields. When a thread yields, the virtual processor switches to the next thread that is ready to run. The virtual processor continues this process, eventually returning to the original thread when that thread is ready to continue. Some threads complete their work, and the virtual processor starts new threads to process new work. Because a virtual processor continually switches between threads, it can keep the CPU processing continually. The speed at which processing occurs produces the appearance that the virtual processor processes multiple tasks simultaneously and, in effect, it does.

Running multiple concurrent threads requires scheduling and synchronization to prevent one thread from interfering with the work of another. Virtual processors use the following structures and methods to coordinate concurrent processing by multiple threads:

- Control structures
- Context switching
- Stacks
- Queues
- Mutexes

These topics describe how virtual processors use these structures and methods.

- Control structures
  When a client connects to the database server, the database server creates a *session* structure, which is called a *session control block*, to hold information about the connection and the user.
- Context switching
  A virtual processor switches from running one thread to running another one by *context switching*.
- Stacks
  The database server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area is called the *stack*.
- Queues
  The database server uses three types of queues to schedule the processing of multiple, concurrently running threads.
- Mutexes
  A mutex (*mut*ually *ex*clusive), also called a *latch*, is a latching mechanism that the database server uses to synchronize access by multiple threads to shared resources.

Control structures

When a client connects to the database server, the database server creates a *session* structure, which is called a *session control block*, to hold information about the connection and the user.

A session begins when a client connects to the database server, and it ends when the connection terminates.

Next, the database server creates a thread structure, which is called a *thread-control block* (TCB) for the session, and initiates a primary thread (**sqlexec**) to process the client request. When a thread *yields*—that is, when it pauses and allows another thread to run—the virtual processor saves information about the state of the thread in the thread-control block. This information includes the content of the process system registers, the program counter (address of the next instruction to execute), and the stack pointer. This information constitutes the context of the thread.

In most cases, the database server runs one primary thread per session. In cases where it performs parallel processing, however, it creates multiple session threads for a single client, and, likewise, multiple corresponding thread-control blocks.

Context switching

A virtual processor switches from running one thread to running another one by *context switching*.

The database server does not preempt a running thread, as the operating system does to a process, when a fixed amount of time (time-slice) expires. Instead, a thread yields at one of the following points:

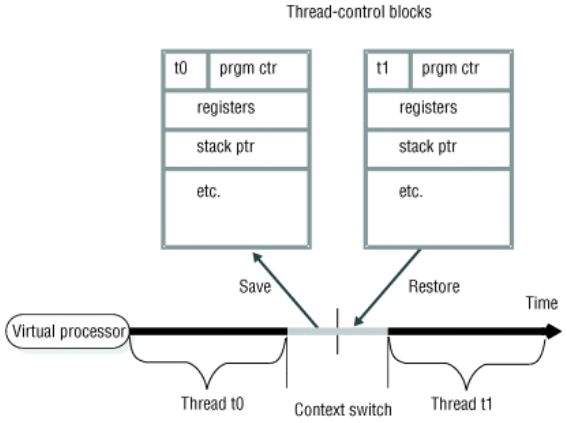
- A predetermined point in the code
- When the thread can no longer execute until some condition is met

When the amount of processing required to complete a task would cause other threads to wait for an undue length of time, a thread yields at a predetermined point. The code for such long-running tasks includes calls to the yield function at strategic points in the processing. When a thread performs one of these tasks, it yields when it encounters a yield function call. Other threads in the ready queue then get a chance to run. When the original thread next gets a turn, it resumes executing code at the point immediately after the call to the yield function. Predetermined calls to the yield function allow the database server to interrupt threads at points that are most advantageous for performance.

A thread also yields when it can no longer continue its task until some condition occurs. For example, a thread yields when it is waiting for disk I/O to complete, when it is waiting for data from the client, or when it is waiting for a lock or other resource.

When a thread yields, the virtual processor saves its context in the thread-control block. Then the virtual processor selects a new thread to run from a queue of ready threads, loads the context of the new thread from its thread-control block, and begins executing at the new address in the program counter. The following figure illustrates

how a virtual processor accomplishes a context switch.
Figure 1. Context switch: how a virtual processor switches from one thread to another



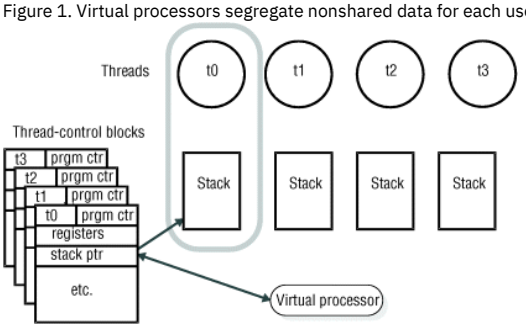
Stacks

The database server allocates an area in the virtual portion of shared memory to store nonshared data for the functions that a thread executes. This area is called the *stack*.

For information about how to set the size of the stack, see Stacks.

The stack enables a virtual processor to protect the nonshared data of a thread from being overwritten by other threads that concurrently execute the same code. For example, if several client applications concurrently perform SELECT statements, the session threads for each client execute many of the same functions in the code. If a thread did not have a private stack, one thread might overwrite local data that belongs to another thread within a function.

When a virtual processor switches to a new thread, it loads a stack pointer for that thread from a field in the thread-control block. The stack pointer stores the beginning address of the stack. The virtual processor can then specify offsets to the beginning address to access data within the stack. The figure illustrates how a virtual processor uses the stack to segregate nonshared data for session threads.
Figure 1. Virtual processors segregate nonshared data for each user



Queues

The database server uses three types of queues to schedule the processing of multiple, concurrently running threads.

Virtual processors of the same class share queues. This fact, in part, enables a thread to migrate from one virtual processor in a class to another when necessary.

- Ready queues
  Ready queues hold threads that are ready to run when the current (running) thread yields.
- Sleep queues
  Sleep queues hold the contexts of threads that have no work to do at a particular time. A thread is put to sleep either for a specified period of time or forever.
- Wait queues
  Wait queues hold threads that must wait for a particular event before they can continue to run.

Ready queues

Ready queues hold threads that are ready to run when the current (running) thread yields.

When a thread yields, the virtual processor picks the next thread with the appropriate priority from the ready queue. Within the queue, the virtual processor processes threads that have the same priority on a first-in-first-out (FIFO) basis.

On a multiprocessor computer, if you notice that threads are accumulating in the ready queue for a class of virtual processors (indicating that work is accumulating faster than the virtual processor can process it), you can start additional virtual processors of that class to distribute the processing load. For information about how to monitor the ready queues, see Monitor virtual processors. For information about how to add virtual processors while the database server is in online mode, see Add virtual processors in online mode.

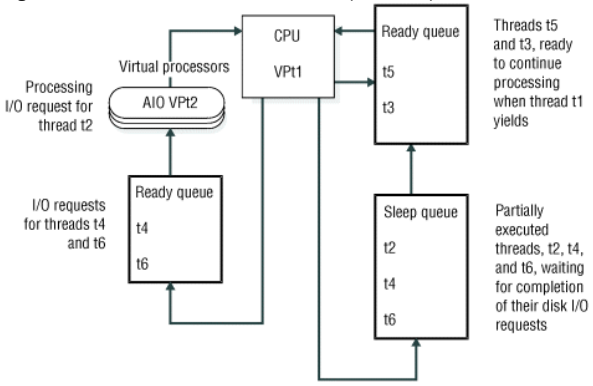
Sleep queues

Sleep queues hold the contexts of threads that have no work to do at a particular time. A thread is put to sleep either for a specified period of time or forever.

The administration class (ADM) of virtual processors runs the system timer and special utility threads. Virtual processors in this class are created and run automatically. No configuration parameters affect this class of virtual processors.

The ADM virtual processor wakes up threads that have slept for the specified time. A thread that runs in the ADM virtual processor checks on sleeping threads at one-second intervals. If a sleeping thread has slept for its specified time, the ADM virtual processor moves it into the appropriate ready queue. A thread that is sleeping for a specified time can also be explicitly awakened by another thread.

A thread that is sleeping forever is awakened when it has more work to do. For example, when a thread that is running on a CPU virtual processor must access a disk, it issues an I/O request, places itself in a sleep queue for the CPU virtual processor, and yields. When the I/O thread notifies the CPU virtual processor that the I/O is complete, the CPU virtual processor schedules the original thread to continue processing by moving it from the sleep queue to a ready queue. The following figure illustrates how the database server threads are queued to perform database I/O.
Figure 1. How database server threads are queued to perform database I/O



Wait queues

Wait queues hold threads that must wait for a particular event before they can continue to run.

Wait queues coordinate access to shared data by threads. When a user thread tries to acquire the logical-log latch but finds that the latch is held by another user, the thread that was denied access puts itself in the logical-log wait queue. When the thread that owns the lock is ready to release the latch, it checks for waiting threads, and, if threads are waiting, it wakes up the next thread in the wait queue.

Mutexes

A mutex (*mut*ually *ex*clusive), also called a *latch*, is a latching mechanism that the database server uses to synchronize access by multiple threads to shared resources.

Mutexes are similar to semaphores, which some operating systems use to regulate access to shared data by multiple processes. However, mutexes permit a greater degree of parallelism than semaphores.

A mutex is a variable that is associated with a shared resource such as a buffer. A thread must acquire the mutex for a resource before it can access the resource. Other threads are excluded from accessing the resource until the owner releases it. A thread acquires a mutex, after a mutex becomes available, by setting it to an in-use state. The synchronization that mutexes provide ensures that only one thread at a time writes to an area of shared memory.

For information about monitoring mutexes, see Monitor the shared-memory profile and latches.

Related concepts:
Buffer pool portion of shared memory

Virtual processor classes

Each class of virtual processor is dedicated to processing certain types of threads.

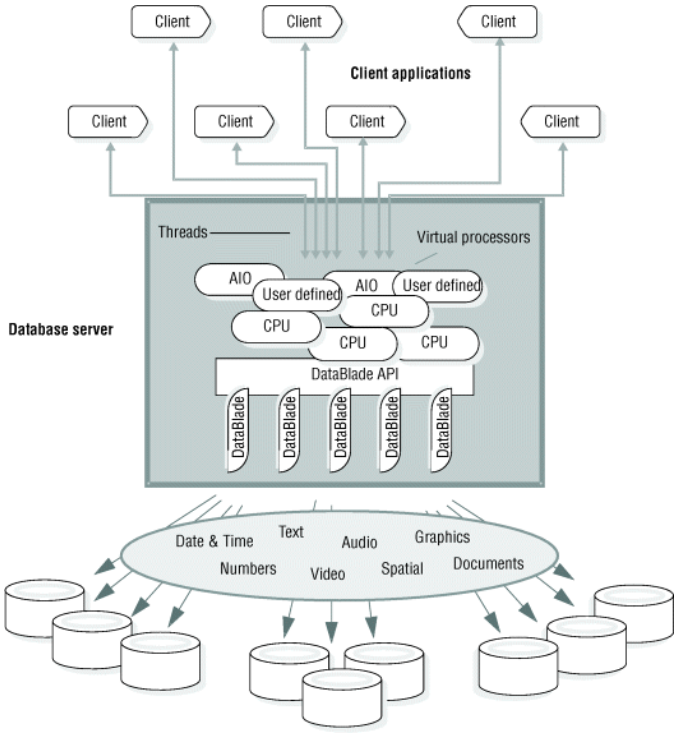
The following table shows the classes of virtual processors and the types of processing that they do.

The number of virtual processors of each class that you configure depends on the availability of physical processors (CPUs), hardware memory, and the database applications in use.

Table 1. Virtual-processor classes

Virtual- processor class	**Category**	**Purpose**
ADM	Administrative	Performs administrative functions.
ADT	Auditing	Performs auditing functions.
AIO	Disk I/O	Performs nonlogging disk I/O. If KAIO is used, AIO virtual processors perform I/O to cooked disk spaces.
BTS	Basic text searching	Runs basic text search index operations and queries.
CPU	Central processing	Runs all session threads and some system threads. Runs thread for kernel asynchronous I/O (KAIO) where available. Can run a single poll thread, depending on configuration.
CSM	Communications Support Module	Performs communications support service operations.
dwavp	Data warehousing	Runs the administrative functions and procedures for Informix® Warehouse Accelerator on a database server that is connected to Informix Warehouse Accelerator.
Encrypt	Encryption	Used by the database server when encryption or decryption functions are called. On Windows systems, the number of **encrypt** virtual processors is always set to 1, regardless of the value that is set in the onconfig file.
IDSXMLVP	XML publishing	Runs XML publishing functions.
JVP	Java™ UDR	Runs Java UDRs. Contains the Java Virtual Machine (JVM).
LIO	Disk I/O	Writes to the logical-log files (internal class) if they are in cooked disk space.
MQ	MQ messaging	Performs MQ messaging transactions.
MSC	Miscellaneous	Services requests for system calls that require a very large stack.
PIO	Disk I/O	Writes to the physical-log file (internal class) if it is in cooked disk space.
SHM	Network	Performs shared memory communication.
SOC	Network	Uses sockets to perform network communication.
tenant	Multitenancy	Runs session threads for tenant databases. Tenant virtual processors are a special case of user-defined processors that are specific to tenant databases.
TLI	Network	Uses the Transport Layer Interface (TLI) to perform network communication.
WFSVP	Web feature service	Runs web feature service routines.
classname	User defined	Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected.

The following figure illustrates the major components and the extensibility of the database server.
Figure 1. Database server



- CPU virtual processors
  The CPU virtual processor runs all session threads (the threads that process requests from SQL client applications) and some internal threads.
- User-defined classes of virtual processors
  You can define special classes of virtual processors to run user-defined routines or to run a DataBlade module.
- Tenant virtual processor class
  Tenant virtual processor classes are specific to tenant databases. If you configure multitenancy for your Informix instance, you can specify that session threads for tenant databases are run in tenant virtual processors instead of CPU virtual processors.
- Java virtual processors
  Java UDRs and Java applications run on specialized virtual processors, called *Java virtual processors* (JVPs).
- Disk I/O virtual processors
  The following classes of virtual processors perform disk I/O: PIO (physical-log I/O), LIO (logical-log I/O), AIO (asynchronous I/O), and CPU (kernel-asynchronous I/O).
- Network virtual processors
  A client can connect to the database server in the through following ways: a network connection, a pipe, or shared memory.
- Communications support module virtual processor
  The communications support module (CSM) class of virtual processors performs communications support service and communications support module functions.
- Encrypt virtual processors
  Use the VPCLASS configuration parameter with the `encrypt` keyword to configure encryption VPs.
- Audit virtual processor
  The database server starts one virtual processor in the audit class (ADT) when you turn on audit mode by setting the ADTMODE parameter in the onconfig file to `1`.
- Miscellaneous virtual processor
  The miscellaneous virtual processor services requests for system calls that might require a very large stack, such as fetching information about the current user or the host-system name.
- Basic text search virtual processors
  A basic text search virtual processor is required to run basic text search queries.
- MQ messaging virtual processor
  An MQ virtual processor is required to use MQ messaging.
- Web feature service virtual processor
  A web feature service virtual processor is required to use web feature service for geospatial data.
- XML virtual processor
  An XML virtual processor is required to perform XML publishing.

Related concepts:
Start and stop virtual processors
Related information:
VPCLASS configuration parameter

CPU virtual processors

The CPU virtual processor runs all session threads (the threads that process requests from SQL client applications) and some internal threads.

Internal threads perform services that are internal to the database server. For example, a thread that listens for connection requests from client applications is an internal thread.

Each CPU virtual processor can have a private memory cache associated with it. Each private memory cache block consists of 1 to 32 memory pages, where each memory page is 4096 bytes. The database server uses the private memory cache to improve access time to memory blocks. Use the VP_MEMORY_CACHE_KB configuration parameter to enable a private memory cache and specify information about the memory cache. For more information, see the *IBM® Informix® Administrator's Reference* and the *IBM Informix Performance Guide*.

- Determine the number of CPU virtual processors needed
  The right number of CPU virtual processors is the number at which they are all kept busy but not so busy that they cannot keep pace with incoming requests. You must not allocate more CPU virtual processors than the number of hardware processors in the computer.
- Run on a multiprocessor computer
  If you are running multiple CPU virtual processors on a multiprocessor computer, set the MULTIPROCESSOR parameter in the onconfig file to `1`.
- Run on a single-processor computer
  If you are running the database server on a single-processor computer, set the MULTIPROCESSOR configuration parameter to `0`. To run the database server with only one CPU virtual processor, set the SINGLE_CPU_VP parameter to `1`.
- Add and drop CPU virtual processors in online mode
  You can add or drop CPU class virtual processors while the database server is online.
- Prevent priority aging
  Some operating systems lower the priority of long-running processes as they accumulate processing time, a feature of the operating system known as *priority aging*.
- Processor affinity
  The database server supports automatic binding of CPU virtual processors to processors on multiprocessor computers that support *processor affinity*.

Determine the number of CPU virtual processors needed

The right number of CPU virtual processors is the number at which they are all kept busy but not so busy that they cannot keep pace with incoming requests. You must not allocate more CPU virtual processors than the number of hardware processors in the computer.

When the database server starts, the number of CPU virtual processors is automatically increased to half the number of CPU processors on the database server computer, unless the SINGLE_CPU_VP configuration parameter is enabled. However, you can adjust the number of CPU VPs based on your system.

You can configure the database server to automatically add CPU VPs when needed, up to the number of CPU processors.

To evaluate the performance of the CPU virtual processors while the database server is running, repeat the following command at regular intervals over a set period:

```
onstat -g glo
```

If the accumulated **usercpu** and **syscpu** times, taken together, approach 100 percent of the actual elapsed time for the period of the test, add another CPU virtual processor if you have a CPU available to run it.

Use the VPCLASS configuration parameter to specify the following information about CPU virtual processors:

- The number of virtual processors to start initially for a class
- The maximum number of virtual processors to run for the class
- Processor affinity for CPU class virtual processors
- Disabling of priority aging, if applicable
- Whether the database server automatically adds CPU virtual processors as needed

In addition to considering the number of CPUs in the computer and the number of users who connect to the database server, also consider that user-defined routines and DataBlade modules, which are collections of user-defined routines, run on either CPU virtual processors or user-defined virtual processors.

Note: Use the VPCLASS configuration parameter instead of the following discontinued configuration parameters: AFF_SPROC, AFFNPROCS, NOAGE, NUMCPUVPS, and NUMAIOVPS.

Related reference:
Run poll threads on CPU or network virtual processors
Assign a UDR to a user-defined virtual-processor class
Related information:
VPCLASS configuration parameter
onstat -g glo command: Print global multithreading information

Run on a multiprocessor computer

If you are running multiple CPU virtual processors on a multiprocessor computer, set the MULTIPROCESSOR parameter in the onconfig file to `1`.

When you set MULTIPROCESSOR to `1`, the database server performs locking in a manner that is appropriate for a multiprocessor computer. For information about setting multiprocessor mode, see the chapter on configuration parameters in the *IBM® Informix® Administrator's Reference*.

Run on a single-processor computer

If you are running the database server on a single-processor computer, set the MULTIPROCESSOR configuration parameter to `0`. To run the database server with only one CPU virtual processor, set the SINGLE_CPU_VP parameter to `1`.

Setting MULTIPROCESSOR to `0` enables the database server to bypass the locking that is required for multiple processes on a multiprocessor computer. For information about the MULTIPROCESSOR configuration parameter, see the *IBM® Informix® Administrator's Reference*.

Setting SINGLE_CPU_VP to `1` allows the database server to bypass some of the mutex calls that it normally makes when it runs multiple CPU virtual processors. For information about setting the SINGLE_CPU_VP parameter, see the *IBM Informix Administrator's Reference*.
Important: Setting *VPCLASS num* to `1` and SINGLE_CPU_VP to `0` does not reduce the number of mutex calls, even though the database server starts only one CPU virtual processor. You must set SINGLE_CPU_VP to `1` to reduce the amount of latching that is performed when you run a single CPU virtual processor.
Setting the SINGLE_CPU_VP parameter to `1` imposes two important restrictions on the database server, as follows:

- Only one CPU virtual processor is allowed.
  You cannot add CPU virtual processors while the database server is in online mode.
- No user-defined classes are allowed. (However, users can still define routines that run directly on the CPU VP.)

For more information, see Add virtual processors in online mode.

Add and drop CPU virtual processors in online mode

You can add or drop CPU class virtual processors while the database server is online.

For instructions on how to add or drop CPU class virtual processors, see Add virtual processors in online mode and Drop CPU and user-defined virtual processors.

Prevent priority aging

Some operating systems lower the priority of long-running processes as they accumulate processing time, a feature of the operating system known as *priority aging*.

Priority aging can cause the performance of database server processes to decline over time. In some cases, however, you can use the operating system to disable this feature and keep long-running processes running at a high priority.

To determine if priority aging is available on your computer, check the machine notes file that comes with your installation and is described in the Introduction to this guide.

If you can disable priority aging through the operating system, you can disable it by specifying `noage` for the priority entry in the VPCLASS configuration parameter. For more information, see the *IBM® Informix® Administrator's Reference*.

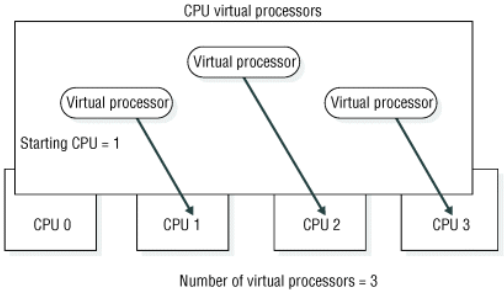
Processor affinity

The database server supports automatic binding of CPU virtual processors to processors on multiprocessor computers that support *processor affinity*.

Your database server distribution includes a machine notes file that contains information about whether your database server version supports this feature. When you assign a CPU virtual processor to a specific CPU, the virtual processor runs only on that CPU, but other processes also can run on that CPU.

Use the VPCLASS configuration parameter with the *aff* option to implement processor affinity on multiprocessor computers that support it.

The following figure illustrates the concept of processor affinity.
Figure 1. Processor affinity



UNIX only: To see if processor affinity is supported on your UNIX platform, see the machine notes file.

- Set processor affinity with the VPCLASS configuration parameter
  To set processor affinity with the VPCLASS configuration parameter, you can specify individual processors or ranges of processors that you want to assign the virtual processors.

Set processor affinity with the VPCLASS configuration parameter

To set processor affinity with the VPCLASS configuration parameter, you can specify individual processors or ranges of processors that you want to assign the virtual processors.

When specifying a range of processors, you can also specify an incremental value with the range that indicates which CPUs in the range are assigned to the virtual processors. For example, you can specify that the virtual processors are assigned to every other CPU in the range 0-6, starting with CPU 0.

```
VPCLASS CPU,num=4,aff=(0-6/2)
```

The virtual processors are assigned to CPUs 0, 2, 4, 6.
If you specify `VPCLASS CPU,num=4,aff=(1-10/3)`, the virtual processors are assigned to every third CPU in the range 1-10, starting with CPU 1. The virtual processors are assigned to CPUs 1, 4, 7, 10.

When you specify more than one value or range, the values and ranges are not required to be incremental or in any particular order. For example you can specify `aff=(8,12,7-9,0-6/2)`.

The database server assigns CPU virtual processors to CPUs in a circular pattern, starting with the first processor number that you specify in the `aff` option. If you specify a larger number of CPU virtual processors than physical CPUs, the database server continues to assign CPU virtual processors starting with the first CPU. For example, suppose you specify the following VPCLASS settings:

```
VPCLASS cpu,num=8,aff=(4-7)
```

The database server makes the following assignments:

- CPU virtual processor number `0` to CPU `4`
- CPU virtual processor number `1` to CPU `5`
- CPU virtual processor number `2` to CPU `6`
- CPU virtual processor number `3` to CPU `7`
- CPU virtual processor number `4` to CPU `4`
- CPU virtual processor number `5` to CPU `5`
- CPU virtual processor number `6` to CPU `6`
- CPU virtual processor number `7` to CPU `7`

For more information, see the VPCLASS configuration parameter in the *IBM® Informix® Administrator's Reference*.

User-defined classes of virtual processors

You can define special classes of virtual processors to run user-defined routines or to run a DataBlade module.

User-defined routines are typically written to support user-defined data types. If you do not want a user-defined routine to run in the CPU class, which is the default, you can assign it to a user-defined class of virtual processors (VPs). User-defined classes of virtual processors are also called *extension virtual processors*.

These topics provide the following information about user-defined virtual processors:

- When to run a C-language UDR in a user-defined VP instead of in the CPU VP
- How to assign a C-language UDR to a particular user-defined VP class
- How to add and drop user-defined VPs when the database server is in online mode

- Determine the number of user-defined virtual processors needed
  You can specify as many user-defined virtual processors as your operating system allows.
- User-defined virtual processors
  User-defined classes of virtual processors protect the database server from *ill-behaved* user-defined routines.
- Specify user-defined virtual processors
  The VPCLASS parameter with the *vpclass* option defines a user-defined VP class. You also can specify a nonyielding user-defined virtual processor.
- Assign a UDR to a user-defined virtual-processor class
  The SQL CREATE FUNCTION statement registers a user-defined routine.
- Add and drop user-defined virtual processors in online mode
  You can add or drop virtual processors in a user-defined class while the database server is online.

Determine the number of user-defined virtual processors needed

You can specify as many user-defined virtual processors as your operating system allows.

If you run many UDRs or parallel PDQ queries with UDRs, you must configure more user-defined virtual processors.

User-defined virtual processors

User-defined classes of virtual processors protect the database server from *ill-behaved* user-defined routines.

An ill-behaved user-defined routine has at least one of the following characteristics:

- Does not yield control to other threads
- Makes blocking operating-system calls
- Modifies the global VP state

A well-behaved C-language UDR has none of these characteristics. Run only well-behaved C-language UDRs in a CPU VP.
Warning: Execution of an ill-behaved routine in a CPU VP can cause serious interference with the operation of the database server, possibly causing it to fail or behave erratically. In addition, the routine itself might not produce correct results.
To ensure safe execution, assign any ill-behaved user-defined routines to a user-defined class of virtual processors. User-defined VPs remove the following programming restrictions on the CPU VP class:

- The requirement to yield the processor regularly
- The requirement to eliminate blocking I/O calls

Functions that run in a user-defined virtual-processor class are not required to yield the processor, and they might issue direct file-system calls that block further processing by the virtual processor until the I/O is complete.

The normal processing of user queries is not affected by ill-behaved traits of a C-language UDR because these UDRs do not execute in CPU virtual processors. For a more detailed explanation of ill-behaved routines, see the *IBM® Informix® DataBlade API Programmer's Guide*.

Specify user-defined virtual processors

The VPCLASS parameter with the *vpclass* option defines a user-defined VP class. You also can specify a nonyielding user-defined virtual processor.

For more information, see Set virtual-processor configuration parameters and the topics about configuration parameters in the *IBM® Informix® Administrator's Reference*.

Assign a UDR to a user-defined virtual-processor class

The SQL CREATE FUNCTION statement registers a user-defined routine.

The following CREATE FUNCTION statement registers the user-defined routine, **GreaterThanEqual()**, and specifies that calls to this routine are executed by the user-defined VP class named UDR:

```
CREATE FUNCTION GreaterThanEqual(ScottishName, ScottishName)
   RETURNS boolean
   WITH (CLASS = UDR  )
   EXTERNAL NAME '/usr/lib/objects/udrs.so'
   LANGUAGE C
```

To execute this function, the onconfig file must include a VPCLASS parameter that defines the UDR class. If not, calls to the **GreaterThanEqual** function fail.
Tip: The CLASS routine modifier can specify any name for the VP class. This class name is not required to exist when you register the UDR. However, when you try to run a UDR that specifies a user-defined VP class for its execution, this class must exist and have virtual processors assigned to it.
To configure the UDR class, include a line similar to the following one in the onconfig file. This line configures the UDR class with two virtual processors and with no priority aging.

```
VPCLASS     UDR  ,num=2,noage
```

The preceding line defines the UDR VP class as a yielding VP class; that is, this VP class allows the C-language UDR to yield to other threads that must access to the UDR VP class. For more information about how to use the VPCLASS configuration parameter, see the *IBM® Informix® Administrator's Reference*.

For more information about the CREATE FUNCTION statement, see the *IBM Informix Guide to SQL: Syntax*.

Related reference:
Determine the number of CPU virtual processors needed

Add and drop user-defined virtual processors in online mode

You can add or drop virtual processors in a user-defined class while the database server is online.

For instructions on how to do this, see Add virtual processors in online mode and Drop CPU and user-defined virtual processors.

Tenant virtual processor class

Tenant virtual processor classes are specific to tenant databases. If you configure multitenancy for your Informix® instance, you can specify that session threads for tenant databases are run in tenant virtual processors instead of CPU virtual processors.

You can create a tenant virtual processor class by defining the class and the number of virtual processors when you create a tenant database. You can assign a tenant virtual processor class to multiple tenant databases. Set the VP_MEMORY_CACHE_KB configuration parameter to create a private memory cache for each CPU virtual processor and tenant virtual processor.

A tenant virtual processor class is automatically dropped when all tenant databases that include the virtual processor class in their definitions are dropped.

Related concepts:
Multitenancy
Related information:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
VP_MEMORY_CACHE_KB configuration parameter
onstat -g vpcache command: Print CPU virtual processor and tenant virtual processor private memory cache statistics

Java virtual processors

Java™ UDRs and Java applications run on specialized virtual processors, called *Java virtual processors* (JVPs).

A JVP embeds a Java virtual machine (JVM) in its code. A JVP has the same capabilities as a CPU VP in that it can process complete SQL queries.

You can specify as many JVPs as your operating system allows. If you run many Java UDRs or parallel PDQ queries with Java UDRs, you must configure more JVPs. For more information about UDRs written in Java, see *IBM® J/Foundation Developer's Guide*.

Use the VPCLASS configuration parameter with the `jvp` keyword to configure JVPs. For more information, see the configuration parameters chapter in the *IBM Informix® Administrator's Reference*.

Disk I/O virtual processors

The following classes of virtual processors perform disk I/O: PIO (physical-log I/O), LIO (logical-log I/O), AIO (asynchronous I/O), and CPU (kernel-asynchronous I/O).

The PIO class performs all I/O to the physical-log file, and the LIO class performs all I/O to the logical-log files, *unless* those files are in raw disk space and the database server has implemented KAIO.

On operating systems that do not support KAIO, the database server uses the AIO class of virtual processors to perform database I/O that is not related to physical or logical logging.

The database server uses the CPU class to perform KAIO when it is available on a platform. If the database server implements KAIO, a KAIO thread performs all I/O to raw disk space, including I/O to the physical and logical logs.

UNIX only: To find out if your UNIX platform supports KAIO, see the machine notes file.
Windows only: Windows supports KAIO.
For more information about nonlogging I/O, see Asynchronous I/O.

- I/O priorities
  The database server prioritizes disk I/O by assigning different types of I/O to different classes of virtual processors and by assigning priorities to the nonlogging I/O queues.
- Logical-log I/O
  The LIO class of virtual processors performs I/O to the logical-log files.
- Physical-log I/O
  The PIO class of virtual processors performs I/O to the physical-log file.
- Asynchronous I/O
  The database server performs database I/O asynchronously, meaning that I/O is queued and performed independently of the thread that requests the I/O. Performing I/O asynchronously allows the thread that makes the request to continue working while the I/O is being performed.

I/O priorities

The database server prioritizes disk I/O by assigning different types of I/O to different classes of virtual processors and by assigning priorities to the nonlogging I/O queues.

Prioritizing ensures that a high-priority log I/O, for example, is never queued behind a write to a temporary file, which has a low priority. The database server prioritizes the different types of disk I/O that it performs, as the table shows.

Table 1. How database server prioritizes disk I/O

Priority	Type of I/O	VP class
1st	Logical-log I/O	CPU or LIO

Priority	Type of I/O	VP class
2nd	Physical-log I/O	CPU or PIO
3rd	Database I/O	CPU or AIO
3rd	Page-cleaning I/O	CPU or AIO
3rd	Read-ahead I/O	CPU or AIO

Logical-log I/O

The LIO class of virtual processors performs I/O to the logical-log files.

I/O is performed to logical-log files in the following cases:

- KAIO is not implemented.
- The logical-log files are in cooked disk space.

Only when KAIO is implemented and the logical-log files are in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the logical log.

The logical-log files store the data that enables the database server to roll back transactions and recover from system failures. I/O to the logical-log files is the highest priority disk I/O that the database server performs.

If the logical-log files are in a dbspace that is **not** mirrored, the database server runs only one LIO virtual processor. If the logical-log files are in a dbspace that is mirrored, the database server runs two LIO virtual processors. This class of virtual processors has no parameters associated with it.

Physical-log I/O

The PIO class of virtual processors performs I/O to the physical-log file.

I/O is performed to the physical-log file in the following cases:

- KAIO is not implemented.
- The physical-log file is stored in buffered-file chunks.

Only when KAIO is implemented and the physical-log file is in raw disk space does the database server use a KAIO thread in the CPU virtual processor to perform I/O to the physical log. The physical-log file stores *before-images* of dbspace pages that have changed since the last *checkpoint.* (For more information about checkpoints, see Checkpoints.) At the start of recovery, before processing transactions from the logical log, the database server uses the physical-log file to restore before-images to dbspace pages that have changed since the last checkpoint. I/O to the physical-log file is the second-highest priority I/O after I/O to the logical-log files.

If the physical-log file is in a dbspace that is **not** mirrored, the database server runs only one PIO virtual processor. If the physical-log file is in a dbspace that is mirrored, the database server runs two PIO virtual processors. This class of virtual processors has no parameters associated with it.

Asynchronous I/O

The database server performs database I/O asynchronously, meaning that I/O is queued and performed independently of the thread that requests the I/O. Performing I/O asynchronously allows the thread that makes the request to continue working while the I/O is being performed.

The database server performs all database I/O asynchronously, using one of the following facilities:

- AIO virtual processors
- KAIO on platforms that support it

Database I/O includes I/O for SQL statements, read-ahead, page cleaning, and checkpoints.

- Kernel-asynchronous I/O
  The database server implements KAIO by running a KAIO thread on the CPU virtual processor. The KAIO thread performs I/O by making system calls to the operating system, which performs the I/O independently of the virtual processor.
- AIO virtual processors
  If the platform does not support KAIO or if the I/O is to buffered-file chunks, the database server performs database I/O through the AIO class of virtual processors. All AIO virtual processors service all I/O requests equally within their class.

Kernel-asynchronous I/O

The database server implements KAIO by running a KAIO thread on the CPU virtual processor. The KAIO thread performs I/O by making system calls to the operating system, which performs the I/O independently of the virtual processor.

The database server uses KAIO when the following conditions exist:

- The computer and operating system support it.
- A performance gain is realized.
- The I/O is to raw disk space.

The KAIO thread can produce better performance for disk I/O than the AIO virtual processor can, because it does not require a switch between the CPU and AIO virtual processors.

UNIX only: IBM® Informix® implements KAIO when ports to a platform that supports this feature. The database server administrator does not configure KAIO. To see if KAIO is supported on your platform, see the machine notes file.
Linux only: Kernel asynchronous I/O (KAIO) is enabled by default. You can disable this by specifying that `KAIOOFF=1` in the environment of the process that starts the server.
On Linux, there is a system-wide limit of the maximum number of parallel KAIO requests. The /proc/sys/fs/aio-max-nr file contains this value. The Linux system administrator can increase the value, for example, by using this command:

```
# echo new_value > /proc/sys/fs/aio-max-nr
```

The current number of allocated requests of all operating system processes is visible in the /proc/sys/fs/aio-nr file.

By default, Dynamic Version allocates half of the maximum number of requests and assigns them equally to the number of configured CPU virtual processors. You can use the environment variable KAIOON to control the number of requests allocated per CPU virtual processor. Do this by setting KAIOON to the required value before starting .

The minimum value for KAIOON is `100`. If Linux is about to run out of KAIO resources, for example when dynamically adding many CPU virtual processors, warnings are printed in the online.log file. If this happens, the Linux system administrator must add KAIO resources as described previously.

AIO virtual processors

If the platform does not support KAIO or if the I/O is to buffered-file chunks, the database server performs database I/O through the AIO class of virtual processors. All AIO virtual processors service all I/O requests equally within their class.

The database server assigns each disk chunk a queue, sometimes known as a *gfd queue*, which is based on the file name of the chunk. The database server orders I/O requests within a queue according to an algorithm that minimizes disk-head movement. The AIO virtual processors service queues that have pending work in round-robin fashion. All other non-chunk I/O is queued in the AIO queue.

Use the VPCLASS parameter with the **aio** keyword to specify the number of AIO virtual processors that the database server starts initially. You can start additional AIO virtual processors while the database server is in online mode. You cannot drop AIO virtual processors while the database server is in online mode.

You can enable the database server to add AIO virtual processors and flusher threads when the server detects that AIO VPs are not keeping up with the I/O workload. Include the **autotune=1** keyword in the VPCLASS configuration parameter setting.

Manually controlling the number of AIO VPs

The goal in allocating AIO virtual processors is to allocate enough of them so that the lengths of the I/O request queues are kept short; that is, the queues have as few I/O requests in them as possible. When the *gfd* queues are consistently short, it indicates that I/O to the disk devices is being processed as fast as the requests occur.

The **onstat-g ioq** command shows the length and other statistics about I/O queues. You can use this command to monitor the length of the *gfd* queues for the AIO virtual processors.

One AIO virtual processor might be sufficient:

- If the database server implements kernel asynchronous I/O (KAIO) on your platform and all of your dbspaces are composed of raw disk space
- If your file system supports direct I/O for the page size that is used for the dbspace chunk and you use direct I/O

Allocate two AIO virtual processors per active dbspace that is composed of buffered file chunks.

- If the database server implements KAIO, but you are using some buffered files for chunks
- IF KAIO is not supports by the system for chunks.

If KAIO is not implemented on your platform, allocate two AIO virtual processors for each disk that the database server accesses frequently.

If you use cooked files and if you enable direct I/O using the DIRECT_IO configuration parameter, you might be able to reduce the number of AIO virtual processors.

If the database server implements KAIO and you enabled direct I/O using the DIRECT_IO configuration parameter, IBM® Informix® attempts to use KAIO, so you probably do not require more than one AIO virtual processor. However, even when direct I/O is enabled, if the file system does not support either direct I/O or KAIO, you still must allocate two AIO virtual processors for every active dbspace that is composed of buffered file chunks or does not use KAIO.

Temporary dbspaces do not use direct I/O. If you have temporary dbspaces, you probably require more than one AIO virtual processors.

Allocate enough AIO virtual processors to accommodate the peak number of I/O requests. Generally, it is not detrimental to allocate too many AIO virtual processors.

Related information:
VPCLASS configuration parameter

Network virtual processors

A client can connect to the database server in the through following ways: a network connection, a pipe, or shared memory.

The network connection can be made by a client on a remote computer or by a client on the local computer mimicking a connection from a remote computer (called a *local-loopback connection*).

- Specifying Network Connections
  In general, the DBSERVERNAME and DBSERVERALIASES parameters define dbservernames that have corresponding entries in the sqlhosts file or registry. Each dbservername parameter in sqlhosts has a **nettype** entry that specifies an interface/protocol combination. The database server runs one or more poll threads for each unique **nettype** entry.
- Run poll threads on CPU or network virtual processors
  Poll threads can run either on CPU virtual processors or on network virtual processors.
- Specify the number of networking virtual processors
  Each poll thread requires a separate virtual processor, so you indirectly specify the number of networking virtual processors when you specify the number of poll threads for an interface/protocol combination and specify that they are to be run by the NET class.
- Specify listen and poll threads for the client/server connection
  When you start the database server, the **oninit** process starts an internal thread, called a *listen thread*, for each dbservername that you specify with the DBSERVERNAME and DBSERVERALIASES parameters in the onconfig file.
- Fast polling
  You can use the FASTPOLL configuration parameter to enable or disable fast polling of your network, if your operating-system platform supports fast polling.
- Multiple listen threads
  You can improve service for connection requests by using multiple listen threads.

Specifying Network Connections

In general, the DBSERVERNAME and DBSERVERALIASES parameters define dbservernames that have corresponding entries in the sqlhosts file or registry. Each dbservername parameter in sqlhosts has a **nettype** entry that specifies an interface/protocol combination. The database server runs one or more poll threads for each unique **nettype** entry.

The NETTYPE configuration parameter provides optional configuration information for an interface/protocol combination. You can use it to allocate more than one poll thread for an interface/protocol combination and also designate the virtual-processor class (CPU or NET) on which the poll threads run.

For a complete description of the NETTYPE configuration parameter, see the *IBM® Informix® Administrator's Reference*.

Related reference:
sqlhosts connectivity information

Run poll threads on CPU or network virtual processors

Poll threads can run either on CPU virtual processors or on network virtual processors.

In general, and particularly on a single-processor computer, poll threads run more efficiently on CPU virtual processors. This might not be true, however, on a multiprocessor computer with many remote clients.

The NETTYPE parameter has an optional entry, called *vp class*, which you can use to specify either CPU or NET, for CPU or network virtual-processor classes, respectively.

If you do not specify a virtual processor class for the interface/protocol combination (poll threads) associated with the DBSERVERNAME variable, the class defaults to CPU. The database server assumes that the interface/protocol combination associated with DBSERVERNAME is the primary interface/protocol combination and that it is the most efficient.

For other interface/protocol combinations, if no vp class is specified, the default is NET.

While the database server is in online mode, you cannot drop a CPU virtual processor that is running a poll or a listen thread.

Important: You must carefully distinguish between poll threads for network connections and poll threads for shared memory connections, which run one per CPU virtual processor. TCP connections must only be in network virtual processors, and you must only have the minimum required to maintain responsiveness. Shared memory connections must only be in CPU virtual processors and run in every CPU virtual processor.
Related reference:
Determine the number of CPU virtual processors needed

Specify the number of networking virtual processors

Each poll thread requires a separate virtual processor, so you indirectly specify the number of networking virtual processors when you specify the number of poll threads for an interface/protocol combination and specify that they are to be run by the NET class.

If you specify CPU for the `vp class`, you must allocate a sufficient number of CPU virtual processors to run the poll threads. If the database server does not have a CPU virtual processor to run a CPU poll thread, it starts a network virtual processor of the specified class to run it.

For most systems, one poll thread and consequently one virtual processor per network interface/protocol combination is sufficient. For systems with 200 or more network users, running additional network virtual processors might improve throughput. In this case, you must experiment to determine the optimal number of virtual processors for each interface/protocol combination.

Specify listen and poll threads for the client/server connection

When you start the database server, the **oninit** process starts an internal thread, called a *listen thread*, for each dbservername that you specify with the DBSERVERNAME and DBSERVERALIASES parameters in the onconfig file.

To specify a listen port for each of these dbservername entries, assign it a unique combination of **hostname** and **service name** entries in sqlhosts. For example, the sqlhosts file or registry entry shown in the following table causes the database server **soc_ol1** to start a listen thread for **port1** on the host, or network address, **myhost**.

Table 1. A listen thread for each listen port

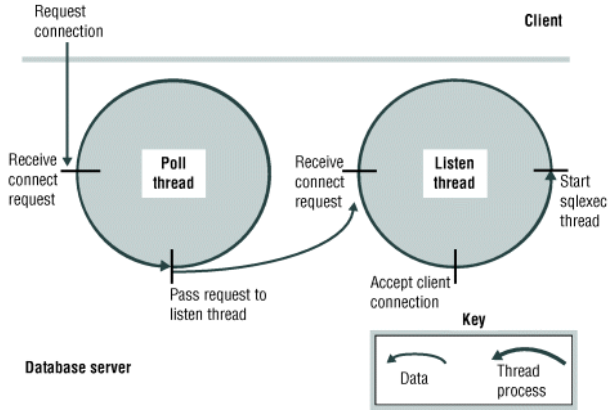
dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1

The listen thread opens the port and requests one of the poll threads for the specified interface/protocol combination to monitor the port for client requests. The poll thread runs either in the CPU virtual processor or in the network virtual processor for the connection that is being used. For information about the number of poll threads, see Specify the number of networking virtual processors.

For information about how to specify whether the poll threads for an interface/protocol combination run in CPU or network virtual processors, see Run poll threads on CPU or network virtual processors and to the NETTYPE configuration parameter in the *IBM® Informix® Administrator's Reference*.

When a poll thread receives a connection request from a client, it passes the request to the listen thread for the port. The listen thread authenticates the user, establishes the connection to the database server, and starts an **sqlexec** thread, the session thread that performs the primary processing for the client. The following figure illustrates the roles of the listen and poll threads in establishing a connection with a client application.

Figure 1. The roles of the poll and the listen threads in connecting to a client

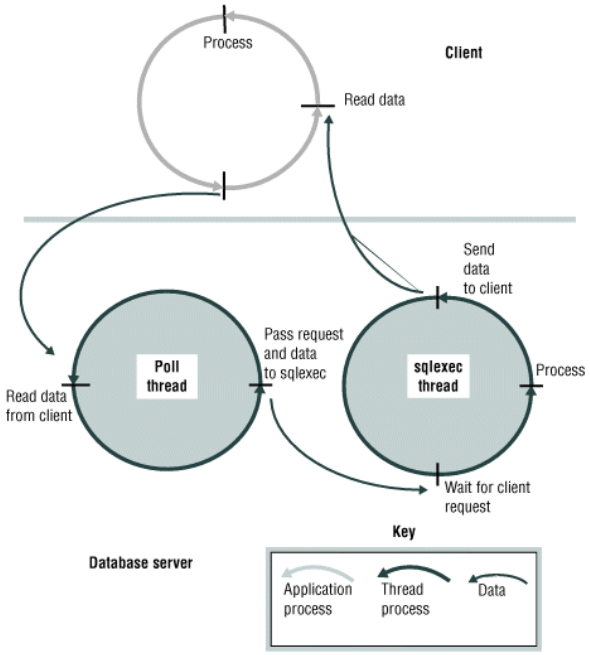


A poll thread waits for requests from the client and places them in shared memory to be processed by the **sqlexec** thread. For network connections, the poll thread places the message in a queue in the shared-memory global pool. The poll thread then wakes up the **sqlexec** thread of the client to process the request. Whenever possible, the **sqlexec** thread writes directly back to the client without the help of the poll thread. In general, the poll thread reads data from the client, and the **sqlexec** thread sends data to the client.

UNIX only: For a shared-memory connection, the poll thread places the message in the communications portion of shared memory.

The following figure illustrates the basic tasks that the poll thread and the **sqlexec** thread perform in communicating with a client application.

Figure 2. The roles of the poll and sqlexec threads in communicating with the client application



Fast polling

You can use the FASTPOLL configuration parameter to enable or disable fast polling of your network, if your operating-system platform supports fast polling.

Fast polling is beneficial if you have many connections. For example, if you have more than 300 concurrent connections with the database server, you can enable the FASTPOLL configuration parameter for better performance. You can enable fast polling by setting the FASTPOLL configuration parameter to `1`.

If your operating system does not support fast polling, IBM® Informix® ignores the FASTPOLL configuration parameter.

Multiple listen threads

You can improve service for connection requests by using multiple listen threads.

If the database server cannot service connection requests satisfactorily for a given interface/protocol combination with a single port and corresponding listen thread, you can improve service for connection requests in the following ways:

- By adding listen threads for additional ports.
- By adding listen threads to the same port if you have the **onimcsoc** or **onsoctcp** protocol
- By adding another network-interface card.
- By dynamically starting, stopping, or restarting listen threads for a SOCTCP or TLITCP network protocol, using SQL administration API or **onmode -P** commands.

If you have multiple listen threads for one port for the **onsoctcp** protocol, the database server can accept new connections if a CPU VP connection is busy.

- Add listen threads
  When you start the database server, the **oninit** process starts a listen thread for servers with the server names and server alias names that you specify with the DBSERVERNAME and DBSERVERALIASES configuration parameters. You can add listen threads for additional ports.
- Add a network-interface card
  You can add a network-interface card to improve performance or connect the database server to multiple networks.
- Dynamically starting, stopping, or restarting a listen thread
  You can dynamically start, stop, or stop and start a listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections. For example, you might want to stop listen threads that are unresponsive and then start new ones in situations when other server functions are performing normally and you do not want to shut down the server.

Add listen threads

When you start the database server, the **oninit** process starts a listen thread for servers with the server names and server alias names that you specify with the DBSERVERNAME and DBSERVERALIASES configuration parameters. You can add listen threads for additional ports.

You can also set up multiple listen threads for one service (port) for the **onimcsoc** or **onsoctcp** protocol.

To add listen threads for additional ports, you must first use the DBSERVERALIASES parameter to specify dbservernames for each of the ports. For example, the DBSERVERALIASES parameter in the following figure defines two additional dbservernames, **soc_ol2** and **soc_ol3**, for the database server instance identified as **soc_ol1**.

```
DBSERVERNAME      soc_ol1
DBSERVERALIASES   soc_ol2,soc_ol3
```

After you define additional dbservernames for the database server, you must specify an interface/protocol combination and port for each of them in the sqlhosts file or registry. Each port is identified by a unique combination of **hostname** and **servicename** entries. For example, the sqlhosts entries shown in the following table cause the database server to start three listen threads for the **onsoctcp** interface/protocol combination, one for each of the ports defined.

Table 1. The sqlhosts entries to listen to multiple ports for a single interface/protocol combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost	port1
soc_ol2	onsoctcp	myhost	port2
soc_ol3	onsoctcp	myhost	port3

If you include a NETTYPE parameter for an interface/protocol combination, it applies to all the connections for that interface/protocol combination. In other words, if a NETTYPE parameter exists for **onsoctcp** in the previous table, it applies to all of the connections shown. In this example, the database server runs one *poll* thread for the **onsoctcp** interface/protocol combination unless the NETTYPE parameter specifies more. For more information about entries in the sqlhosts file or registry, see Connectivity files.

Setting up multiple listen threads for one port for the onimcsoc or onsoctcp protocol

To set up multiple listen threads for one service (port) for the **onimcsoc** or **onsoctcp** protocol, specify DBSERVERNAME and DBSERVERALIASES information as follows:

- DBSERVERNAME <*name*>-<*n*>
- DBSERVERALIASES <*name1*>-<*n*>,<*name2*>

For example:

- To bring up two listen threads for the server with the DBSERVERNAME of `ifx`, specify:

  ```
  DBSERVERNAME ifx-2
  ```

- To bring up two listen threads for DBSERVERALIASES `ifx_a` and `ifx_b`, specify:

  ```
  DBSERVERALIASES ifx_a-2,ifx_b-2
  ```


Add a network-interface card

You can add a network-interface card to improve performance or connect the database server to multiple networks.

You might want to improve performance if the network-interface card for the host computer cannot service connection requests satisfactorily.

To support multiple network-interface cards, you must assign each card a unique **hostname** (network address) in sqlhosts.

For example, using the same dbservernames shown in Add listen threads, the sqlhosts file or registry entries shown in the following table cause the database server to start three listen threads for the same interface/protocol combination (as did the entries in Add listen threads). In this case, however, two of the threads are listening to ports on one interface card (**myhost1**), and the third thread is listening to a port on the second interface card (**myhost2**).

Table 1. Example of sqlhosts entries to support two network-interface cards for the onsoctcp interface/protocol combination

dbservername	nettype	hostname	service name
soc_ol1	onsoctcp	myhost1	port1
soc_ol2	onsoctcp	myhost1	port2
soc_ol3	onsoctcp	myhost2	port1

Dynamically starting, stopping, or restarting a listen thread

You can dynamically start, stop, or stop and start a listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections. For example, you might want to stop listen threads that are unresponsive and then start new ones in situations when other server functions are performing normally and you do not want to shut down the server.

The listen thread must be defined in the sqlhosts file for the server. If necessary, before start, stop, or restart a listen thread, you can revise the sqlhosts entry.
To dynamically start, stop, or restart listen threads:

1. Run one of the following **onmode -P** commands:
   - **onmode -P start** *server_name*
   - **onmode -P stop** *server_name*
   - **onmode -P restart** *server_name*

2. Alternatively, if you are connected to the **sysadmin** database, either directly or remotely, you can run one of the following commands:
   - An **admin()** or **task()** command with the start listen argument, using the format

     ```
     EXECUTE FUNCTION task("start listen", "server_name");
     ```

   - An **admin()** or **task()** command with the stop listen argument, using the format

     ```
     EXECUTE FUNCTION task("stop listen" ,"server_name");
     ```

   - An **admin()** or **task()** command with the restart listen argument, using the format

     ```
     EXECUTE FUNCTION task("restart listen", "server_name");
     ```

For example, either of the following commands starts a new listen thread for a server named **ifx_serv2**:

```
onmode -P start ifx_serv2
```

```
EXECUTE FUNCTION task("start listen", "ifx_serv2");
```


Communications support module virtual processor

The communications support module (CSM) class of virtual processors performs communications support service and communications support module functions.

The database server starts the same number of CSM virtual processors as the number of CPU virtual processors that it starts, unless the communications support module is set to GSSCSM to support single sign-on. When the communications support module is GSSCSM, the database server starts only one CSM virtual processor.

For more information about the communications support service, see Client/server communication.

Encrypt virtual processors

Use the VPCLASS configuration parameter with the `encrypt` keyword to configure encryption VPs.

If the `encrypt` option of the VPCLASS parameter is not defined in the onconfig configuration file, the database server starts one ENCRYPT VP the first time that any encryption or decryption functions defined for column-level encryption are called. You can define multiple ENCRYPT VPs if necessary to decrease the time required to start the database server.

To add five ENCRYPT VPs, add information in the onconfig file as follows:

```
VPCLASS encrypt,num=5
```

You can modify the same information using the **onmode** utility, as follows:

```
onmode -p 5 encrypt
```

For more information, see the configuration parameters and the **onmode** utility topics in the *IBM® Informix® Administrator's Reference*. For more information about column-level encryption, see the *IBM Informix Security Guide*.

Audit virtual processor

The database server starts one virtual processor in the audit class (ADT) when you turn on audit mode by setting the ADTMODE parameter in the onconfig file to `1`.

For more information about database server auditing, see the *IBM® Informix® Security Guide*.

Miscellaneous virtual processor

The miscellaneous virtual processor services requests for system calls that might require a very large stack, such as fetching information about the current user or the host-system name.

Only one thread runs on this virtual processor; it executes with a stack of 128 KB.

Basic text search virtual processors

A basic text search virtual processor is required to run basic text search queries.

A basic text search virtual processor is added automatically when you create a basic text search index.

A basic text search virtual processor runs without yielding; it processes one index operation at a time. To run multiple basic text search index operations and queries simultaneously, create additional basic text search virtual processors.

Use the VPCLASS configuration parameter with the BTS keyword to configure basic text search virtual processors. For example, to add five BTS virtual processors, add the following line to the onconfig and restart the database server:

```
VPCLASS bts,num=5
```

You can dynamically add BTS virtual processors by using the **onmode -p** command, for example:

```
onmode -p 5 bts
```

Related information:
VPCLASS configuration parameter
onmode -p: Add or drop virtual processors
Basic Text Search

MQ messaging virtual processor

An MQ virtual processor is required to use MQ messaging.

When you perform MQ messaging transactions, an MQ virtual processor is created automatically.

An MQ virtual processor runs without yielding; it processes one operation at a time. To perform multiple MQ messaging transactions simultaneously, create additional MQ virtual processors.

Use the VPCLASS configuration parameter with the MQ keyword to configure MQ virtual processors. For example, to add five MQ virtual processors, add the following line to the onconfig and restart the database server:

```
VPCLASS mq,noyield,num=5
```

For more information about the VPCLASS configuration parameter, see the *IBM® Informix® Administrator's Reference*. For more information about MQ messaging, see the *IBM Informix Database Extensions User's Guide*.

Web feature service virtual processor

A web feature service virtual processor is required to use web feature service for geospatial data.

When you run a WFS routine, a WFS virtual processor is created automatically.

A WFS virtual processor runs without yielding; it processes one operation at a time. To run multiple WFS routines simultaneously, create additional WFS virtual processors.

Use the VPCLASS configuration parameter with the WFSVP keyword to configure WFS virtual processors. For example, to add five WFS virtual processors, add the following line to the onconfig and restart the database server:

```
VPCLASS wfsvp,noyield,num=5
```

For more information about the VPCLASS configuration parameter, see the *IBM® Informix® Administrator's Reference*. For more information about WFS, see the *IBM Informix Database Extensions User's Guide*.

XML virtual processor

An XML virtual processor is required to perform XML publishing.

When you run an XML function, an XML virtual processor is created automatically.

An XML virtual processor runs one XML function at a time. To run multiple XML functions simultaneously, create additional XML virtual processors.

Use the VPCLASS configuration parameter with the IDSXMLVP keyword to configure XML virtual processors. For example, to add five XML virtual processors, add the following line to the onconfig and restart the database server:

```
VPCLASS idsxmlvp,num=5
```

You can dynamically add XML virtual processors by using the **onmode -p** command, for example:

```
onmode -p 5 idsxmlvp
```

For more information about the VPCLASS configuration parameter and the **onmode** utility, see the *IBM® Informix® Administrator's Reference*. For more information about XML publishing, see the *IBM Informix Database Extensions User's Guide*.

Manage virtual processors

These topics describe how to set the configuration parameters that affect database server virtual processors, and how to start and stop virtual processors.

For descriptions of the virtual-processor classes and for advice on how many virtual processors you must specify for each class, see Virtual processors and threads.

- Set virtual-processor configuration parameters
  Use the VPCLASS configuration parameter to designate a class of virtual processors (VPs), create a user-defined virtual processor, and specify options such as the number of VPs that the server starts, the maximum number of VPs allowed for the class, and the assignment of VPs to CPUs if processor affinity is available.
- Start and stop virtual processors
  When you start the database server, the **oninit** utility starts the number and types of virtual processors that you specify directly and indirectly.
- Monitor virtual processors
  Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity.

Set virtual-processor configuration parameters

Use the VPCLASS configuration parameter to designate a class of virtual processors (VPs), create a user-defined virtual processor, and specify options such as the number of VPs that the server starts, the maximum number of VPs allowed for the class, and the assignment of VPs to CPUs if processor affinity is available.

The table lists the configuration parameters that are used to configure virtual processors.

Table 1. Configuration parameters for configuring virtual processors

Parameter	Description
MULTIPROCESSOR	Set to `1` to support multiple CPU virtual processors, or to `0` for only a single CPU VP
NETTYPE	Specifies parameters for network protocol threads and virtual processors
SINGLE_CPU_VP	Set to `0` to enable user-defined CPU VPs, or to any other setting for only a single CPU VP
VPCLASS	Each defines a VP class and its properties, such as how many VPs of this class start when the server starts
VP_MEMORY_CACHE_KB	Speeds access to memory blocks by creating a private memory cache for each CPU virtual processor

Related information:
VPCLASS configuration parameter
MULTIPROCESSOR configuration parameter
SINGLE_CPU_VP configuration parameter
VP_MEMORY_CACHE_KB configuration parameter

Start and stop virtual processors

When you start the database server, the **oninit** utility starts the number and types of virtual processors that you specify directly and indirectly.

You configure virtual processors primarily through configuration parameters and, for network virtual processors, through parameters in the sqlhosts information.

You can use the database server to start a maximum of 1000 virtual processors.

After the database server is in online mode, you can start more virtual processors to improve performance, if necessary.

While the database server is in online mode, you can drop virtual processors of the CPU and user-defined classes.

To shut down the database server and stop all virtual processors, use the **onmode -k** command.

- Add virtual processors in online mode
  While the database server is running, you can start additional virtual processors for some virtual processor classes with the **-p** option of the **onmode** utility.
- Drop CPU and user-defined virtual processors
  While the database server is in online mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of the CPU and user-defined classes.

Related concepts:
Virtual processor classes
Related information:
onmode -k, -m, -s, -u, -j: Change database server mode

Add virtual processors in online mode

While the database server is running, you can start additional virtual processors for some virtual processor classes with the **-p** option of the **onmode** utility.

You can start these additional virtual processors with the **-p** option of the **onmode** utility. You can add network virtual processors with the NETTYPE configuration parameter.

You can also start additional virtual processors for user-defined classes to run user-defined routines. For more information about user-defined virtual processors, see Assign a UDR to a user-defined virtual-processor class.

- Add virtual processors in online mode with onmode
  Use the **-p** option of the **onmode** command to add virtual processors while the database server is in online mode.
- Add network virtual processors
  When you add network virtual processors, you add poll threads, each of which requires its own virtual processor to run.

Related information:
onmode -p: Add or drop virtual processors
NETTYPE configuration parameter

Add virtual processors in online mode with onmode

Use the **-p** option of the **onmode** command to add virtual processors while the database server is in online mode.

Specify the number of virtual processors that you want to add with a positive number. As an option, you can precede the number of virtual processors with a plus sign (+). Following the number, specify the virtual processor class in lowercase letters. For example, either of the following commands starts four additional virtual processors in the AIO class:

```
onmode -p 4 aio
```

```
onmode -p +4 aio
```

The **onmode** utility starts the additional virtual processors immediately.

You can add virtual processors to only one class at a time. To add virtual processors for another class, you must run **onmode** again.

Add network virtual processors

When you add network virtual processors, you add poll threads, each of which requires its own virtual processor to run.

In the following example, the poll threads handle a total of 240 connections:

```
NETTYPE ipcshm,4,60,CPU # Configure poll thread(s) for nettype
```

For ipcshm, the number of poll threads correspond to the number of memory segments. For example, if NETTYPE is set to `3,100` and you want one poll thread, set the poll thread to `1,300`.

Drop CPU and user-defined virtual processors

While the database server is in online mode, you can use the **-p** option of the **onmode** utility to drop, or terminate, virtual processors of the CPU and user-defined classes.

Drop CPU virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the CPU class in lowercase letters. For example, the following command drops two CPU virtual processors:

```
% onmode -p -2 cpu
```

If you attempt to drop a CPU virtual processor that is running a poll thread, you receive the following message:

```
onmode: failed when trying to change the number of cpu virtual processor by -number.
```

For more information, see Run poll threads on CPU or network virtual processors.

Drop user-defined virtual processors

Following the **onmode** command, specify a negative number that is the number of virtual processors that you want to drop, and then specify the user-defined class in lowercase letters. For example, the following command drops two virtual processors of the class *usr*:

```
onmode -p -2 usr
```

Windows only: In Windows, you can have only one user-defined virtual processor class at a time. Omit the *number* parameter in the **onmode -p *vpclass*** command.
For information about how to create a user-defined class of virtual processors and assign user-defined routines to it, see User-defined classes of virtual processors.

Monitor virtual processors

Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity.

For more information about these **onstat -g** options, see the topics on the effect of configuration on CPU utilization in the *IBM® Informix® Performance Guide*.

For examples of output for the **onstat -g** commands, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

- Monitor virtual processors with command-line utilities
  Use **onstat -g** options to monitor virtual processors.
- Monitor virtual processors with SMI tables
  Query the **sysvpprof** table to obtain information about the virtual processors that are currently running.

Monitor virtual processors with command-line utilities

Use **onstat -g** options to monitor virtual processors.

The following options can be used to monitor virtual processors:

- The onstat -g ath command
- The onstat -g glo command
  Use the **onstat -g glo** command to display information about each virtual processor that is currently running, and cumulative statistics for each virtual processor class.
- The onstat -g ioq command
  Use the **onstat -g ioq** option to determine whether you must allocate additional virtual processors. The command **onstat -g ioq** displays the length and other statistics about I/O queues.
- The onstat -g rea command
  Use the **onstat -g rea** option to monitor the number of threads in the ready queue.

The onstat -g ath command

The **onstat -g ath** command displays information about system threads and the virtual-processor classes.

The onstat -g glo command

Use the **onstat -g glo** command to display information about each virtual processor that is currently running, and cumulative statistics for each virtual processor class.

For an example of **onstat -g glo** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

The onstat -g ioq command

Use the **onstat -g ioq** option to determine whether you must allocate additional virtual processors. The command **onstat -g ioq** displays the length and other statistics about I/O queues.

If the length of the I/O queue is growing, I/O requests are accumulating faster than the AIO virtual processors can process them. If the length of the I/O queue continues to show that I/O requests are accumulating, consider adding AIO virtual processors.

For an example of **onstat -g ioq** output, see information in the *IBM® Informix® Administrator's Reference*.

The onstat -g rea command

Use the **onstat -g rea** option to monitor the number of threads in the ready queue.

If the number of threads in the ready queue is growing for a class of virtual processors (for example, the CPU class), you might be required to add more virtual processors to your configuration.

For an example of **onstat -g rea** output, see information in the *IBM® Informix® Administrator's Reference*.

Monitor virtual processors with SMI tables

Query the **sysvpprof** table to obtain information about the virtual processors that are currently running.

This table contains the following columns.

Column	Description
vpid	Virtual-processor ID number
class	Virtual-processor class
usercpu	Minutes of user CPU used
syscpu	Minutes of system CPU used

Shared memory

These topics describe the content of database server shared memory, the factors that determine the sizes of shared-memory areas, and how data moves into and out of shared memory.

For information about how to change the database server configuration parameters that determine shared memory allocations, see Manage shared memory.

- Shared memory
  Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory.
- Shared-memory use
  The database server uses shared memory to enable virtual processors and utilities to share data and to provide a fast communications channel for local client applications that use IPC communication.
- Processes that attach to shared memory
  A number of processes attach to the database server shared memory.
- Resident portion of shared memory
  The operating system, as it switches between the processes that run on the system, normally swaps the contents of portions of memory to disk. When a portion of memory is designated as *resident*, however, it is not swapped to disk. Keeping frequently accessed data resident in memory improves performance because it reduces the number of disk I/O operations that would otherwise be required to access that data.
- Buffer pool portion of shared memory
  The buffer pool portion of shared memory contains the buffers that store dbspace pages that are read from disk.
- Virtual portion of shared memory
  The virtual portion of shared memory is expandable by the database server and can be paged out to disk by the operating system.
- Communications portion of shared memory (UNIX)
  The database server allocates memory for the IPC communication portion of shared memory if you configure at least one of your connections as an IPC shared-memory connection. The database server performs this allocation when you set up shared memory.
- Virtual-extension portion of shared memory
  The virtual-extension portion of shared memory contains additional virtual segments and virtual-extension segments.
- Concurrency control
  The database server threads that run on the same virtual processor and on separate virtual processors share access to resources in shared memory.
- Database server thread access to shared buffers
  Database server threads access shared buffers through a system of queues, using mutexes and locks to synchronize access and protect data.
- Flush data to disk
  Writing a buffer to disk is called *buffer flushing*.
- Buffer large-object data
  Simple large objects (TEXT or BYTE data) can be stored in either dbspaces or blobspaces. Smart large objects (CLOB or BLOB data) are stored only in sbspaces.
- Memory use on 64-bit platforms
  With 64-bit addressing, you can have larger buffer pools to reduce the amount of I/O operations to obtain data from disks.

Related reference:
Database server maintenance tasks

Shared memory

Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory.

The database server uses shared memory for the following purposes:

- To reduce memory usage and disk I/O
- To perform high-speed communication between processes

Shared memory enables the database server to reduce overall memory usage because the participating processes, in this case, virtual processors, do not require maintaining private copies of the data that is in shared memory.

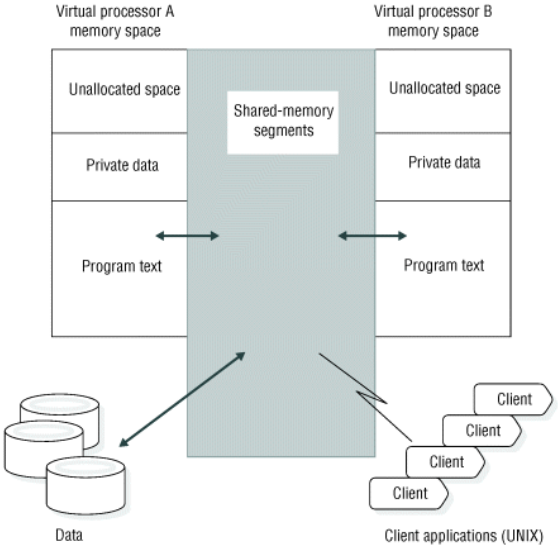
Shared memory reduces disk I/O, because buffers, which are managed as a common pool, are flushed on a database server-wide basis instead of a per-process basis. Furthermore, a virtual processor can often avoid reading data from disk because the data is already in shared memory as a result of an earlier read operation. The reduction in disk I/O reduces execution time.

Shared memory provides the fastest method of interprocess communication, because it processes read and write messages at the speed of memory transfers.

Shared-memory use

The database server uses shared memory to enable virtual processors and utilities to share data and to provide a fast communications channel for local client applications that use IPC communication.

The following figure illustrates the shared-memory scheme.
Figure 1. How the database server uses shared memory



- Shared-memory allocation
  The database server creates portions in shared memory to handle different processes.
- Shared-memory size
  Each portion of the database server shared memory consists of one or more operating-system segments of memory, each one divided into a series of blocks that are 4 KB in size and managed by a bitmap.
- Action to take if SHMTOTAL is exceeded
  When the database server requires more memory than SHMTOTAL allows, a transient condition occurs, perhaps caused by a burst of activity that exceeds the normal processing load.

Shared-memory allocation

The database server creates portions in shared memory to handle different processes.

The database server creates the following portions of shared memory:

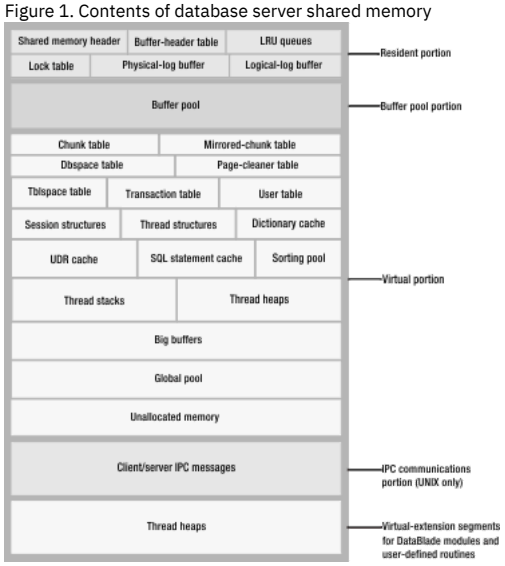
- The *resident* portion
- The *buffer pool* portion
- The *virtual* portion
- The IPC *communications* or message portion
  If the sqlhosts file specifies shared-memory communications, the database server allocates memory for the communications portion.
- The *virtual-extension* portion

The database server adds operating-system segments, as required, to the virtual and virtual-extension portions of shared memory.

For more information about shared-memory settings for your platform, see the machine notes. The following figure shows the contents of each portion of shared memory.

All database server virtual processors have access to the same shared-memory segments. Each virtual processor manages its work by maintaining its own set of pointers to shared-memory resources such as buffers, locks, and latches. Virtual processors attach to shared memory when you take the database server from offline mode to quiescent, administration, or online. The database server uses locks and latches to manage concurrent access to shared-memory resources by multiple threads.

Figure 1. Contents of database server shared memory



Shared-memory size

Each portion of the database server shared memory consists of one or more operating-system segments of memory, each one divided into a series of blocks that are 4 KB in size and managed by a bitmap.

The header-line output by the **onstat** utility contains the size of the database server shared memory, expressed in KB. You can also use **onstat -g seg** to monitor how much memory the database server allocates for each portion of shared memory. For information about how to use **onstat**, see the *IBM® Informix® Administrator's Reference*.

You can set the SHMTOTAL parameter in the onconfig file to limit the amount of memory overhead that the database server can place on your computer or node. The SHMTOTAL parameter specifies the total amount of shared memory that the database server can use for all memory allocations. However, certain operations might fail if the database server requires more memory than the amount set in SHMTOTAL. If this condition occurs, the database server displays the following message in the message log:

```
size of resident + virtual segments x + y > z
   total allowed by configuration parameter SHMTOTAL
```

In addition, the database server returns an error message to the application that initiated the offending operation. For example, if the database server requires more memory than you specify in SHMTOTAL while it tries to perform an operation such as an index build or a hash join, it returns an error message to the application that is similar to one of the following messages:

```
-567    Cannot write sorted rows.
-116    ISAM error: cannot allocate memory.
```

After the database server sends these messages, it rolls back any partial results performed by the offending query.

Internal operations, such as page-cleaner or checkpoint activity, can also cause the database server to exceed the SHMTOTAL ceiling. When this situation occurs, the database server sends a message to the message log. For example, suppose that the database server attempts and fails to allocate additional memory for page-cleaner activity. As a consequence, the database server sends information to the message log that is similar to the following messages:

```
17:19:13          Assert Failed: WARNING! No memory available for page cleaners
17:19:13          Who: Thread(11, flush_sub(0), 9a8444, 1)
17:19:13          Results: Database server may be unable to complete a checkpoint
17:19:13          Action: Make more virtual memory available to database server
17:19:13          See Also: /tmp/af.c4
```

After the database server informs you about the failure to allocate additional memory, it rolls back the transactions that caused it to exceed the SHMTOTAL limit. Immediately after the rollback, operations no longer fail from lack of memory, and the database server continues to process transactions as usual.

Action to take if SHMTOTAL is exceeded

When the database server requires more memory than SHMTOTAL allows, a transient condition occurs, perhaps caused by a burst of activity that exceeds the normal processing load.

Only the operation that caused the database server to run out of memory temporarily fails. Other operations continue to be processed in a normal fashion.

If messages indicate on a regular basis that the database server requires more memory than SHMTOTAL allows, you have not configured the database server correctly. Lowering DS_TOTAL_MEMORY or the **buffers** value in the BUFFERPOOL configuration parameter is one possible solution; increasing the value of SHMTOTAL is another.

Processes that attach to shared memory

A number of processes attach to the database server shared memory.

The following processes attach to the database server shared memory:

- Client-application processes that communicate with the database server through the shared-memory communications portion (**ipcshm**)
- Database server virtual processors
- Database server utilities

The following topics describe how each type of process attaches to the database server shared memory.

- How a client attaches to the communications portion (UNIX)
  Client-application processes that communicate with the database server through shared memory (nettype `ipcshm`) attach transparently to the communications portion of shared memory. System-library functions that are automatically compiled into the application enable it to attach to the communications portion of shared memory.
- How utilities attach to shared memory
  Database server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory.
- How virtual processors attach to shared memory
  The database server virtual processors attach to shared memory during setup.

How a client attaches to the communications portion (UNIX)

Client-application processes that communicate with the database server through shared memory (nettype `ipcshm`) attach transparently to the communications portion of shared memory. System-library functions that are automatically compiled into the application enable it to attach to the communications portion of shared memory.

For information about specifying a shared-memory connection, see Client/server communication, and Network virtual processors.

If the INFORMIXSHMBASE environment variable is not set, the client application attaches to the communications portion at an address that is platform-specific. If the client application attaches to other shared-memory segments (not database server shared memory), the user can set the INFORMIXSHMBASE environment variable to specify the address at which to attach the database server shared-memory communications segments. When you specify the address at which to address the shared-memory communications segments, you can prevent the database server from colliding with the other shared-memory segments that your application uses. For information about how to set the INFORMIXSHMBASE environment variable, see the *IBM® Informix® Guide to SQL: Reference*.

Related reference:
Shared-memory connections (UNIX)
Environment variables for network connections

How utilities attach to shared memory

Database server utilities such as **onstat**, **onmode**, and **ontape** attach to shared memory.

The **onstat**, **onmode**, and **ontape** utilities attach to shared memory through one of the following files.

Operating system	File
UNIX	$INFORMIXDIR/etc/.infos.*servername*
Windows	%INFORMIXDIR%\etc\.infos.*servername*

The variable **servername** is the value of the DBSERVERNAME parameter in the onconfig file. The utilities obtain the **servername** portion of the file name from the INFORMIXSERVER environment variable.

The **oninit** process reads the onconfig file and creates the file .infos.servername when it starts the database server. The file is removed when the database server terminates.

How virtual processors attach to shared memory

The database server virtual processors attach to shared memory during setup.

During this process, the database server must satisfy the following two requirements:

- Ensure that all virtual processors can locate and access the same shared-memory segments
- Ensure that the shared-memory segments are located in physical memory locations that are different than the shared-memory segments assigned to other instances of the database server, if any, on the same computer

The database server uses two configuration parameters, SERVERNUM and SHMBASE, to meet these requirements.

When a virtual processor attaches to shared memory, it performs the following major steps:

- Accesses the SERVERNUM parameter from the onconfig file
- Uses SERVERNUM to calculate a shared-memory key value
- Requests a shared-memory segment using the shared-memory key value
  The operating system returns the shared-memory identifier for the first shared-memory segment.
- Directs the operating system to attach the first shared-memory segment to its process space at SHMBASE
- Attaches additional shared-memory segments, if required, to be contiguous with the first segment

The following topics describe how the database server uses the values of the SERVERNUM and SHMBASE configuration parameters in the process of attaching shared-memory segments.

- Obtain key values for shared-memory segments
  The values of the SERVERNUM configuration parameter and *shmkey*, an internally calculated number, determine the unique key value for each shared-memory segment.
- Specify where to attach the first shared-memory segment
  The SHMBASE parameter in the onconfig file specifies the virtual address where each virtual processor attaches the first, or base, shared-memory segment.
- Attach additional shared-memory segments
  To attach additional shared-memory segments, a virtual processor requests them from the operating system in much the same way that it requested the first segment.
- Define the shared-memory lower-boundary address
  If your operating system uses a parameter to define the lower boundary address for shared memory, and the parameter is set incorrectly, it can prevent the shared-memory segments from being attached contiguously.

Obtain key values for shared-memory segments

The values of the SERVERNUM configuration parameter and *shmkey*, an internally calculated number, determine the unique key value for each shared-memory segment.

To see the key values for shared-memory segments, run the **onstat -g seg** command. For more information, see the sections on SHMADD and the buffer pool in your *IBM® Informix® Performance Guide*.

When a virtual processor requests that the operating system attach the first shared-memory segment, it supplies the unique key value to identify the segment. In return, the operating system passes back a shared-memory segment identifier associated with the key value. Using this identifier, the virtual processor requests that the operating system attach the segment of shared memory to the virtual-processor address space.

Specify where to attach the first shared-memory segment

The SHMBASE parameter in the onconfig file specifies the virtual address where each virtual processor attaches the first, or base, shared-memory segment.

Each virtual processor attaches to the first shared-memory segment at the same virtual address. This situation enables all virtual processors within the same database server instance to reference the same locations in shared memory without calculating shared-memory addresses. All shared-memory addresses for an instance of the database server are relative to SHMBASE.
Warning: Do not change the value of SHMBASE.
The value of SHMBASE is sensitive for the following reasons:

- The specific value of SHMBASE depends on the platform and whether the processor is a 32-bit or 64-bit processor. The value of SHMBASE is not an arbitrary number and is intended to keep the shared-memory segments safe when the virtual processor dynamically acquires additional memory space.
- Different operating systems accommodate additional memory at different virtual addresses. Some architectures extend the highest virtual address of the virtual-processor data segment to accommodate the next segment. In this case, the data segment might grow into the shared-memory segment.
- Some versions of UNIX require the user to specify an SHMBASE parameter of virtual address zero. The zero address informs the UNIX kernel that the kernel picks the best address at which to attach the shared-memory segments. However, not all UNIX architectures support this option. Moreover, on some systems, the selection that the kernel makes might not be the best selection.

For information about SHMBASE, see your IBM® Informix® machine notes.

Attach additional shared-memory segments

To attach additional shared-memory segments, a virtual processor requests them from the operating system in much the same way that it requested the first segment.

Each virtual processor must attach to the total amount of shared memory that the database server has acquired. After a virtual processor attaches each shared-memory segment, it calculates how much shared memory it has attached and how much remains. The database server facilitates this process by writing a shared-memory header to the first shared-memory segment. Sixteen bytes into the header, a virtual processor can obtain the following data:

- The total size of shared memory for this database server
- The size of each shared-memory segment

For the additional segments, however, the virtual processor adds 1 to the previous value of *shmkey*. The virtual processor directs the operating system to attach the segment at the address that results from the following calculation:

SHMBASE + (*seg_size* x number of attached segments)

The virtual processor repeats this process until it has acquired the total amount of shared memory.

Given the initial key value of (SERVERNUM * 65536) + *shmkey*, the database server can request up to 65,536 shared-memory segments before it can request a shared-memory key value used by another database server instance on the same computer.

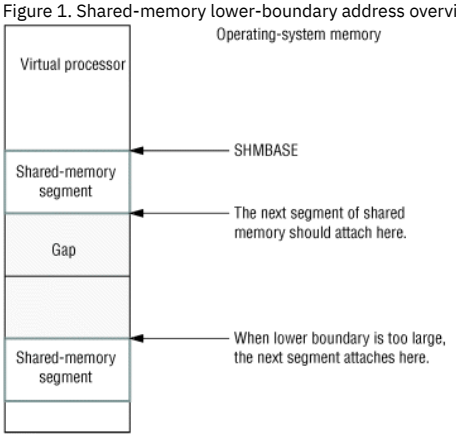
Define the shared-memory lower-boundary address

If your operating system uses a parameter to define the lower boundary address for shared memory, and the parameter is set incorrectly, it can prevent the shared-memory segments from being attached contiguously.

The following figure illustrates the problem. If the lower-boundary address is less than the ending address of the previous segment plus the size of the current segment, the operating system attaches the current segment at a point beyond the end of the previous segment. This action creates a gap between the two segments. Because shared memory must be attached to a virtual processor so that it looks like contiguous memory, this gap creates problems. The database server receives errors when this situation occurs.

To correct the problem, check the operating-system kernel parameter that specifies the lower-boundary address or reconfigure the kernel to allow larger shared-memory segments.

Figure 1. Shared-memory lower-boundary address overview



Resident portion of shared memory

The operating system, as it switches between the processes that run on the system, normally swaps the contents of portions of memory to disk. When a portion of memory is designated as *resident*, however, it is not swapped to disk. Keeping frequently accessed data resident in memory improves performance because it reduces the number of disk I/O operations that would otherwise be required to access that data.

The database server requests that the operating system keep the virtual portions in physical memory when the following two conditions exist:

- The operating system supports shared-memory residency.
- The RESIDENT parameter in the onconfig file is set to -1 or a value that is greater than 0.

Warning: You must consider the use of shared memory by all applications when you consider whether to set the RESIDENT parameter to -1. Locking all shared memory for the use of the IBM® Informix® database server can adversely affect the performance of other applications, if any, on the same computer.
The resident portion of the database server shared memory stores the following data structures that do not change in size while the database server is running:

- Shared-memory header
- Logical-log buffer
- Physical-log buffer
- Lock table

- Shared-memory header
  The shared-memory header contains a description of all other structures in shared memory, including internal tables and the buffer pool, and pointers to the locations of these structures.
- Logical-log buffer
  The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server.
- Physical-log buffer
  The database server uses the physical-log buffer to hold before-images of some of the modified dbspace pages.
- High-Availability Data-Replication buffer
  Data replication requires two instances of the database server, a primary instance and a secondary instance, running on two computers.
- Lock table
  A lock is created when a user thread writes an entry in the lock table. A single transaction can own multiple locks. The lock table is the pool of available locks.

Related information:
RESIDENT configuration parameter

Shared-memory header

The shared-memory header contains a description of all other structures in shared memory, including internal tables and the buffer pool, and pointers to the locations of these structures.

When a virtual processor first attaches to shared memory, it reads address information in the shared-memory header for directions to all other structures.

The size of the shared-memory header is about 200 KB, but the size varies depending on the computer platform. You cannot tune the size of the header.

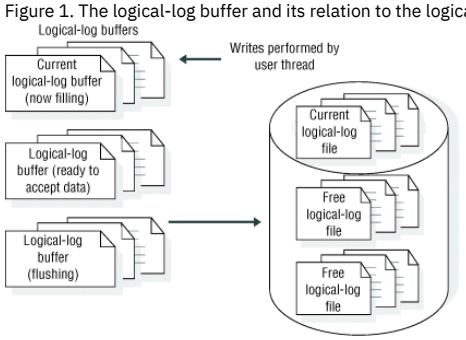
Logical-log buffer

The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server.

The logical log contains the following five types of log records, in addition to many others:

- SQL data definition statements for all databases
- SQL data manipulation statements for databases that were created with logging
- Record of a change to the logging status of a database
- Record of a checkpoint
- Record of a change to the configuration

The database server uses only one of the logical-log buffers at a time. This buffer is the current logical-log buffer. Before the database server flushes the current logical-log buffer to disk, it makes the second logical-log buffer the current one so that it can continue writing while the first buffer is flushed. If the second logical-log buffer fills before the first one finishes flushing, the third logical-log buffer becomes the current one. This process is illustrated in the following figure.
Figure 1. The logical-log buffer and its relation to the logical-log files on disk



For a description of how the database server flushes the logical-log buffer, see Flush the logical-log buffer.

The LOGBUFF configuration parameter specifies the size of the logical-log buffers. Small buffers can create problems if you store records larger than the size of the buffers (for example, TEXT or BYTE data in dbspaces). The recommended value for the size of a logical log buffer is 64 KB. Whenever the setting is less than the recommended value, the database server suggests a value during server startup. For the possible values that you can assign to this configuration parameter, see the *IBM® Informix® Administrator's Reference*.

For information about the affect of TEXT and BYTE data on shared memory buffers, see Buffer large-object data.

Physical-log buffer

The database server uses the physical-log buffer to hold before-images of some of the modified dbspace pages.

The before-images in the physical log and the logical-log records enable the database server to restore consistency to its databases after a system failure.

The physical-log buffer is actually two buffers. Double buffering permits the database server processes to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. For a description of how the database server flushes the physical-log buffer, see Flush the physical-log buffer. For information about monitoring the physical-log file, see Monitor physical and logical-logging activity.

The PHYSBUFF parameter in the onconfig file specifies the size of the physical-log buffers. A write to the physical-log buffer writes exactly one page. If the specified size of the physical-log buffer is not evenly divisible by the page size, the database server rounds the size down to the nearest value that is evenly divisible by the page size. Although some operations require the buffer to be flushed sooner, in general the database server flushes the buffer to the physical-log file on disk when the buffer fills. Thus, the size of the buffer determines how frequently the database server must flush it to disk.

The default value for the physical log buffer size is 512 KB. If you decide to use a smaller value, the database server displays a message indicating that optimal performance might not be attained. Using a physical log buffer smaller than 512 KB affects performance only, not transaction integrity.

For more information about this configuration parameter, see the *IBM® Informix® Administrator's Reference*.

High-Availability Data-Replication buffer

Data replication requires two instances of the database server, a primary instance and a secondary instance, running on two computers.

If you implement data replication for your database server, the primary database server holds logical-log records in the data replication buffers before it sends them to the secondary database server. A data replication buffer is always the same size as the logical-log buffer. For information about the size of the logical-log buffer, see the preceding topic, Logical-log buffer. For more information about how the data replication buffer is used, see How data replication works.

Lock table

A lock is created when a user thread writes an entry in the lock table. A single transaction can own multiple locks. The lock table is the pool of available locks.

For an explanation of locking and the SQL statements associated with locking, see the *IBM® Informix® Guide to SQL: Tutorial*.

The following information, which is stored in the lock table, describes the lock:

- The address of the transaction that owns the lock
- The type of lock (exclusive, update, shared, byte, or intent)
- The page or rowid that is locked
- The table space where the lock is placed
- Information about the bytes locked (byte-range locks for smart large objects):
  - Smart-large-object ID
  - Offset into the smart large object where the locked bytes begin
  - The number of bytes locked, starting at the offset

To specify the initial size of the lock table, set the LOCKS configuration parameter. For information about using the LOCKS configuration parameter to specify the number of locks for a session, see the topics about configuration parameters in the *IBM Informix Administrator's Reference* and the topics about configuration effects on memory utilization in your *IBM Informix Performance Guide*.

If the number of locks allocated by sessions exceeds the value specified in the LOCKS configuration parameter, the database server doubles the size of the lock table, up to 15 times. The database server increases the size of the lock table by attempting to double the lock table on each increase. However, the amount added during each increase is limited to a maximum value. For 32-bit platforms, a maximum of 100,000 locks can be added during each increase. Therefore, the total maximum locks allowed for 32-bit platforms is 8,000,000 (maximum number of starting locks) + 99 (maximum number of dynamic lock table extensions) x 100,000 (maximum number of locks added per lock table extension). For 64-bit platforms, a maximum of 1,000,000 locks can be added during each increase. Therefore, the total maximum locks allowed is 500,000,000 (maximum number of starting locks) + 99 (maximum number of dynamic lock table extensions) x 1,000,000 (maximum number of locks added per lock table extension).

Use the DEF_TABLE_LOCKMODE configuration parameter to set the lock mode to page or row for new tables.

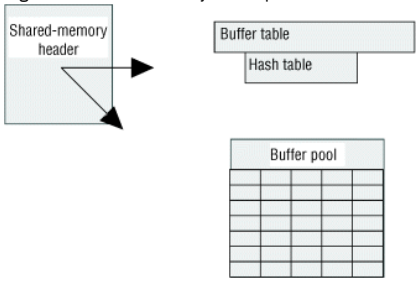
Locks can prevent sessions from reading data until after a concurrent transaction is committed or rolled back. For databases created with transaction logging, you can use the USELASTCOMMITTED configuration parameter in the onconfig file to specify whether the database server uses the last committed version of the data. The last committed version of the data is the version of the data that existed before any updates occurred. The value you set with the USELASTCOMMITTED configuration parameter overrides the isolation level that is specified in the SET ISOLATION TO COMMITTED READ statement of SQL. For more information about using the USELASTCOMMITTED configuration parameter, see the topics about configuration parameters in the *IBM Informix Administrator's Reference*.

For more information about using and monitoring locks, see the topics about locking in your *IBM Informix Performance Guide* and the *IBM Informix Guide to SQL: Tutorial*.

Buffer pool portion of shared memory

The buffer pool portion of shared memory contains the buffers that store dbspace pages that are read from disk.

The following figure illustrates the shared-memory header and the buffer pool.
Figure 1. Shared-memory buffer pool



You use the BUFFERPOOL configuration parameter to specify information about a buffer pool, including the number of buffers in the buffer pool or the overall size of the buffer pool. Each buffer is the size of one database server page. Too few buffers can severely affect performance. You can set the BUFFERPOOL configuration parameter to allow the database server to automatically increase the number of buffers as needed to improve performance. Otherwise, you must monitor the database server and tune the number of buffers to determine an acceptable value.

A buffer pool manages one size of pages. You need a different buffer pool for each page size that is used by storage spaces in the database server. The database server automatically creates the required buffer pools. For example, if you create the first dbspace that has a page size of 6 KB, the database server creates a buffer pool to cache the default number of 6 KB pages in memory. You can control the properties of buffer pools with the BUFFERPOOL configuration parameter.

If the database server is in online, quiescent, or administration mode, you can also use the **onparams -b** command to add a buffer pool of a different size. When you use the **onparams -b** command, the information that you specify is transferred automatically to the onconfig file as a new entry of the BUFFERPOOL configuration parameter.

In general, the database server performs I/O in full-page units, the size of a buffer. The exceptions are I/O performed from big buffers, from blobspace buffers, or from lightweight I/O buffers.

Automatic LRU (least recently used) tuning affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values that can be explicitly set by the BUFFERPOOL configuration parameter.

The status of the buffers is tracked through the buffer table. Within shared memory, buffers are organized into FIFO/LRU buffer queues. Buffer acquisition is managed by mutexes and lock-access information.

The **onstat -b** command shows information about the buffers.

Related concepts:
Thread data
Mutexes
Related reference:
Creation of blobpage buffers
Related information:
BUFFERPOOL configuration parameter
onstat -b command: Print buffer information for buffers in use
The BUFFERPOOL configuration parameter and memory utilization
onparams -b: Add a buffer pool

Virtual portion of shared memory

The virtual portion of shared memory is expandable by the database server and can be paged out to disk by the operating system.

As the database server executes, it automatically attaches additional operating-system segments, as necessary, to the virtual portion.

- Management of the virtual portion of shared memory
  The database server uses memory *pools* to track memory allocations that are similar in type and size.
- Components of the virtual portion of shared memory
  The virtual portion of shared memory stores a variety of data.
- Data-distribution cache
  The database server uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost.

Management of the virtual portion of shared memory

The database server uses memory *pools* to track memory allocations that are similar in type and size.

Keeping related memory allocations in a pool helps to reduce memory fragmentation. It also enables the database server to free a large allocation of memory at one time, as opposed to freeing each piece that makes up the pool.

All sessions have one or more memory pools. When the database server requires memory, it looks first in the specified pool. If insufficient memory is available in a pool to satisfy a request, the database server adds memory from the system pool. If the database server cannot find enough memory in the system pool, it dynamically allocates more segments to the virtual portion.

The database server allocates virtual shared memory for each of its subsystems (session pools, stacks, heaps, control blocks, system catalog, SPL routine caches, SQL statement cache, sort pools, and message buffers) from pools that track free space through a linked list. When the database server allocates a portion of memory, it first searches the pool free-list for a *fragment* of sufficient size. If it finds none, it brings new blocks into the pool from the virtual portion. When memory is freed, it goes back to the pool as a free fragment and remains there until the pool is deleted. When the database server starts a session for a client application, for example, it allocates memory for the session pool. When the session terminates, the database server returns the allocated memory as free fragments.

- Size of the virtual portion of shared memory
  Use configuration parameters to specify the initial size of the virtual portion of shared memory, the size of segments to be added later, and the amount of memory available for PDQ queries.

Size of the virtual portion of shared memory

Use configuration parameters to specify the initial size of the virtual portion of shared memory, the size of segments to be added later, and the amount of memory available for PDQ queries.

To specify the initial size of the virtual shared-memory portion, set the SHMVIRTSIZE configuration parameter. To specify the size of segments that are added later to the virtual shared memory, set the SHMADD and the EXTSHMADD configuration parameter.

To specify the amount of memory available for PDQ queries, set the DS_TOTAL_MEMORY parameter.

If you want to increase the amount of memory that is available for a query that is not a PDQ query and the PDQ priority is set to 0 (zero), you can change the amount in any of the following ways:

- Set the DS_NONPDQ_QUERY_MEM configuration parameter
- Run the **onmode -wm** or the **onmode -wf** command

For example, if you use the **onmode** utility, specify a value as shown in the following example:

```
onmode -wf DS_NONPDQ_QUERY_MEM=500
```

The minimum value for DS_NONPDQ_QUERY_MEM is 128 KB. The maximum supported value is 25 percent of the value of DS_TOTAL_MEMORY.
Related reference:
Add a segment to the virtual portion of shared memory
Related information:
DS_TOTAL_MEMORY configuration parameter
DS_NONPDQ_QUERY_MEM configuration parameter
SHMVIRTSIZE configuration parameter
SHMADD configuration parameter
EXTSHMADD configuration parameter

Components of the virtual portion of shared memory

The virtual portion of shared memory stores a variety of data.

The following data is stored in the virtual portion of shared memory:

- Internal tables
- Big buffers
- Session data
- Thread data (stacks and heaps)
- Data-distribution cache
- Dictionary cache
- SPL routine cache
- SQL statement cache
- Sorting pool
- Global pool

- Shared-memory internal tables
  The database server shared memory contains seven internal tables that track shared-memory resources.
- Big buffers
  A big buffer is a single buffer that is made up of several pages. The actual number of pages is platform-dependent.
- Session data
  When a client application requests a connection to the database server, the database server begins a *session* with the client and creates a data structure for the session in shared memory. The created data structure is called the *session-control block*.
- Thread data
  When a client connects to the database server, in addition to starting a session, the database server starts a primary session thread and creates a *thread-control block* for it in shared memory.

Shared-memory internal tables

The database server shared memory contains seven internal tables that track shared-memory resources.

The shared-memory internal tables are as follows:

- Buffer table
- Chunk table
- Dbspace table
- Page-cleaner table
- Tblspace table
- Transaction table
- User table

- Buffer table
  The buffer table tracks the addresses and status of the individual buffers in the shared-memory pool.
- Chunk table
  The chunk table tracks all chunks in the database server.
- Dbspace table
  The dbspace table tracks storage spaces in the database server.
- Page-cleaner table
  The page-cleaner table tracks the state and location of each of the page-cleaner threads.
- Tblspace table
  he tblspace table tracks all active tblspaces in a database server instance.
- Transaction table
  The transaction table tracks all transactions in the database server.
- User table
  The user table tracks all user threads and system threads.

Buffer table

The buffer table tracks the addresses and status of the individual buffers in the shared-memory pool.

When a buffer is used, it contains an image of a data or index page from disk. For more information about the purpose and content of a disk page, see Pages.

Each buffer in the buffer table contains the following control information, which is required for buffer management:

Buffer status
: Buffer status is described as empty, unmodified, or modified. An unmodified buffer contains data, but the data can be overwritten. A modified (dirty) buffer contains data that must be written to disk before it can be overwritten.

Current lock-access level
: Buffers receive lock-access levels depending on the type of operation that the user thread is executing. The database server supports two buffer lock-access levels: shared and exclusive.

Threads waiting for the buffer
: Each buffer header maintains a list of the threads that are waiting for the buffer and the lock-access level that each waiting thread requires.

Each database server buffer has one entry in the buffer table.

For information about the database server buffers, see Resident portion of shared memory. For information about how to monitor the buffers, see Monitor buffers.

The database server determines the number of entries in the buffer-table hash table, based on the number of allocated buffers. The maximum number of hash values is the largest power of 2 that is less than the value of **buffers**, which is specified in one of the BUFFERPOOL configuration parameter fields.

Chunk table

The chunk table tracks all chunks in the database server.

If mirroring has been enabled, a corresponding mirror chunk table is also created when shared memory is set up. The mirror chunk table tracks all mirror chunks.

The chunk table in shared memory contains information that enables the database server to locate chunks on disk. This information includes the number of the initial chunk and the number of the next chunk in the dbspace. Flags also describe chunk status: mirror or primary; offline, online, or recovery mode; and whether this chunk is part of a blobspace. For information about monitoring chunks, see Monitor chunks.

The maximum number of entries in the chunk table might be limited by the maximum number of file descriptors that your operating system allows per process. You can usually specify the number of file descriptors per process with an operating-system kernel-configuration parameter. For details, consult your operating-system manuals.

Dbspace table

The dbspace table tracks storage spaces in the database server.

The dbspace-table information includes the following information about each dbspace:

- Dbspace number
- Dbspace name and owner
- Dbspace mirror status (mirrored or not)
- Date and time that the dbspace was created

If the storage space is a blobspace, flags indicate the media where the blobspace is located: magnetic or removable. If the storage space is an sbspace, it contains internal tables that track metadata for smart large objects and large contiguous blocks of pages containing user data.

For information about monitoring dbspaces, see Monitor disk usage.

Page-cleaner table

The page-cleaner table tracks the state and location of each of the page-cleaner threads.

The number of page-cleaner threads is specified by the CLEANERS configuration parameter in the onconfig file. For advice on how many page-cleaner threads to specify, see the chapter on configuration parameters in the *IBM® Informix® Administrator's Reference*.

The page-cleaner table always contains 128 entries, regardless of the number of page-cleaner threads specified by the CLEANERS parameter in the onconfig file.

For information about monitoring the activity of page-cleaner threads, see information about the **onstat -F** option in the *IBM Informix Administrator's Reference*.

Tblspace table

he tblspace table tracks all active tblspaces in a database server instance.

TAn active tblspace is one that is currently in use by a database session. Each active table accounts for one entry in the tblspace table. Active tblspaces include database tables, temporary tables, and internal control tables, such as system catalog tables. Each tblspace table entry includes header information about the tblspace, the tblspace name, and pointers to the tblspace **tblspace** in dbspaces on disk. (The shared-memory active tblspace table is different from the tblspace **tblspace**.) For information about monitoring tblspaces, see Monitor tblspaces and extents.

The database server manages one tblspace table for each dbspace.

Transaction table

The transaction table tracks all transactions in the database server.

Tracking information derived from the transaction table is shown in the **onstat -x** display. For an example of the output that **onstat -x** displays, see monitoring transactions in your *IBM® Informix® Performance Guide*.

The database server automatically increases the number of entries in the transaction table, up to a maximum of 32,767, based on the number of current transactions.

For more information about transactions and the SQL statements that you use with transactions, see the *IBM Informix Guide to SQL: Tutorial*, the *IBM Informix Guide to SQL: Reference*, and the *IBM Informix Guide to SQL: Syntax*.

UNIX only: The transaction table also specifically supports the X/Open environment. Support for the X/Open environment requires TP/XA.

User table

The user table tracks all user threads and system threads.

Each client session has one primary thread and zero-to-many secondary threads, depending on the level of parallelism specified. System threads include one to monitor and control checkpoints, one to process **onmode** commands, the B-tree scanner threads, and page-cleaner threads.

The database server increases the number of entries in the user table as necessary. You can monitor user threads with the **onstat -u** command.

Big buffers

A big buffer is a single buffer that is made up of several pages. The actual number of pages is platform-dependent.

The database server allocates big buffers to improve performance on large reads and writes. The database server uses a big buffer whenever it writes to disk multiple pages that are physically contiguous. For example, the database server tries to use a big buffer to perform a series of sequential reads (light scans) or to read into shared memory simple large objects that are stored in a dbspace.

Users do not have control over the big buffers. If the database server uses light scans, it allocates big buffers from shared memory.

For information about monitoring big buffers with the **onstat** command, see the topics about configuration effects on I/O activity in your *IBM® Informix® Performance Guide*.

Session data

When a client application requests a connection to the database server, the database server begins a *session* with the client and creates a data structure for the session in shared memory. The created data structure is called the *session-control block*.

The session-control block stores the session ID, the user ID, the process ID of the client, the name of the host computer, and various status flags.

The database server allocates memory for session data as necessary.

You can impose restrictions on the memory allocated for sessions by setting the SESSION_LIMIT_MEMORY configuration parameter to specify the maximum amount of memory that a session can allocate. The limits do not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Related information:
SESSION_LIMIT_MEMORY configuration parameter
onstat -g ses command: Print session-related information

Thread data

When a client connects to the database server, in addition to starting a session, the database server starts a primary session thread and creates a *thread-control block* for it in shared memory.

The database server also starts internal threads on its own behalf and creates thread-control blocks for them. When the database server switches from running one thread to running another one (a context switch), it saves information about the thread— such as the register contents, program counter (address of the next instruction), and global pointers—in the thread-control block. For more information about the thread-control block and how it is used, see Context switching.

The database server allocates memory for thread-control blocks as necessary.

- Stacks
  Each thread in the database server has its own stack area in the virtual portion of shared memory.
- Heaps
  Each thread has a heap to hold data structures that it creates while it is running.

Related concepts:
Buffer pool portion of shared memory

Stacks

Each thread in the database server has its own stack area in the virtual portion of shared memory.

For a description of how threads use stacks, see Stacks. For information about how to monitor the size of the stack for a session, see monitoring sessions and threads section in your *IBM® Informix® Performance Guide*.

The size of the stack space for user threads is specified by the STACKSIZE parameter in the onconfig file. You can change the size of the stack for all user threads, if necessary, by changing the value of STACKSIZE.

You can use the INFORMIXSTACKSIZE environment variable to override the STACKSIZE value in the server configuration file. Set INFORMIXSTACKSIZE in the environment and recycle the instance.

Related information:
STACKSIZE configuration parameter

Heaps

Each thread has a heap to hold data structures that it creates while it is running.

A heap is dynamically allocated when the thread is created. The size of the thread heap is not configurable.

Data-distribution cache

The database server uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost.

When the database server accesses the distribution statistics for a specific column the first time, it reads the distribution statistics from the **sysdistrib** system catalog table on disk and stores the statistics in the data-distribution cache. These statistics can then be read for the optimization of subsequent queries that access the column.

Performance improves if these statistics are efficiently stored and accessed from the data-distribution cache. You can configure the size of the data-distribution cache with the DS_HASHSIZE and DS_POOLSIZE configuration parameters. For information about changing the default size of the data-distribution cache, see the topics about queries and the query optimizer in your *IBM® Informix® Performance Guide*.

- Dictionary cache
  When a session executes an SQL statement that requires access to a system catalog table, the database server reads data from the system catalog tables.
- SQL statement cache
  The SQL statement cache reduces memory usage and preparation time for queries.
- Sort memory
  The amount of virtual shared memory that the database server allocates for a sort depends on the number of rows to be sorted and the size of the row, along with other factors.
- SPL routine and the UDR cache
  The database server converts an SPL routine to executable format and stores the routine in the UDR cache, where it can be accessed by any session.
- Global pool
  The global pool stores structures that are global to the database server.

Dictionary cache

When a session executes an SQL statement that requires access to a system catalog table, the database server reads data from the system catalog tables.

The database server stores the catalog data for each queried table in structures that it can access more efficiently during subsequent queries on that table. These structures are created in the virtual portion of shared memory for use by all sessions. These structures constitute the dictionary cache.

You can configure the size of the dictionary cache with the DD_HASHSIZE and DD_HASHMAX configuration parameters. For more information about these parameters, see the chapter on configuration effects on memory in your *IBM® Informix® Performance Guide*.

SQL statement cache

The SQL statement cache reduces memory usage and preparation time for queries.

The database server uses the SQL statement cache to store optimized SQL statements that a user runs. When users run a statement that is stored in the SQL statement cache, the database server does not optimize the statement again, so performance improves.

For more information, see Set SQL statement cache parameters. For details on how these parameters affect the performance of the SQL statement cache, see the *IBM® Informix® Performance Guide*.

Sort memory

The amount of virtual shared memory that the database server allocates for a sort depends on the number of rows to be sorted and the size of the row, along with other factors.

The following database operations can use large amounts of the virtual portion of shared memory to sort data:

- Decision-support queries that involve joins, groups, aggregates and sort operations
- Index builds
- UPDATE STATISTICS statement in SQL

For information about parallel sorts, see your *IBM® Informix® Performance Guide*.

SPL routine and the UDR cache

The database server converts an SPL routine to executable format and stores the routine in the UDR cache, where it can be accessed by any session.

When a session is required to access an SPL routine or other user-defined routine for the first time, the database server reads the definition from the system catalog tables and stores the definition in the UDR cache.

You can configure the size of the UDR cache with the PC_HASHSIZE and PC_POOLSIZE configuration parameters. For information about changing the default size of the UDR cache, see the chapter on queries and the query optimizer in your *IBM® Informix® Performance Guide*.

Global pool

The global pool stores structures that are global to the database server.

The global pool contains the message queues where poll threads for network communications deposit messages from clients. The **sqlexec** threads pick up the messages from the global pool and process them.

For more information, see the sections on network buffer pools and virtual portion of shared memory in your *IBM® Informix® Performance Guide*.

Communications portion of shared memory (UNIX)

The database server allocates memory for the IPC communication portion of shared memory if you configure at least one of your connections as an IPC shared-memory connection. The database server performs this allocation when you set up shared memory.

The communications portion contains the message buffers for local client applications that use shared memory to communicate with the database server.

The size of the communications portion of shared memory equals approximately 12 KB multiplied by the expected number of connections required for shared-memory communications (**nettype** `ipcshm`). If **nettype** `ipcshm` is not present, the expected number of connections defaults to `50`. For information about how a client attaches to the communications portion of shared memory, see How a client attaches to the communications portion (UNIX).

Related reference:
Shared-memory connections (UNIX)

Virtual-extension portion of shared memory

The virtual-extension portion of shared memory contains additional virtual segments and virtual-extension segments.

Virtual-extension segments contain thread heaps for DataBlade modules and user-defined routines that run in user-defined virtual processors.

The EXTSHMADD configuration parameter sets the size of virtual-extension segments. The SHMADD and SHMTOTAL configuration parameters apply to the virtual-extension portion of shared memory, just as they do to the other portions of shared memory.

Concurrency control

The database server threads that run on the same virtual processor and on separate virtual processors share access to resources in shared memory.

When a thread writes to shared memory, it uses mechanisms called *mutexes* and *locks* to prevent other threads from simultaneously writing to the same area. A mutex gives a thread the right to access a shared-memory resource. A lock prevents other threads from writing to a buffer until the thread that placed the lock is finished with the buffer and releases the lock.

- Shared-memory mutexes
  The database server uses *mutexes* to coordinate threads as they attempt to modify data in shared memory. Every modifiable shared-memory resource is associated with a mutex.
- Shared-memory buffer locks
  A primary benefit of shared memory is the ability of database server threads to share access to disk pages stored in the shared-memory buffer pool. The database server maintains thread isolation while it achieves this increased concurrency through a strategy for locking the data buffers.

Shared-memory mutexes

The database server uses *mutexes* to coordinate threads as they attempt to modify data in shared memory. Every modifiable shared-memory resource is associated with a mutex.

Before a thread can modify a shared-memory resource, it must first acquire the mutex associated with that resource. After the thread acquires the mutex, it can modify the resource. When the modification is complete, the thread releases the mutex.

If a thread tries to obtain a mutex and finds that it is held by another thread, the incoming thread must wait for the mutex to be released.

For example, two threads can attempt to access the same slot in the chunk table, but only one can acquire the mutex associated with the chunk table. Only the thread that holds the mutex can write its entry in the chunk table. The second thread must wait for the mutex to be released and then acquire it.

For information about monitoring mutexes (which are also called latches), see Monitor the shared-memory profile and latches.

Shared-memory buffer locks

A primary benefit of shared memory is the ability of database server threads to share access to disk pages stored in the shared-memory buffer pool. The database server maintains thread isolation while it achieves this increased concurrency through a strategy for locking the data buffers.

- Types of buffer locks
  The database server uses two types of locks to manage access to shared-memory buffers.

Types of buffer locks

The database server uses two types of locks to manage access to shared-memory buffers.

The two types of locks are:

- Share locks
- Exclusive locks

Each of these lock types enforces the required level of thread isolation during execution.

- Share lock
  A buffer is in share mode, or has a share lock, if multiple threads have access to the buffer to read the data but none intends to modify the data.
- Exclusive lock
  A buffer is in exclusive mode, or has an exclusive lock, if a thread demands exclusive access to the buffer.

Share lock

A buffer is in share mode, or has a share lock, if multiple threads have access to the buffer to read the data but none intends to modify the data.

Exclusive lock

A buffer is in exclusive mode, or has an exclusive lock, if a thread demands exclusive access to the buffer.

All other thread requests that access the buffer are placed in the wait queue. When the executing thread is ready to release the exclusive lock, it wakes the next thread in the wait queue.

Database server thread access to shared buffers

Database server threads access shared buffers through a system of queues, using mutexes and locks to synchronize access and protect data.

- FIFO/LRU queues
  A buffer holds data for the purpose of caching. The database server uses the least-recently used (LRU) queues to replace the cached data. IBM® Informix® also has a first-in first-out (FIFO) queue. When you set the number of LRU queues, you are actually setting the number of FIFO/LRU queues.
- Read-ahead operations
  The database server automatically reads several pages ahead of the current pages that are being processed for a query, unless you disable automatic read ahead operations. Reading ahead enables applications to run faster because they spend less time waiting for disk I/O.
- Database server thread access to buffer pages
  The database server uses shared-lock buffering to allow more than one database server thread to access the same buffer concurrently in shared memory.

FIFO/LRU queues

A buffer holds data for the purpose of caching. The database server uses the least-recently used (LRU) queues to replace the cached data. IBM® Informix® also has a first-in first-out (FIFO) queue. When you set the number of LRU queues, you are actually setting the number of FIFO/LRU queues.

Use the BUFFERPOOL configuration parameter to specify information about the buffer pool, including information about the number of LRU queues to create when database server shared memory is set up and values for **lru_min_dirty** and **lru_max_dirty**, which control how frequently the shared-memory buffers are flushed to disk.

To improve transaction throughput, increase the **lru_min_dirty** and **lru_max_dirty** values. However, do not change the gap between the **lru_min_dirty** and **lru_max_dirty** values. If the AUTO_LRU_TUNING configuration parameter is enabled, the values of the **lru_max_dirty** and **lru_min_dirty** fields are reset automatically as needed to improve performance.

- Components of LRU queue
  Each LRU queue is composed of a pair of linked lists.
- Pages in least-recently used order
  When the database server processes a request to read a page from disk, it must decide which page to replace in memory.
- LRU queues and buffer-pool management
  Before processing begins, all page buffers are empty, and every buffer is represented by an entry in one of the FLRU queues.
- Number of LRU queues to configure
  Multiple LRU queues reduce user-thread contention and allow multiple cleaners to flush pages from the queues so that the percentage of dirty pages is maintained at an acceptable level.
- Number of cleaners to allocate
  You must configure one cleaner for each disk that your applications update frequently. However, you must also consider the length of your LRU queues and frequency of checkpoints.
- Number of pages added to the MLRU queues
  The page-cleaner threads flush the modified buffers in an MLRU queue to disk. To specify the point at which cleaning begins, use the BUFFERPOOL configuration parameter to specify a value for **lru_max_dirty**, which limits the number of page buffers that can be appended to an MLRU queue.
- End of MLRU cleaning
  You can also specify the point at which MLRU cleaning can end.

Related information:
AUTO_LRU_TUNING configuration parameter
BUFFERPOOL configuration parameter

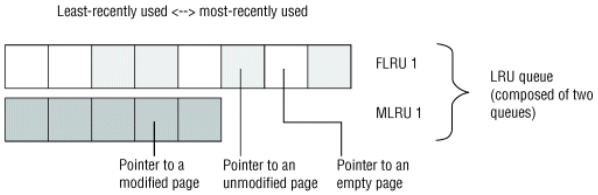
Components of LRU queue

Each LRU queue is composed of a pair of linked lists.

The linked list pairs are as follows:

- FLRU (free least-recently used) list, which tracks free or unmodified pages in the queue
- MLRU (modified least-recently used) list, which tracks modified pages in the queue

The free or unmodified page list is called the FLRU queue of the queue pair, and the modified page list is called the MLRU queue. The two separate lists eliminate the task of searching a queue for a free or unmodified page. The following figure illustrates the structure of the LRU queues.
Figure 1. LRU queue



Pages in least-recently used order

When the database server processes a request to read a page from disk, it must decide which page to replace in memory.

Rather than select a page randomly, the database server assumes that recently referenced pages are more likely to be referenced in the future than pages that it has not referenced for some time. Thus, rather than replacing a recently accessed page, the database server replaces a least-recently accessed page. By maintaining pages in least-recently to most-recently used order, the database server can easily locate the least-recently used pages in memory.

LRU queues and buffer-pool management

Before processing begins, all page buffers are empty, and every buffer is represented by an entry in one of the FLRU queues.

The buffers are evenly distributed among the FLRU queues. To calculate the number of buffers in each queue, divide the total number of buffers by the number of LRU queues. The number of buffers and LRU queues are specified in the BUFFERPOOL configuration parameter.

When a user thread is required to acquire a buffer, the database server randomly selects one of the FLRU queues and uses the oldest or least-recently used entry in the list. If the least-recently used page can be latched, that page is removed from the queue.

If the FLRU queue is locked, and the end page cannot be latched, the database server randomly selects another FLRU queue.

If a user thread is searching for a specific page in shared memory, it obtains the LRU-queue location of the page from the control information stored in the buffer table.

After an executing thread finishes its work, it releases the buffer. If the page has been modified, the buffer is placed at the most-recently used end of an MLRU queue. If the page was read but not modified, the buffer is returned to the FLRU queue at its most-recently used end. For information about how to monitor LRU queues, see Monitor buffers.

Number of LRU queues to configure

Multiple LRU queues reduce user-thread contention and allow multiple cleaners to flush pages from the queues so that the percentage of dirty pages is maintained at an acceptable level.

You specify the number of LRU queues by setting the **lrus** value in the BUFFERPOOL configuration parameter. The default number of LRU queues depends on the number of CPUs on your computer:

- If you have a uniprocessor computer, the default value of the **lrus** field is 8.
- If you have a multiprocessor computer and the MULTIPROCESSOR configuration parameter is enabled, the default value of the **lrus** field is the greater of 8 or the number of CPU virtual processors.

After you provide an initial value for the **lrus** field in the BUFFERPOOL configuration parameter, monitor your LRU queues with the **onstat -R** command. If you find that the percentage of dirty LRU queues consistently exceeds the value of the **lru_max_dirty** field in the BUFFERPOOL configuration parameter, increase the value of the **lrus** field to add more LRU queues.

For example, if the value of the **lru_max_dirty** field is `70` and your LRU queues are consistently 75 percent dirty, you can increase the value of the **lrus** field. If you increase the number of LRU queues, you shorten the length of the queues, which reduces the work of the page cleaners. However, you must allocate enough page cleaners with the CLEANERS configuration parameter.

Related information:
LRU tuning
BUFFERPOOL configuration parameter

Number of cleaners to allocate

You must configure one cleaner for each disk that your applications update frequently. However, you must also consider the length of your LRU queues and frequency of checkpoints.

Another factor that influences whether page cleaners keep up with the number of pages that require cleaning is whether you have enough page-cleaner threads allocated. The percent of dirty pages might exceed the BUFFERPOOL value specified for **lru_max_dirty** in some queues because no page cleaners are available to clean the queues. After a while, the page cleaners might be too far behind to catch up, and the buffer pool becomes dirtier than the percent that you specified in **lru_max_dirty**.

For example, suppose that the CLEANERS parameter is set to `8`, and you increase the number of LRU queues from 8 to 12. You can expect little in the way of a performance gain because the 8 cleaners must now share the work of cleaning an additional 4 queues. If you increase the number of CLEANERS to 12, each of the now-shortened queues can be more efficiently cleaned by a single cleaner.

Setting CLEANERS too low can cause performance to suffer whenever a checkpoint occurs because page cleaners must flush all modified pages to disk during checkpoints. If you do not configure a sufficient number of page cleaners, checkpoints take longer, causing overall performance to suffer.

For more information, see Flush buffer-pool buffers.

Number of pages added to the MLRU queues

The page-cleaner threads flush the modified buffers in an MLRU queue to disk. To specify the point at which cleaning begins, use the BUFFERPOOL configuration parameter to specify a value for **lru_max_dirty**, which limits the number of page buffers that can be appended to an MLRU queue.

The initial setting of **lru_max_dirty** is `60.00`, so page cleaning begins when 60 percent of the buffers managed by a queue are modified.

In practice, page cleaning begins under several conditions, only one of which is when an MLRU queue reaches the value of **lru_max_dirty**. For more information about how the database server performs buffer-pool flushing, see Flush data to disk.

The following example shows how the value of **lru_max_dirty** is applied to an LRU queue to specify when page cleaning begins and thereby limit the number of buffers in an MLRU queue.

```
Buffers specified as 8000
lrus specified as 8
lru_max_dirty specified as 60 percent

Page cleaning begins when the number of buffers in the MLRU
   queue is equal to lru_max_dirty.
```

```
Buffers per lru queue = (8000/8) = 1000

Max buffers in MLRU queue and point at which page cleaning
   begins: 1000 x 0.60 = 600
```


End of MLRU cleaning

You can also specify the point at which MLRU cleaning can end.

The **lru_min_dirty** value in the BUFFERPOOL configuration parameter specifies the acceptable percentage of buffers in an MLRU queue. For example, if **lru_min_dirty** is set to `50.00`, page cleaning is not required when 50 percent of the buffers in an LRU queue are modified. In practice, page cleaning can continue beyond this point, as directed by the page-cleaner threads.

The following example shows how the value of **lru_min_dirty** is applied to the LRU queue to specify the acceptable percent of buffers in an MLRU queue and the point at which page cleaning ends.

```
Buffers specified as 8000
lrus specified as 8
lru_min_dirty specified as 50 percent

The acceptable number of buffers in the MLRU queue and
   the point at which page cleaning can end is equal
   to lru_min_dirty.

Buffers per LRU queue = (8000/8) = 1000

Acceptable number of buffers in MLRU queue and the point
   at which page cleaning can end: 1000 x .50 = 500
```

You can use decimals for the **lru_max_dirty** and the **lru_min_dirty** values. For example, if you set **lru_max_dirty** to `1.0333` and **lru_min_dirty** to `1.0`, this triggers the LRU to write at `3,100` dirty buffers and to stop at `3,000` dirty buffers.

For more information about how the database server flushes the buffer pool, see Flush data to disk.

Read-ahead operations

The database server automatically reads several pages ahead of the current pages that are being processed for a query, unless you disable automatic read ahead operations. Reading ahead enables applications to run faster because they spend less time waiting for disk I/O.

Automatic *read-ahead* requests for pages to be brought into the bufferpool cache during sequential scans of data records improves the performance of a query, including OLTP queries and index scans, when the server detects that the query is encountering I/O.

By default, the database server automatically determines when to issue read-ahead requests and when to stop based on when the query is encountering i/o from disk:

- If queries encounter I/O, the server issues read-ahead requests to improve the performance of the query. This performance improvement occurs because read-ahead requests can greatly increase the speed of database processing by compensating for the slowness of I/O processing relative to the speed of CPU processing.
- If queries are mostly cached, the server detects that no I/O is occurring and does not read ahead.

Use the AUTO_READAHEAD configuration parameter to change the automatic read-ahead mode or to disable automatic read ahead for a query. You can:

- Dynamically change the value of the AUTO_READAHEAD configuration parameter by running an **onmode -wm** or **onmode -wf** command.
- Run a SET ENVIRONMENT AUTO_READAHEAD statement to change the mode or enable or disable automatic read-ahead for a session.

You can use the **onstat -p** command to view database server reads and writes and monitor number of times that a thread was required to wait for a shared-memory latch. The `RA-pgsused` output field shows the number of pages used that the database server read ahead and monitor the database server use of read-ahead.

Use the **onstat -g rah** command to display statistics about read-ahead requests.

Related information:
AUTO_READAHEAD configuration parameter
onstat -p command: Print profile counts
onstat -g rah command: Print read-ahead request statistics

Database server thread access to buffer pages

The database server uses shared-lock buffering to allow more than one database server thread to access the same buffer concurrently in shared memory.

The database server uses two types of buffer locks to provide this concurrency without a loss in thread isolation. The two types of lock access are share and exclusive. (For more information, see Types of buffer locks.)

Flush data to disk

Writing a buffer to disk is called *buffer flushing*.

When a user thread modifies data in a buffer, it marks the buffer as *dirty*. When the database server flushes the buffer to disk, it subsequently marks the buffer as *not dirty* and allows the data in the buffer to be overwritten.

The database server flushes the following buffers:

- Buffer pool (covered in this section)
- Physical-log buffer
  See Flush the physical-log buffer.
- Logical-log buffer
  See Flush the logical-log buffer.

Page-cleaner threads manage buffer flushing. The database server always runs at least one page-cleaner thread. If the database server is configured for more than one page-cleaner thread, the LRU queues are divided among the page cleaners for more efficient flushing. For information about specifying how many page-cleaner threads the database server runs, see the CLEANERS configuration parameter in the *IBM® Informix® Administrator's Reference*.

Flushing the physical-log buffer, the modified shared-memory page buffers, and the logical-log buffer must be synchronized with page-cleaner activity according to specific rules designed to maintain data consistency.

- Flush buffer-pool buffers
  Flushing of the buffers is triggered by specific conditions.
- Flush before-images first
  The before-images of modified pages are flushed to disk before the modified pages themselves.
- Flush the physical-log buffer
  The database server always flushes the contents of the physical-log buffer to disk before any data buffers.
- Synchronize buffer flushing
  When shared memory is first set up, all buffers are empty. As processing occurs, data pages are read from disk into the buffers, and user threads begin to modify these pages.
- Types of writes during flushing
  To provide you with information about the specific condition that prompted buffer-flushing activity, the database server defines three types of writes and counts how often each write occurs.
- Flush the logical-log buffer
  A number of events can cause the logical-log buffer to flush.

Flush buffer-pool buffers

Flushing of the buffers is triggered by specific conditions.

Flushing of the buffers is initiated when:

- The number of buffers in an MLRU queue reaches the number specified by the **lru_max_dirty** value in the BUFFERPOOL configuration parameter.
- The page-cleaner threads cannot keep up. In other words, a user thread must acquire a buffer, but no unmodified buffers are available.
- The database server must execute a checkpoint. (See Checkpoints.)

Automatic LRU tuning affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

Flush before-images first

The before-images of modified pages are flushed to disk before the modified pages themselves.

In practice, the physical-log buffer is flushed first and then the buffers that contain modified pages. Therefore, even when a shared-memory buffer page must be flushed because a user thread is trying to acquire a buffer but none is available (a foreground write), the buffer pages cannot be flushed until the before-image of the page has been written to disk.

Flush the physical-log buffer

The database server always flushes the contents of the physical-log buffer to disk before any data buffers.

The database server temporarily stores before-images of some of the modified disk pages in the physical-log buffer. If the before-image is written to the physical-log buffer but not to the physical log on disk, the server flushes the physical-log buffer to disk before flushing the modified page to disk.

The following events cause the active physical-log buffer to flush:

- The active physical-log buffer becomes full.
- A modified page in shared memory must be flushed, but the before-image is still in the active physical-log buffer.
- A checkpoint occurs.

The database server uses only one of the two physical-log buffers at a time. This buffer is the active (or current) physical-log buffer. Before the database server flushes the active physical-log buffer to disk, it makes the other buffer the active physical-log buffer so that the server can continue writing to a buffer while the first buffer is being flushed.

Both the physical-log buffer and the physical log help maintain the physical and logical consistency of the data. For information about physical logging, checkpoints, and fast recovery, see Physical logging, checkpoints, and fast recovery.

Synchronize buffer flushing

When shared memory is first set up, all buffers are empty. As processing occurs, data pages are read from disk into the buffers, and user threads begin to modify these pages.

Types of writes during flushing

To provide you with information about the specific condition that prompted buffer-flushing activity, the database server defines three types of writes and counts how often each write occurs.

The types of writes are as follows:

- Foreground write
- LRU write
- Chunk write

To see the write counts that the database server maintains, run the **onstat -F** command.

If you implement mirroring for the database server, data is always written to the primary chunk first. The write is then repeated on the mirror chunk. Writes to a mirror chunk are included in the counts.

- Foreground write
  Whenever an **sqlexec** thread writes a buffer to disk, it is termed a *foreground write*. A foreground write occurs when an **sqlexec** thread searches through the LRU queues on behalf of a user but cannot locate an empty or unmodified buffer.
- LRU write
  LRU writes are performed by page cleaners rather than by **sqlexec** threads. The database server performs LRU writes as background writes that typically occur when the percentage of dirty buffers exceeds the percent that is specified for **lru_max_dirty** in the BUFFERPOOL configuration parameter.
- Chunk write
  Chunk writes are commonly performed by page-cleaner threads during a checkpoint or, possibly, when every page in the shared-memory buffer pool is modified. Chunk writes, which are performed as sorted writes, are the most efficient writes available to the database server.

Related reference:
Monitor buffers
Related information:
onstat -F command: Print counts

Foreground write

Whenever an **sqlexec** thread writes a buffer to disk, it is termed a *foreground write*. A foreground write occurs when an **sqlexec** thread searches through the LRU queues on behalf of a user but cannot locate an empty or unmodified buffer.

To make space, the **sqlexec** thread flushes pages, one at a time, to hold the data to be read from disk. For more information, see FIFO/LRU queues.

If the **sqlexec** thread must perform buffer flushing just to acquire a shared-memory buffer, performance can suffer. Foreground writes must be avoided. To display a count of the number of foreground writes, run **onstat -F**. If you find that foreground writes are occurring on a regular basis, tune the value of the page-cleaning parameters. Either increase the number of page cleaners or decrease the BUFFERPOOL **lru_max_dirty** value.

LRU write

LRU writes are performed by page cleaners rather than by **sqlexec** threads. The database server performs LRU writes as background writes that typically occur when the percentage of dirty buffers exceeds the percent that is specified for **lru_max_dirty** in the BUFFERPOOL configuration parameter.

A foreground write can trigger an LRU write. When a foreground write occurs, the **sqlexec** thread that performed the write alerts a page-cleaner to wake up and clean the LRU for which it performed the foreground write.

In an appropriately tuned system, page cleaners ensure that enough unmodified buffer pages are available for storing pages to be read from disk. Thus, **sqlexec** threads that perform a query are not required to flush a page to disk before they read in the disk pages required by the query. This condition can result in significant performance gains for queries that do not make use of foreground writes.

LRU writes are preferred over foreground writes because page-cleaner threads perform buffer writes much more efficiently than **sqlexec** threads do. To monitor both types of writes, use **onstat -F**.

Chunk write

Chunk writes are commonly performed by page-cleaner threads during a checkpoint or, possibly, when every page in the shared-memory buffer pool is modified. Chunk writes, which are performed as sorted writes, are the most efficient writes available to the database server.

During a chunk write, each page-cleaner thread is assigned to one or more chunks. Each page-cleaner thread reads through the buffer headers and creates an array of pointers to pages that are associated with its specific chunk. (The page cleaners have access to this information because the chunk number is contained within the physical page number address, which is part of the page header.) This sorting minimizes head movement (disk seek time) on the disk and enables the page-cleaner threads to use the big buffers during the write, if possible.

In addition, because user threads must wait for the checkpoint to complete, the page-cleaner threads are not competing with many threads for CPU time. As a result, the page-cleaner threads can finish their work with less context switching.

Flush the logical-log buffer

A number of events can cause the logical-log buffer to flush.

The database server uses the shared-memory logical-log buffer as temporary storage for records that describe modifications to database server pages. From the logical-log buffer, these records of changes are written to the current logical-log file on disk and eventually to the logical-log backup media. For a description of logical logging, see Logical log.

Five events cause the current logical-log buffer to flush:

- The current logical-log buffer becomes full.
- A transaction is prepared or committed in a database with unbuffered logging.
- A nonlogging database session terminates.
- A checkpoint occurs.
- A page is modified that does not require a before-image in the physical log.

The following topics explain each of these events in detail.

- After a transaction is prepared or terminated in a database with unbuffered logging
  A number of log records cause flushing of the logical-log buffers in a database that uses unbuffered logging.
- When a session that uses nonlogging databases or unbuffered logging terminates
  Even for nonlogging databases, the database server logs certain activities that alter the database schema, such as the creation of tables or extents.
- When a checkpoint occurs
  Checkpoints occur after specific events.
- When a page is modified that does not require a before-image in the physical-log file
  When a page is modified that does not require a before-image in the physical log, the logical-log buffer must be flushed before that page is flushed to disk.

After a transaction is prepared or terminated in a database with unbuffered logging

A number of log records cause flushing of the logical-log buffers in a database that uses unbuffered logging.

The log records that cause flushing are:

- COMMIT
- PREPARE
- XPREPARE
- ENDTRANS

For a comparison of buffered versus unbuffered logging, see the SET LOG statement in the *IBM® Informix® Guide to SQL: Syntax*.

When a session that uses nonlogging databases or unbuffered logging terminates

Even for nonlogging databases, the database server logs certain activities that alter the database schema, such as the creation of tables or extents.

When the database server terminates sessions that use unbuffered logging or nonlogging databases, the logical-log buffer is flushed to make sure that any logging activity is recorded.

When a checkpoint occurs

Checkpoints occur after specific events.

For a detailed description of the events that occur during a checkpoint, see Checkpoints.

When a page is modified that does not require a before-image in the physical-log file

When a page is modified that does not require a before-image in the physical log, the logical-log buffer must be flushed before that page is flushed to disk.

Buffer large-object data

Simple large objects (TEXT or BYTE data) can be stored in either dbspaces or blobspaces. Smart large objects (CLOB or BLOB data) are stored only in sbspaces.

The database server uses different methods to access each type of storage space. The following topics describe buffering methods for each.

- Write simple large objects
  The database server writes simple large objects to disk pages in a dbspace in the same way that it writes any other data type.
- Access smart large objects
  The database server accesses smart large objects through the shared-memory buffers, in the same way that it accesses data that is stored in a dbspace. However, the user-data portion of a smart large object is buffered at a lower priority than normal buffer pages to prevent flushing data of higher value out of the buffer pool. Buffering permits faster access to smart large objects that are accessed frequently.

Write simple large objects

The database server writes simple large objects to disk pages in a dbspace in the same way that it writes any other data type.

For more information, see Flush data to disk.

You can also assign simple large objects to a blobspace. The database server writes simple large objects to a blobspace differently from the way that it writes other data to a shared-memory buffer and then flushes it to disk. For a description of blobspaces, see the chapter on disk structure and storage in the *IBM® Informix® Administrator's Reference*.

- Blobpages and shared memory
  Blobspace blobpages store large amounts of data.
- Creation of simple large objects
  When simple-large-object data is written to disk, the row to which it belongs might not exist yet.
- Creation of blobpage buffers
  To receive simple large object data from the application process, the database server creates a pair of blobspace buffers, one for reading and one for writing, each the size of one blobspace blobpage. Each user has only one set of blobspace buffers and, therefore, can access only one simple large object at a time.

Blobpages and shared memory

Blobspace blobpages store large amounts of data.

The database server does not create or access blobpages by way of the shared-memory buffer pool, and it does not write blobspace blobpages to either the logical or physical logs.

If blobspace data passed through the shared-memory pool, it might dilute the effectiveness of the pool by driving out index pages and data pages. Instead, blobpage data is written directly to disk when it is created.

To reduce logical-log and physical-log traffic, the database server writes blobpages from magnetic media to dbspace backup tapes and logical-log backup tapes in a different way than it writes dbspace pages. For a description of how blobspaces are logged, see Log blobspaces and simple large objects.

Creation of simple large objects

When simple-large-object data is written to disk, the row to which it belongs might not exist yet.

During an insert, for example, the simple large object is transferred before the rest of the row data. After the simple large object is stored, the data row is created with a 56-byte descriptor that points to its location. For a description of how simple large objects are stored physically, see the structure of a dbspace blobpage in the disk storage and structure chapter of the *IBM® Informix® Administrator's Reference*.

Creation of blobpage buffers

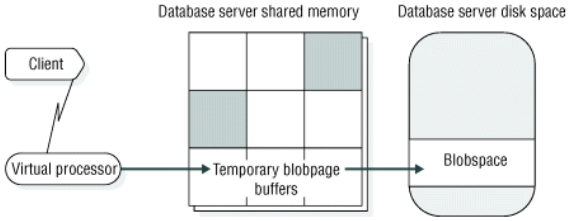
To receive simple large object data from the application process, the database server creates a pair of blobspace buffers, one for reading and one for writing, each the size of one blobspace blobpage. Each user has only one set of blobspace buffers and, therefore, can access only one simple large object at a time.

Simple large object data is transferred from the client-application process to the database server in 1 KB segments. The database server begins filling the blobspace buffers with the 1 KB pieces and attempts to buffer two blobpages at a time. The database server buffers two blobpages so that it can determine when to add a forwarding pointer from one page to the next. When it fills the first buffer and discovers that more data remains to transfer, it adds a forward pointer to the next page before it writes the page to disk. When no more data remains to transfer, the database server writes the last page to disk without a forward pointer.

When the thread begins writing the first blobspace buffer to disk, it attempts to perform the I/O based on the user-defined blobpage size. For example, if the blobpage size is 32 KB, the database server attempts to read or write the data in 32,768-byte increments. If the underlying hardware (such as the disk controller) cannot transfer this amount of data in a single operation, the operating-system kernel loops internally (in kernel mode) until the transfer is complete.

The blobspace buffers remain until the thread that created them is finished. When the simple large object is written to disk, the database server deallocates the pair of blobspace buffers. The following figure illustrates the process of writing a simple large object to a blobspace.

Figure 1. Writing simple large object to a blobspace



Blobspace blobpages are allocated and tracked with the free-map page. Links that connect the blobpages and pointers to the next blobpage segments are created as necessary.

A record of the operation (insert, update, or delete) is written to the logical-log buffer.

Related concepts:
Buffer pool portion of shared memory

Access smart large objects

The database server accesses smart large objects through the shared-memory buffers, in the same way that it accesses data that is stored in a dbspace. However, the user-data portion of a smart large object is buffered at a lower priority than normal buffer pages to prevent flushing data of higher value out of the buffer pool. Buffering permits faster access to smart large objects that are accessed frequently.

A smart large object is stored in an sbspace. You cannot store simple large objects in an sbspace, and you cannot store smart large objects in a blobspace. An sbspace consists of a user-data area and a metadata area. The user-data area contains the smart-large-object data. The metadata area contains information about the content of the sbspace. For more information about sbspaces, see Sbspaces.

Because smart large objects pass through the shared-memory buffer pool and can be logged, you must consider them when you allocate buffers. Use the BUFFERPOOL configuration parameter to allocate shared-memory buffers. As a general rule, try to have enough buffers to hold two smart-large-object pages for each concurrently open smart large object. (The additional page is available for read-ahead purposes.) For more information about tuning buffers for smart large objects, see your *IBM® Informix® Performance Guide*.

Use the LOGBUFF configuration parameter to specify the size of the logical-log buffer. For information about setting each of the following configuration parameters, see the *IBM Informix Administrator's Reference*:

- BUFFERPOOL
- LOGBUFF

The user-data area of smart large objects that are logged does not pass through the physical log, so changing the PHYSBUFF parameter is not required for smart large objects.

For more information about the structure of an sbspace, see sbspace structure in the disk structures and storage chapter of the *IBM Informix Administrator's Reference*. For information about creating an sbspace, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Memory use on 64-bit platforms

With 64-bit addressing, you can have larger buffer pools to reduce the amount of I/O operations to obtain data from disks.

Because 64-bit platforms allow for larger memory-address space, the maximum values for the following memory-related configuration parameters are larger on 64-bit platforms:

- BUFFERPOOL
- CLEANERS
- DS_MAX_QUERIES
- DS_TOTAL_MEMORY
- LOCKS
- SHMADD
- SHMVIRTSIZE

The machine notes for each 64-bit platform lists the maximum values for these configuration parameters and platform-specific parameters such as SHMMAX. For more information about the configuration parameters, see the *IBM® Informix® Administrator's Reference* and the chapter on shared memory in the *IBM Informix Performance Guide*.

Manage shared memory

These topics inform you how to perform the following tasks, which concern managing shared memory:

- Setting the shared-memory configuration parameters
- Setting up shared memory
- Turning residency on or off for the resident portion of the database server shared memory
- Adding a segment to the virtual portion of shared memory
- Reserving memory for critical activities
- Maintaining a targeted amount of memory in applications with memory limitations
- Monitoring shared memory

These topics do not cover the DS_TOTAL_MEMORY configuration parameter. This parameter places a ceiling on the allocation of memory for decision-support queries. For information about this parameter, see your *IBM® Informix® Performance Guide*.

- Set operating-system shared-memory configuration parameters
  Several operating-system configuration parameters can affect the use of shared memory by the database server.
- Set database server shared-memory configuration parameters
  You can modify the configuration parameters that affect the resident, buffer pool, or virtual portion of shared memory.
- Set SQL statement cache parameters
  There are different ways that you can configure the SQL statement cache.
- Set up shared memory
  To set up shared memory, take the database server offline and then online.
- Turn residency on or off for resident shared memory
  You can turn residency on or off for the resident portion of shared memory.
- Add a segment to the virtual portion of shared memory
  You can use the **-a** option of the **onmode** utility to add a segment of specified size to virtual shared memory.
- Reserve memory for critical activities
  You can reserve a specific amount of memory for use when critical activities (such as rollback activities) are required and the database server has limited free memory. This prevents the database server from crashing if the server runs out of free memory during critical activities.
- Configure the server response when memory is critically low
  You can configure the actions that primary or standard database server takes when memory is critically low. You can specify the criteria for terminating sessions based on idle time, memory usage, and other factors so that the targeted application can continue to process. Low-memory responses are useful for embedded applications that have memory limitations.
- Monitor shared memory
  These topics describe how to monitor shared-memory segments, the shared-memory profile, and the use of specific shared-memory resources (buffers, latches, and locks).
- Deleting shared memory segments after a server failure
  You must close shared memory segments after a database server failure.

Set operating-system shared-memory configuration parameters

Several operating-system configuration parameters can affect the use of shared memory by the database server.

Parameter names are not provided because names vary among platforms, and not all parameters exist on all platforms. The following list describes these parameters by function:

- Maximum operating-system shared-memory segment size, expressed in bytes or KB

- Minimum shared-memory segment size, expressed in bytes
- Maximum number of shared-memory identifiers
- Lower-boundary address for shared memory
- Maximum number of attached shared-memory segments per process
- Maximum amount of systemwide shared memory

UNIX only:

- Maximum number of semaphore identifiers
- Maximum number of semaphores
- Maximum number of semaphores per identifier

On UNIX, the machine notes file contains recommended values that you use to configure operating-system resources. Use these recommended values when you configure the operating system. For information about how to set these operating-system parameters, consult your operating-system manuals.

For specific information about your operating-system environment, see the machine notes file that is provided with the database server.

- Maximum shared-memory segment size
  When the database server creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible.
- Semaphores (UNIX)
  The database server operation requires one UNIX semaphore for each virtual processor, one for each user who connects to the database server through shared memory (**ipcshm** protocol), six for database server utilities, and sixteen for other purposes.

Related information:
UNIX configuration parameters that affect CPU utilization
Windows configuration parameters that affect CPU utilization

Maximum shared-memory segment size

When the database server creates the required shared-memory segments, it attempts to acquire as large an operating-system segment as possible.

The first segment size that the database server tries to acquire is the size of the portion that it is allocating (resident, virtual, or communications), rounded up to the nearest multiple of 8 KB.

The database server receives an error from the operating system if the requested segment size exceeds the maximum size allowed. If the database server receives an error, it divides the requested size by two and tries again. Attempts at acquisition continue until the largest segment size that is a multiple of 8 KB can be created. Then the database server creates as many additional segments as it requires.

- Using more than two gigabytes of memory (Windows)
  The database server can access shared-memory segments larger than two gigabytes on Windows.
- Maximum number of shared-memory identifiers (UNIX)
  The operating system identifies each shared-memory segment with a shared-memory identifier. Shared-memory identifiers affect the database server operation when a virtual processor attempts to attach to shared memory.

Using more than two gigabytes of memory (Windows)

The database server can access shared-memory segments larger than two gigabytes on Windows.

For Windows version 2003 and earlier, you must enable this feature with an entry in the Windows boot file. Enabling larger shared-memory segments is referred to by Microsoft as 4-gigabyte tuning (4GT).

To add the entry, edit the boot.ini file (in the top level, or root directory). You can either add a boot option or use the currently existing boot option. To enable support for more than two gigabytes, add the following text to the end of the boot line:

```
/3GB
```

The following example has support for more than two gigabytes enabled:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Windows NT
Workstation Version 4.00"
/3GB
```

The maximum size of the shared-memory segment depends on the operating system, but it is approximately 3 gigabytes for Windows without additional drivers.

Maximum number of shared-memory identifiers (UNIX)

The operating system identifies each shared-memory segment with a shared-memory identifier. Shared-memory identifiers affect the database server operation when a virtual processor attempts to attach to shared memory.

For most operating systems, virtual processors receive identifiers on a first-come, first-served basis, up to the limit that is defined for the operating system as a whole. For more information about shared-memory identifiers, see How virtual processors attach to shared memory.

You might be able to calculate the maximum amount of shared memory that the operating system can allocate by multiplying the number of shared-memory identifiers by the maximum shared-memory segment size.

Semaphores (UNIX)

The database server operation requires one UNIX semaphore for each virtual processor, one for each user who connects to the database server through shared memory (**ipcshm** protocol), six for database server utilities, and sixteen for other purposes.

Set database server shared-memory configuration parameters

You can modify the configuration parameters that affect the resident, buffer pool, or virtual portion of shared memory.

Set parameter for buffer pool shared memory

The BUFFERPOOL configuration parameter in the onconfig file specifies information about a buffer pool. Each page size that is used by the database server requires a buffer pool, which is represented in the onconfig file by a BUFFERPOOL configuration parameter entry.

Set parameters for resident shared memory

The following list contains parameters in the onconfig file that specify the configuration of the buffer pool and the internal tables in the resident portion of shared memory. Before any changes that you make to the configuration parameters take effect, you must shut down and restart the database server.

LOCKS
: Specifies the initial number of locks for database objects; for example, rows, key values, pages, and tables.

LOGBUFF
: Specifies the size of the logical-log buffers.

PHYSBUFF
: Specifies the size of the physical-log buffers.

RESIDENT
: Specifies residency for the resident portion of the database server shared memory.

SERVERNUM
: Specifies a unique identification number for the database server on the local host computer.

SHMTOTAL
: Specifies the total amount of memory to be used by the database server.

Set parameters for virtual shared memory

The following list contains the configuration parameters that you use to configure the virtual portion of shared memory:

DS_HASHSIZE
: Number of hash buckets for lists in the data-distribution cache.

DS_POOLSIZE
: Maximum number of entries in the data-distribution cache.

PC_HASHSIZE
: Specifies the number of hash buckets for the UDR cache and other caches that the database server uses.

PC_POOLSIZE
: Specifies the number of UDRs (SPL routines and external routines) that can be stored in the UDR cache. In addition, this parameter specifies the size of other database server caches, such as the typename cache and the opclass cache.

SHMADD
: Specifies the size of dynamically added shared-memory segments.

SHMNOACCES
: Specifies a list of virtual memory address ranges that are not used to attach shared memory. Use this parameter to avoid conflicts with other processes.

EXTSHMADD
: Specifies the size of a virtual-extension segment added when a user-defined routine or a DataBlade routine runs in a user-defined virtual processor.

SHMTOTAL
: Specifies the total amount of memory to be used by the database server.

SHMVIRTSIZE
: Specifies the initial size of the virtual portion of shared memory.

STACKSIZE
: Specifies the stack size for the database server user threads.

Set parameters for shared-memory performance

The following configuration parameters affect shared-memory performance.

AUTO_READAHEAD
: Specifies the automatic read-ahead mode or disables automatic read-ahead operations for a query. Automatic read-ahead operations help improve query performance by issuing asynchronous page requests when the database server detects that the query is encountering I/O. Asynchronous page requests can improve query performance by overlapping query processing with the processing necessary to retrieve data from disk and put it in the buffer pool.

CKPTINTVL
: Specifies the maximum number of seconds that can elapse before the database server checks if a checkpoint is required and the RTO_SERVER_RESTART configuration parameter is not set to turn on automatic checkpoint tuning.

CLEANERS
: Specifies the number of page-cleaner threads that the database server is to run.

Related information:
Configuration parameters that affect memory utilization
Database configuration parameters
Shared memory
Modifying the onconfig file

Set SQL statement cache parameters

There are different ways that you can configure the SQL statement cache.

The following table shows ways to configure the SQL statement cache.

Table 1. Configure the SQL statement cache

Configuration parameter	Purpose	The onmode command
STMT_CACHE	Turns on, enables, or disables the SQL statement cache in memory. If turned on, specifies whether the SQL statement cache can hold a parsed and optimized SQL statement.	**onmode -e *mode*** **onmode -wm STMT_CACHE=0** (0 is mode 'off') **onmode -wm STMT_CACHE=1** (1 is mode 'on') **onmode -wm STMT_CACHE=2** (2 is mode 'enable')
STMT_CACHE_HITS	Specifies the number of hits (references) to a statement before it is fully inserted into the SQL statement cache.	**onmode -W STMT_CACHE_HITS** **onmode -wm STMT_CACHE_HITS**=*<numhits>*
STMT_CACHE_NOLIMIT	Controls whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.	**onmode -W STMT_CACHE_NOLIMIT** **onmode -wm STMT_CACHE_NOLIMIT**={0/1}
STMT_CACHE_NUMPOOL	Defines the number of memory pools for the SQL statement cache.	None
STMT_CACHE_SIZE	Specifies the size of the SQL statement cache.	None
STMT_QUERY_PLAN	Specifies the query plan from any query that exists in the Statement Cache.	**onmode -wm STMT_QUERY_PLAN**={0/1}

Use the following **onstat** options to monitor the SQL statement cache:

- **onstat -g ssc**
- **onstat -g ssc all**
- **onstat -g ssc pool**

For more information about these configuration parameters, **onstat -g** options, and **onmode** commands, see the *IBM® Informix® Administrator's Reference*.

For more information about using the SQL statement cache, monitoring it with the **onstat -g** options, and tuning the configuration parameters, see improving query performance in the *IBM Informix Performance Guide*. For details on qualifying and identical statements, see the *IBM Informix Guide to SQL: Syntax*.

Set up shared memory

To set up shared memory, take the database server offline and then online.

For information about how to take the database server from online mode to offline, see Changing database server operating modes.

Turn residency on or off for resident shared memory

You can turn residency on or off for the resident portion of shared memory.

Residency can be turned on or off in either of the following two ways:

- Use the **onmode** utility to reverse the state of shared-memory residency immediately while the database server is in online mode.
- Change the RESIDENT parameter in the onconfig file to turn shared-memory residency on or off for the next time that you set up the database server shared memory.

For a description of the resident portion of shared memory, see Resident portion of shared memory.

- Turn residency on or off in online mode
  To turn residency on or off while the database server is in online mode, use the **onmode** utility.
- Turn residency on or off when restarting the database server
  You can use a text editor to turn residency on or off.

Turn residency on or off in online mode

To turn residency on or off while the database server is in online mode, use the **onmode** utility.

To turn on residency immediately for the resident portion of shared memory, run the following command:**% onmode -r**

To turn off residency immediately for the resident portion of shared memory, run the following command: **% onmode -n**

These commands do not change the value of the RESIDENT parameter in the onconfig file. That is, this change is not permanent, and residency reverts to the state specified by the RESIDENT parameter the next time that you set up shared memory. On UNIX, you must be **root** or user **informix** to turn residency on or off. On Windows, you must be a user in the **Informix® Admin** group to turn residency on or off.

Turn residency on or off when restarting the database server

You can use a text editor to turn residency on or off.

To change the current state of residency, use a text editor to locate the RESIDENT parameter. Set RESIDENT to `1` to turn residency on or to `0` to turn residency off, and rewrite the file to disk. Before the changes take effect, you must shut down and restart the database server.

Add a segment to the virtual portion of shared memory

You can use the **-a** option of the **onmode** utility to add a segment of specified size to virtual shared memory.

You are not normally required to add segments to virtual shared memory because the database server automatically adds segments as necessary.

The option to add a segment with the **onmode** utility is useful if the number of operating-system segments is limited, and the initial segment size is so low, relative to the amount that is required, that the operating-system limit of shared-memory segments is nearly exceeded.

Related concepts:
Size of the virtual portion of shared memory

Reserve memory for critical activities

You can reserve a specific amount of memory for use when critical activities (such as rollback activities) are required and the database server has limited free memory. This prevents the database server from crashing if the server runs out of free memory during critical activities.

If you enable the new LOW_MEMORY_RESERVE configuration parameter by setting it to a specified value in kilobytes, critical activities, such as rollback activities, can complete even when a user is getting out of memory errors. If the value of LOW_MEMORY_RESERVE is 0, the low memory reserve functionality is turned off.

For example, 512 kilobytes is a reasonable amount of reserved memory. To reserve 512 kilobytes, specify:

```
LOW_MEMORY_RESERVE 512
```

You can also use the **onmode -wm** or **onmode -wf** command to dynamically adjust the value of the LOW_MEMORY_RESERVE configuration parameter.

Use the **onstat -g seg** command to monitor the LOW_MEMORY_RESERVE value. Look for the last two lines of output, which contain the phrase "`low memory reserve`." The first of these output lines shows the size of memory reserved in bytes. The second of these lines shows the number times that the database server has used this memory and the maximum memory required. Both of these values are reset when the server is restarted.

Related information:
LOW_MEMORY_RESERVE configuration parameter
onstat -g seg command: Print shared memory segment statistics

onmode -wf, -wm: Dynamically change certain configuration parameters

Configure the server response when memory is critically low

You can configure the actions that primary or standard database server takes when memory is critically low. You can specify the criteria for terminating sessions based on idle time, memory usage, and other factors so that the targeted application can continue to process. Low-memory responses are useful for embedded applications that have memory limitations.

To set up automatic low-memory management on a primary or standard server:

- Set the LOW_MEMORY_MGR configuration parameter to `1`, which enables low-memory management when the database server starts.
- Set the threshold parameters for the amount of memory to maintain by using an SQL administration API command with the scheduler lmm enable argument.
- Verify that the SHMTOTAL configuration parameter is set to a positive integer value.

To disable automatic low-memory management, run an SQL administration API command with the scheduler lmm disable argument.

- Scenario for maintaining a targeted amount of memory
  The scenario in this topic shows how you can maintain a targeted amount of memory in applications that have memory limitations.

Related information:
LOW_MEMORY_MGR configuration parameter
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)
onstat -g lmm command: Print low memory management information

Scenario for maintaining a targeted amount of memory

The scenario in this topic shows how you can maintain a targeted amount of memory in applications that have memory limitations.

Suppose you want to specify that when the database server has 10 MB or less of free memory, it starts running the low memory management processes that can stop applications and free memory. Suppose you also want to specify that the server stops running the low memory management processes when the server has 20 MB or more of free memory:

1. Set the LOW_MEMORY_MGR configuration parameter to `1` and restart the server, or run an **onmode -wf** command to change the value of the LOW_MEMORY_MGR configuration parameter.
2. Run an SQL administration API command with the scheduler lmm enable argument and low memory parameters, as follows:

```
EXECUTE FUNCTION task("scheduler lmm enable",
        "LMM START THRESHOLD", "10MB",
        "LMM STOP THRESHOLD", "20MB",
        "LMM IDLE TIME", "300");
```

3. Run the **onstat -g lmm** command to display information about automatic low memory management settings, including the amount of memory that the server is attempting to maintain, the amount of memory currently used by the server, the low memory start and stop thresholds, and other memory-related statistics.
   You can also view low memory management information in the online.log file.

Related information:
LOW_MEMORY_MGR configuration parameter
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)
onstat -g lmm command: Print low memory management information

Monitor shared memory

These topics describe how to monitor shared-memory segments, the shared-memory profile, and the use of specific shared-memory resources (buffers, latches, and locks).

You can use the **onstat -o** utility to capture a static snapshot of database server shared memory for later analysis and comparison.

- Monitor shared-memory segments
  Monitor the shared-memory segments to determine the number and size of the segments that the database server creates.
- Monitor the shared-memory profile and latches
  Monitor the database server profile to analyze performance and the use of shared-memory resources.
- Monitor buffers
  You can obtain both statistics on buffer use and information about specific buffers.

Monitor shared-memory segments

Monitor the shared-memory segments to determine the number and size of the segments that the database server creates.

The database server allocates shared-memory segments dynamically, so these numbers can change. If the database server is allocating too many shared-memory segments, you can increase the SHMVIRTSIZE configuration parameter. For more information, see the topics about configuration parameters in the *IBM® Informix® Administrator's Reference*.

The **onstat -g seg** command lists information for each shared-memory segment, including the address and size of the segment, and the amount of memory that is free or in use. For an example of **onstat -g seg** output, see information about the **onstat** utility in the *IBM Informix Administrator's Reference*.

Monitor the shared-memory profile and latches

Monitor the database server profile to analyze performance and the use of shared-memory resources.

You can obtain statistics on latch use and information about specific latches. These statistics provide a measure of the system activity.

To reset these statistics to zero, use the **onstat -z** option. For a description of all the fields that **onstat** displays, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

- Command-line utilities to monitor shared memory and latches
  Use command-line utilities to monitor shared memory and latches.
- SMI tables
  Query the **sysprofile** table to obtain shared-memory statistics.

Command-line utilities to monitor shared memory and latches

Use command-line utilities to monitor shared memory and latches.

You can use the following command-line utilities to monitor shared memory and latches:

onstat -s
Use **onstat -s** command to obtain latch information.

onstat -p
Run **onstat -p** to display statistics on database server activity and waiting latches (in the **lchwaits** field). For an example of **onstat -p** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

SMI tables

Query the **sysprofile** table to obtain shared-memory statistics.

This table contains all of the statistics available in **onstat -p** output except the **ovbuff**, **usercpu**, and **syscpu** statistics.

Monitor buffers

You can obtain both statistics on buffer use and information about specific buffers.

The statistical information includes the percentage of data writes that are cached to buffers and the number of times that threads were required wait to obtain a buffer. The percentage of writes that are cached is an important measure of performance. The number of waits for buffers gives a measure of system concurrency.

Information about specific buffers includes a listing of all the buffers in shared memory that are held by a thread. You can use this information to track the status of a particular buffer. For example, you can determine whether another thread is waiting for the buffer.

You can obtain statistics that relate to buffer availability and information about the buffers in each LRU queue. The statistical information includes the number of times that the database server attempted to exceed the maximum number of buffers and the number of writes to disk (categorized by the event that caused the buffers to flush). These statistics help you determine if the number of buffers is appropriate. Information about the buffers in each LRU queue consists of the length of the queue and the percentage of the buffers in the queue that were modified.

You can obtain information about buffer pool activity from the **onstat** utility, the **sysprofile** SMI table.

onstat commands to monitor buffers

You can use the following **onstat** commands to monitor buffers:

onstat -g buf
Run the **onstat -g buf** command to obtain statistics about how active and efficient each buffer is. The following types of statistics are shown:

- Page reads and writes
- Caching percentages
- Waits for buffers
- Flushes
- Extensions of the buffer pool
- Buffer pool segments
- Fast cache

onstat -B
Run the **onstat -B** command to obtain information about all of the buffers that are not on the free-list, including:

- The shared memory address of the buffer
- The address of the thread that currently holds the buffer
- The address of the first thread that is waiting for each buffer
- Information about buffer pools

onstat -b
Run the **onstat -b** command to obtain the following information about each buffer:

- Address of each buffer that is currently held by a thread
- Page numbers for the page that is held in the buffer
- Type of page that is held in the buffer (for example, data page, tblspace page, and so on)
- Type of lock that is placed on the buffer (exclusive or shared)
- Address of the thread that is holding the buffer
- Address of the first thread that is waiting for each buffer
- Information about buffer pools

You can compare the addresses of the user threads to the addresses that are shown in the **onstat -u** output to obtain the session ID number.

onstat -X
Run the **onstat -X** command to obtain the same information as for **onstat -b**, along with the complete list of all threads that are waiting for buffers, not just the first waiting thread.

onstat -R
Run the **onstat -R** command to show information about buffer pools, the number of buffers in each LRU queue, and the number and percentage of the buffers that are modified or free.

onstat -F
Run the **onstat-F** command to obtain a count by write type of the writes that are performed and the following information about the page cleaners:

- Page-cleaner number
- Page-cleaner shared-memory address
- Current state of the page cleaner
- LRU queue to which the page cleaner was assigned

The sysprofile SMI table

Query the **sysprofile** table to obtain statistics on cached reads and writes, write types, and total buffer waits. The following rows are relevant.

bufreads
Number of reads from buffers

bufwrites
Number of writes to buffers

buffwts
Number of times that any thread was required to wait for a buffer

chunkwrites
Number of chunk writes

dskreads
Number of reads from disk

dskwrites
Number of writes to disk

fgwrites
Number of foreground writes

lruwrites
Number of LRU writes

Related concepts:
Types of writes during flushing
Related information:
onstat -g buf command: Print buffer pool profile information
onstat -b command: Print buffer information for buffers in use
onstat -B command: Prints information about used buffers
onstat -X command: Print thread information
onstat -R command: Print LRU, FLRU, and MLRU queue information
onstat -F command: Print counts
sysprofile

Deleting shared memory segments after a server failure

You must close shared memory segments after a database server failure.

Important: This procedure must be performed by a DBA with experience using IBM® Informix®. Consult technical support for assistance. This procedure is for UNIX systems only.
In the event of a failure of a database server instance, follow this procedure to delete shared memory segments:

1. Log on as user **informix**.
2. Use the **onmode -k** command to take the database server to offline mode and remove shared memory.
3. If the **onmode -k** command fails and the server is not offline, either run the **onclean -k** command, or perform the following steps:
   a. Use the **onstat -g glo** command to display multithreading information.
   b. In the output from the previous command, find the process ID (pid) associated with the first instance of **cpu** in the class column. For example, in the following output from the **onstat -g glo** command, there are four occurrences of **cpu** in the class column, having pids of 2599, 2603, 2604, and 2605:

```
MT global info:
sessions threads  vps      lngspins
0        49       14       1
         sched calls     thread switches yield 0   yield n   yield forever
total:   900100          898846          1238      27763     423778
per sec: 327             325             2         12        151
Virtual processor summary:
 class       vps       usercpu   syscpu    total
 cpu         4         0.92      0.10      1.02
 aio         4         0.02      0.02      0.04
 lio         1         0.00      0.00      0.00
 pio         1         0.00      0.00      0.00
 adm         1         0.00      0.01      0.01
 msc         1         0.00      0.00      0.00
 fifo        2         0.00      0.00      0.00
 total       14        0.94      0.13      1.07
Individual virtual processors:
 vp     pid       class       usercpu   syscpu    total
 1      2599      cpu         0.25      0.06      0.31
 2      2602      adm         0.00      0.01      0.01
 3      2603      cpu         0.23      0.00      0.23
 4      2604      cpu         0.21      0.03      0.24
 5      2605      cpu         0.23      0.01      0.24
 6      2606      lio         0.00      0.00      0.00
 7      2607      pio         0.00      0.00      0.00
 8      2608      aio         0.02      0.02      0.04
 9      2609      msc         0.00      0.00      0.00
 10     2610      fifo        0.00      0.00      0.00
 11     2611      fifo        0.00      0.00      0.00
 12     2612      aio         0.00      0.00      0.00
 13     2613      aio         0.00      0.00      0.00
 14     2614      aio         0.00      0.00      0.00
                  tot         0.94      0.13      1.07
```

   c. Use the **kill** command to terminate (in order) process IDs 2599, 2603, 2604, and 2605.
4. If the shared segments have not been removed then follow these steps:
   a. Determine the server number. The server number can be found by examining the onconfig file of the instance
   b. Add the server number to 21078. For example, if the server number is 1, then add 1 to 21078, giving 21079.
   c. Convert the sum from the previous step to hexadecimal. In the previous example, 21079 is 5257 hexadecimal.
   d. Concatenate 48 to the hex value from the previous step. For example, 525748.
   e. Run the **ipcs** utility as root to display the shared memory segments, if any, left open by the server. Search the key column for the number from 4.d.
   f. Remove each shared memory ID associated with the number from 4.d.

For more information about the **onclean** utility, see the *IBM Informix Administrator's Reference*.

Consult your operating system documentation for the correct **ipcm** syntax for your system.

Data storage

The database server uses physical units of storage to allocate disk space. It stores data in logical units. Unlike the logical units of storage whose size fluctuates, each of the physical units has a fixed or assigned size that is determined by the disk architecture.

The following topics define terms and explain concepts that you must understand to manage disk space. These topics cover the following areas:

- Definitions of the physical and logical units that the database server uses to store data on disk
- Instructions on how to calculate the amount of disk space that you require to store your data
- Guidelines on how to lay out your disk space and where to place your databases and tables
- Instructions on using external tables

The database server uses the following physical units to manage disk space:

Chunks
Pages
Blobpages

Sbpages
Extents

The database server stores data in the following logical units:

Dbspaces
Temporary dbspaces
Blobspaces
Sbspaces
Temporary sbspaces
Plogspace
Extspaces
Databases
Tables
Tblspaces
Partitions and offsets

The database server maintains the following storage structures to ensure physical and logical consistency of data:

Logical log
Physical log
Reserved pages

- Table fragmentation and data storage
  The fragmentation feature gives you more control over where the database stores data. You are not limited to specifying the locations of individual tables and indexes. You can also specify the location of table and index *fragments*, which are different parts of a table or index that are on different storage spaces.
- Amount of disk space needed to store data
  To determine how much disk space you require, follow these steps:
- The storage pool
  Every instance of the database server has a storage pool. The storage pool contains information about the directories, cooked files, and raw devices that the server can use if necessary to automatically expand an existing dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.
- Disk-layout guidelines
  The following goals are typical for efficient disk layout:
- Sample disk layouts
  When setting out to organize disk space, the database server administrator usually has one or more of the following objectives in mind:
- Logical-volume manager
  You can use the logical-volume manager (LVM) utility to manage your disk space through user-defined logical volumes.

Related concepts:
Storage space creation and management
Manage disk space
Related information:
Limits in Informix
Reserved Pages

Chunks

A *chunk* is the largest unit of physical disk dedicated to database server data storage.

Chunks provide administrators with a significantly large unit for allocating disk space. The maximum size of an individual chunk is 4 TB. The number of allowable chunks is 32,766. If you have upgraded from a version before version 10.00, you must run the **onmode -BC 2** command to enable the maximum size of a chunk and the maximum number allowable, otherwise, the maximum chunk size is 2 GB.

The following storage spaces are comprised of chunks:

- Dbspaces
- Blobspaces
- Sbspaces
- Temporary dbspaces
- Temporary sbspaces

When you create a chunk, you specify its path, size, and the associated storage space name.

The database server also uses chunks for mirroring. When you mirror a chunk, the database server maintains two copies of the data on the chunk. Every write operation to a primary chunk is automatically followed by an identical write operation to the mirror chunk. Read operations are evenly divided between the two chunks. If either the primary chunk or the mirror chunk fails, the chunk that failed is marked as down, and the other chunk performs all operations without interrupting the user access to data.

When you create tables, indexes, and other database objects, chunk space is allocated, or assigned, to those objects. Space that is allocated is not necessarily used. For example, when you create a table, you allocate space for it, but that space is not used until you add data to the table. When all the chunks in a dbspace report 0 free pages, you cannot create new database objects in that dbspace. However, you can continue to add data to existing database objects as long as they have unused space. You can monitor chunks by using the **onstat -d** command.

- Disk allocation for chunks
  The database server can use regular operating-system files or *raw disk devices* to store data. On UNIX, you must use raw disk devices to store data whenever performance is important. On Windows, using NTFS files to store data is recommended for ease of administration.
- Extendable chunks
  Extendable chunks are chunks that Informix® can automatically extend or you can manually extend when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space will run out of space and when it will run out of space.

- Partitions and offsets
  The system administrator might divide a physical disk into *partitions*, which are different parts of a disk that have separate path names. Although you must use an entire disk partition when you allocate a chunk on a raw disk device, you can subdivide partitions or cooked files into smaller chunks using offsets.

Related concepts:
Sbspaces
Blobspaces
Dbspaces
Mirroring
Related reference:
Specify names for storage spaces and chunks
Related information:
onstat -d command: Print chunk information
onmode -BC: Allow large chunk mode

Disk allocation for chunks

The database server can use regular operating-system files or *raw disk devices* to store data. On UNIX, you must use raw disk devices to store data whenever performance is important. On Windows, using NTFS files to store data is recommended for ease of administration.

A storage space can be on an NFS-mounted file system using regular operating-system files.

- Disk access on Windows
  On Windows, both raw disks and NTFS use kernel asynchronous I/O (KAIO). The Windows file system manager adds additional overhead to disk I/O, so using raw disks provides slight performance advantages. Because NTFS files are a more standard method of storing data, you must use NTFS files instead of raw disks. Consider using raw disks if your database server requires a large amount of disk access.
- Unbuffered or buffered disk access on UNIX
  You can allocate disk space in two ways. You can either use files that are buffered through the operating system, or you can use unbuffered disk access.

Disk access on Windows

On Windows, both raw disks and NTFS use kernel asynchronous I/O (KAIO). The Windows file system manager adds additional overhead to disk I/O, so using raw disks provides slight performance advantages. Because NTFS files are a more standard method of storing data, you must use NTFS files instead of raw disks. Consider using raw disks if your database server requires a large amount of disk access.

Raw disk space on Windows

On Windows, *raw disk space* can be either a physical drive without a drive letter or a logical disk partition that has been assigned a drive letter using the **Disk Administrator**. The space can either be formatted or unformatted. If it contains data, the data is overwritten after the space has been allocated to the database server. For more information, see Allocating raw disk space on Windows.

NTFS files

You must use NTFS files, not FAT files, for disk space on Windows. For more information, see Allocating NTFS file space on Windows.

Unbuffered or buffered disk access on UNIX

You can allocate disk space in two ways. You can either use files that are buffered through the operating system, or you can use unbuffered disk access.

Files that are buffered through the operating system are often called *cooked* files.

Unbuffered disk access is also called *raw* disk space.

When dbspaces are located on *raw disk devices* (also called *character-special devices*), the database server uses unbuffered disk access.

To create a raw device, configure a *block device* (hard disk) with a raw interface. The storage space that the device provides is called *raw disk space*. A chunk of raw disk space is physically contiguous.

The name of the chunk is the name of the character-special file in the /dev directory. In many operating systems, you can distinguish the character-special file from the block-special file by the first letter in the file name (typically `r`). For example, /dev/rsd0f is the character-special device that corresponds to the /dev/sd0f block-special device.

For more information, see Allocating raw disk space on UNIX.

A *cooked file* is a regular file that the operating system manages. Cooked file chunks and raw disk chunks are equally reliable. Unlike raw disk space, the logically contiguous blocks of a cooked file might not be physically contiguous.

You can more easily allocate cooked files than raw disk space. To allocate a cooked file, you must create the file on any existing partition. The name of the chunk is the complete path name of the file. These steps are described in Allocating cooked file spaces on UNIX.

In a learning environment, where performance is not critical, or for static data, cooked files can be convenient. If you must use cooked UNIX files, store the least frequently accessed data in those files. Store the files in a file system with minimal activity.

For cooked file chunks, the operating system processes all chunk *I/O* from its own buffer pool and ensures that all writes to chunks are physically written to the disk.

Important: While you must generally use raw disk devices on UNIX to achieve better performance, if you enable the DIRECT_IO configuration parameter, the performance for cooked files can approach the performance of raw devices used for dbspace chunks. This occurs because direct I/O bypasses the use of the file system buffers. If you have an AIX® operating system, you can also enable concurrent I/O for IBM® Informix® to use with direct IO when reading and writing to chunks that use cooked files. For more information about using direct IO or concurrent IO, see the *IBM Informix Performance Guide*.
To determine the best device for performance, perform benchmark testing on the system with both types of devices for the dbspace and table layout.

When using raw disks, you are not required to take any special action to create chunks and files that are larger than two gigabytes. If you want to create large chunks in cooked files, or if you want to use the various database export and import utilities with large files, you must ensure that the files systems that hold the large files are appropriately configured.

Extendable chunks

Extendable chunks are chunks that Informix® can automatically extend or you can manually extend when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space will run out of space and when it will run out of space.

Configuring Informix to automatically add more storage space prevents the error that can occur if a partition requires additional storage space and cannot find that space in one of the chunks in the space in which the partition is located.

An extendable chunk must be in a nonmirrored dbspace or temporary dbspace.

You use an SQL administration API command with the modify space sp_sizes argument to modify the extend size and the create size for the space in which your extendable chunk is located.

Related concepts:
Automatic space management
The storage pool
Related tasks:
Marking a chunk as extendable or not extendable
Manually expanding a space or extending an extendable chunk

Partitions and offsets

The system administrator might divide a physical disk into *partitions*, which are different parts of a disk that have separate path names. Although you must use an entire disk partition when you allocate a chunk on a raw disk device, you can subdivide partitions or cooked files into smaller chunks using offsets.

Tip: With a 4-terabyte limit to the size of a chunk, you can avoid partitioning a disk by assigning a single chunk per disk drive.
You can use an offset to indicate the location of a chunk on the disk partition, file, or device. For example, suppose that you create a 1000 KB chunk that you want to divide into two chunks of 500 KB each. You can use an offset of 0 KB to mark the beginning of the first chunk and an offset of 500 KB to mark the beginning of the second chunk.

You can specify an offset whenever you create, add, or drop a chunk from a dbspace, blobspace, or sbspace.

You might also be required to specify an offset to prevent the database server from overwriting partition information.

Related concepts:
Disk-layout guidelines
Related tasks:
Allocating raw disk space on UNIX

Pages

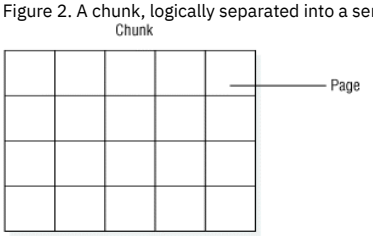
A *page* is the physical unit of disk storage that the database server uses to read from and write to IBM® Informix® databases.

The following figure illustrates the concept of a page, represented by a darkened sector of a disk platter.
Figure 1. A page on disk

On most UNIX platforms, the page size is 2 KB. On Windows, the page size is 4 KB. Because your hardware determines the size of your page, you cannot alter this value.

A chunk contains a certain number of pages, as the following figure illustrates. A page is always entirely contained within a chunk; that is, a page cannot cross chunk boundaries.

Figure 2. A chunk, logically separated into a series of pages



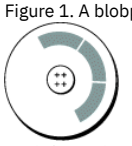
For information about how the database server structures data within a page, see the chapter on disk structures and storage in the *IBM Informix Administrator's Reference*

Blobpages

A *blobpage* is the unit of disk-space allocation that the database server uses to store simple large objects (TEXT or BYTE data) within a blobspace.

You specify blobpage size as a multiple of the database server page size. Because the database server allocates blobpages as contiguous spaces, it is more efficient to store simple large objects in blobpages that are as close to the size of the data as possible. The following figure illustrates the concept of a blobpage, represented as a multiple (three) of a data page.

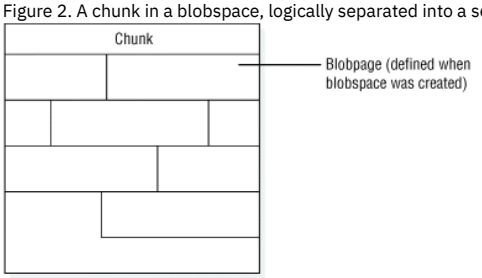
Figure 1. A blobpage on disk



For information about how IBM® Informix® structures data stored in a blobpage, see structure of a blobspace blobpage in the disk structures and storage topics of the *IBM Informix Administrator's Reference*.

Just as with pages in a chunk, a certain number of blobpages compose a chunk in a blobspace, as the following figure illustrates. A blobpage is always entirely contained in a chunk and cannot cross chunk boundaries.

Figure 2. A chunk in a blobspace, logically separated into a series of blobpages



Instead of storing simple-large-object data in a blobspace, you can choose to store it in a dbspace. However, for a simple large object larger than two pages, performance improves when you store it in a blobspace. Simple large objects stored in a dbspace can share a page, but simple large objects stored in a blobspace do not share pages.

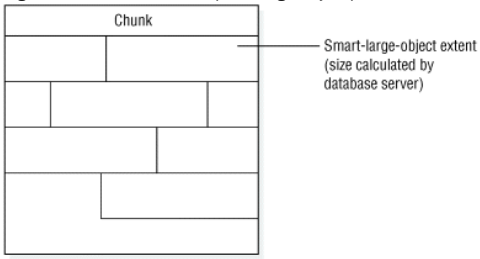
For information about how to determine the size of a blobpage, see Determine blobpage size. For a description of blobspaces, see Blobspaces.

Sbpages

An *sbpage* is the type of page that the database server uses to store smart large objects within an sbspace. Unlike blobpages, sbpages are not configurable. An sbpage is the same size as the database server page, which is usually 2 KB on UNIX and 4 KB on Windows.

The unit of allocation in an sbspace is an extent, whereas the unit of allocation in a blobspace is a blobpage. Just as with pages in a chunk, a certain number of smart large object extents compose a chunk in an sbspace, as the following figure illustrates. An extent is always entirely contained in a chunk and cannot cross chunk boundaries.

Figure 1. A chunk in an sbspace, logically separated into a series of extents



Smart large objects cannot be stored in a dbspace or blobspace. For more information, see Sbspaces, and sbspace structure in the disk structures and storage chapter of the *IBM® Informix® Administrator's Reference*.

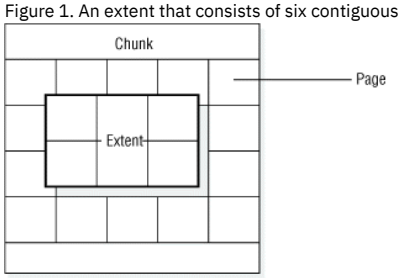
The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For more information, see Extent sizes for sbspaces.

Extents

An extent consists of a collection of contiguous pages that store data for a given table.

When you create a table, the database server allocates a fixed amount of space to contain the data to be stored in that table. (See Tables.) When this space fills, the database server must allocate space for additional storage. The physical unit of storage that the database server uses to allocate both the initial and subsequent storage space is called an *extent*.

The following figure illustrates the concept of an extent.
Figure 1. An extent that consists of six contiguous pages on a raw disk device



Every permanent database table has two extent sizes associated with it. The *initial-extent* size is the number of KB allocated to the table when it is first created. The *next-extent* size is the number of KB allocated to the table when the initial extent (and any subsequent extents) becomes full. For permanent tables and user-defined temporary tables, the next-extent size begins to double after each extent. For system-created temporary tables, the next-extent size begins to double after 4 extents have been added.

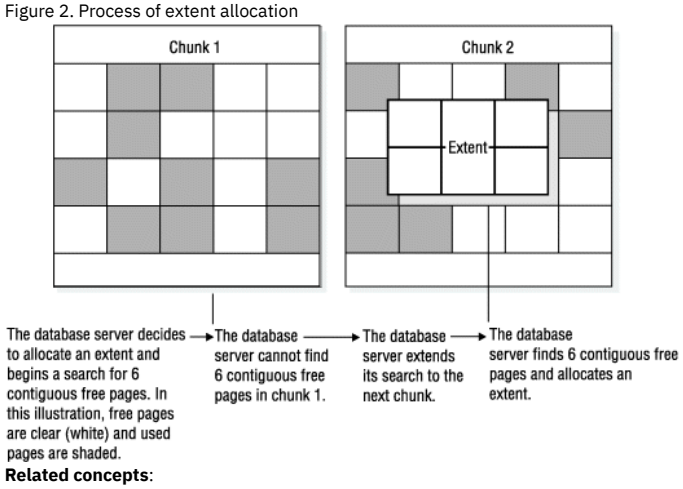
When you create a table, you can specify the size of the initial extent, and the size of the extents to be added as the table grows. You can also modify the size of an extent in a table in a dbspace, and you can modify the size of new subsequent extents. To specify the initial-extent size and next-extent size, use the CREATE TABLE and ALTER TABLE statements. For more information, see the *IBM® Informix® Guide to SQL: Syntax* and disk structures in the *IBM Informix Administrator's Reference*.

When you create a table with a column for CLOB or BLOB data types, you also define extents for an sbspace. For more information, see Storage characteristics of sbspaces.

The following figure shows how the database server allocates six pages for an extent:

- An extent is always entirely contained in a chunk; an extent cannot cross chunk boundaries.
- If the database server cannot find the contiguous disk space that is specified for the next-extent size, it searches the next chunk in the dbspace for contiguous space.

Figure 2. Process of extent allocation



Related concepts:
Tables
Related information:
Extent size doubling

Dbspaces

A *dbspace* is a logical unit that can contain between 1 and 32,766 chunks. The database server uses the dbspace to store databases and tables. Place databases, tables, logical-log files, and the physical log in dbspaces.

When you create a standard or temporary dbspace, you can specify the page size for the dbspace. You cannot specify a page size for blobspaces, sbspaces, or external spaces. If you do not specify a page size, the size of the root dbspace is the default page size.

When you create a standard dbspace, you can specify the first and next extent sizes for the **tblspace** in the dbspace. Specifying the extent sizes reduces the number of **tblspace** extents and reduces the frequency of situations when you must place the **tblspace** extents in non-primary chunks.

You can mirror every chunk in a mirrored dbspace. As soon as the database server allocates a mirror chunk, it flags all space in that mirror chunk as full.

- Control of where simple large object data is stored
  A key responsibility of the database server administrator is to control where the database server stores data.
- Root dbspace
  The *root dbspace* is the initial dbspace that the database server creates.
- Temporary dbspaces
  A *temporary dbspace* is a dbspace reserved exclusively for the storage of temporary tables. It behaves differently from a standard dbspace in many ways.

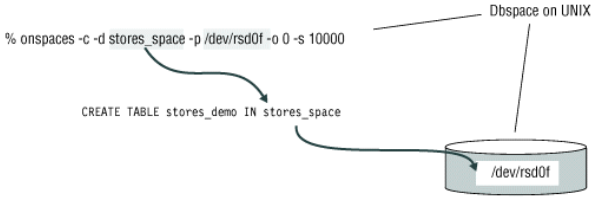
Related concepts:
Chunks

Control of where simple large object data is stored

A key responsibility of the database server administrator is to control where the database server stores data.

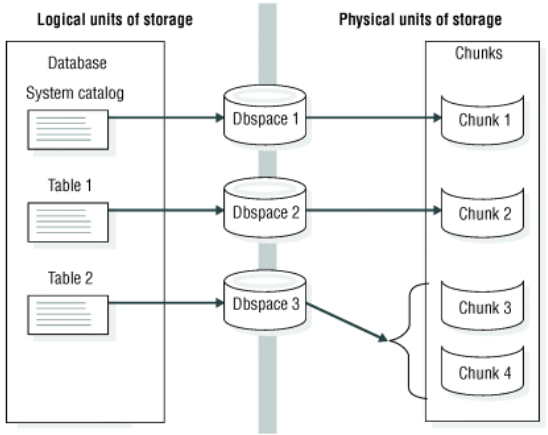
By storing high-use access tables or *critical dbspaces* (root dbspace, physical log, and logical log) on your fastest disk drive, you can improve performance. By storing critical data on separate physical devices, you ensure that when one of the disks that holds noncritical data fails, the failure affects only the availability of data on that disk.

As the following figure shows, to control the placement of databases or tables, you can use the IN *dbspace* option of the CREATE DATABASE or CREATE TABLE statements.
Figure 1. Control table placement with the CREATE TABLE... IN statement



Before you create a database or table in a dbspace, you must first create the dbspace.

A dbspace includes one or more chunks, as the following figure shows. You can add more chunks at any time. A database server administrator must to monitor dbspace chunks for fullness and to anticipate the necessity to allocate more chunks to a dbspace. When a dbspace contains more than one chunk, you cannot specify the chunk in which the data is located.
Figure 2. Dbspaces that link logical and physical units of storage



Related concepts:
Tables
Manage dbspaces
Related reference:
Monitor disk usage

Root dbspace

The *root dbspace* is the initial dbspace that the database server creates.

The root dbspace is special because it contains reserved pages and internal tables that describe and track all physical and logical units of storage. (For more information about these topics, see Tables and the disk structures and storage chapter in the *IBM® Informix® Administrator's Reference*.) The initial chunk of the root dbspace and its mirror are the only chunks created during disk-space setup. You can add other chunks to the root dbspace after disk-space setup.

The following disk-configuration parameters in the onconfig configuration file refer to the first (initial) chunk of the root dbspace:

- ROOTPATH
- ROOTOFFSET
- ROOTNAME
- MIRRORPATH
- MIRROROFFSET
- TBLTBLFIRST
- TBLTBLNEXT

The root dbspace is also the default dbspace location for any database created with the CREATE DATABASE statement.

The root dbspace is the default location for all temporary tables created by the database server to perform requested data management.

See Size of the root dbspace for information about how much space to allocate for the root dbspace. You can also add extra chunks to the root dbspace after you set up database server disk space.

Temporary dbspaces

A *temporary dbspace* is a dbspace reserved exclusively for the storage of temporary tables. It behaves differently from a standard dbspace in many ways.

A temporary dbspace is temporary only in the sense that the database server does not preserve any of its contents when the database server restarts. The database server never drops a temporary dbspace unless it is explicitly directed to do so.

Temporary dbspaces cannot be mirrored by the database server.

Whenever you start the database server, all chunks in temporary dbspaces are recreated from scratch. These chunks can therefore be located on RAM drives if desired.

The database server does not perform logical or physical logging for temporary dbspaces. Because temporary dbspaces are not physically logged, fewer checkpoints and I/O operations occur, which improves performance.

For a temporary table in a standard dbspace, at minimum the server logs table creation, the allocation of extents, and the dropping of the table. In contrast, the database server does not log any operations on tables stored in temporary dbspaces. Logical-log suppression in temporary dbspaces reduces the number of log records to roll forward during logical recovery as well, thus improving the performance during critical downtime.

Temporary dbspaces are never archived by the database server, reducing the size of your storage-space backup.

In addition to temporary tables, the database server uses temporary dbspaces to store the before images of data that is overwritten while backups are occurring and overflow from query processing that occurs in memory. Make sure that you have correctly set the DBSPACETEMP environment variable or parameter to specify dbspaces with enough space for your needs. If there is not enough room in the specified dbspaces, the root dbspace is used. If the root dbspace fills, the backup may fail.

If you have more than one temporary dbspace and execute a SELECT statement into a temporary table, the results of the query are inserted in round robin order.

For detailed instructions on how to create a temporary dbspace, see Creating a temporary dbspace.

Blobspaces

A *blobspace* is a logical storage unit composed of one or more chunks that store only TEXT and BYTE data.

A blobspace stores TEXT and BYTE data in the most efficient way possible. You can store TEXT and BYTE columns associated with distinct tables (see Tables) in the same blobspace.

The database server writes data stored in a blobspace directly to disk. This data does not pass through resident shared memory. If it did, the volume of data might occupy so many of the buffer-pool pages that other data and index pages would be forced out. For the same reason, the database server does not write TEXT or BYTE objects that are assigned to a blobspace to either the logical or physical log. The database server logs blobspace objects by writing them directly from disk to the logical-log backup tapes when you back up the logical logs. Blobspace objects never pass through the logical-log files.

When you create a blobspace, you assign to it one or more chunks. You can add more chunks at any time. One of the tasks of a database server administrator is to monitor the chunks for fullness and anticipate the necessity to allocate more chunks to a blobspace. For instructions on how to monitor chunks for fullness, see Monitor simple large objects in a blobspace. For instructions on how to create a blobspace, add chunks to a blobspace, or drop a chunk from a blobspace, see Manage disk space.

For information about the structure of a blobspace, see the topics about disk structures and storage in the *IBM® Informix® Administrator's Reference*.

Related concepts:
Chunks

Sbspaces

An *sbspace* is a logical storage unit composed of one or more chunks that store *smart large objects*.

Smart large objects consist of CLOB (character large object) and BLOB (binary large object) data types. User-defined data types can also use sbspaces. For more information about data types, see the *IBM® Informix® Guide to SQL: Reference*.

- Advantages of using sbspaces
  Sbspaces have advantages over blobspaces.
- Sbspaces and Enterprise Replication
  Before you define a replication server for Enterprise Replication, you must create an sbspace. Enterprise Replication spools the replicated data to smart large objects.
- Metadata, user data, and reserved area
  As with blobspaces and dbspaces, when you create an sbspace, you assign to it one or more chunks. The first chunk of an sbspace always has three areas.
- Control of where smart large object data is stored
  You specify the data type of a column when you create the table.
- Storage characteristics of sbspaces
  As the database server administrator, you can use the system default values for these storage characteristics, or you can specify them in the **-Df** tags when you create the sbspace with **onspaces -c**. You can change these sbspace characteristics with the **onspaces -ch** option.
- Levels of inheritance for sbspace characteristics
  The four levels of inheritance for sbspace characteristics are system, sbspace, column, and smart large objects. You can use the system default values for sbspace attributes, or override them for specific sbspaces, columns in a table, or smart large objects.
- More information about sbspaces
  There are various tasks related to using and managing sbspaces.
- Temporary sbspaces
  Use a *temporary sbspace* to store temporary smart large objects without metadata logging and user-data logging.

Related concepts:
Chunks

Advantages of using sbspaces

Sbspaces have advantages over blobspaces.

The following are advantages of using sbspaces:

- They have read, write, and seek properties similar to a standard UNIX file.
  Programmers can use functions similar to UNIX and Windows functions to read, write, and seek smart large objects. IBM® Informix® provides this smart-large-object interface in the DataBlade API and the Informix ESQL/C programming interface.
- They are recoverable.
  You can log all write operations on data stored in sbspaces. You can commit or roll back changes if a failure occurs during a transaction.
- They obey transaction isolation modes.
  You can lock smart large objects at different levels of granularity, and the lock durations obey the rules for transaction isolation levels. For more information about locking and concurrency, see your *IBM Informix Performance Guide*.
- Smart large objects within table rows are not required to be retrieved in one statement.
  An application can store or retrieve smart large objects in pieces using either the DataBlade API or the Informix ESQL/C programming interface. For more information about the DataBlade API functions, see the *IBM Informix DataBlade API Function Reference*. For more information about the Informix ESQL/C functions, see the *IBM Informix ESQL/C Programmer's Manual*.

Sbspaces and Enterprise Replication

Before you define a replication server for Enterprise Replication, you must create an sbspace. Enterprise Replication spools the replicated data to smart large objects.

Specify the sbspace name in the CDR_QDATA_SBSPACE configuration parameter. Enterprise Replication uses the default log mode with which the sbspace was created for spooling the row data. The CDR_QDATA_SBSPACE configuration parameter accepts multiple sbspaces, up to a maximum of 32 sbspaces. Enterprise Replication can support a combination of logging and non-logging sbspaces for storing spooled row data. For more information, see the .

You can have Enterprise Replication automatically configure disk space from the storage pool and set the appropriate configuration parameters when defining a replication server. If the CDR_QDATA_SBSPACE or the CDR_DBSPACE configuration parameter is not set or is set to blank, the **cdr define server** command automatically creates the necessary disk space and sets the configuration parameters to appropriate values.

Metadata, user data, and reserved area

As with blobspaces and dbspaces, when you create an sbspace, you assign to it one or more chunks. The first chunk of an sbspace always has three areas.

The following are the three areas of the first chunk of an sbspace:

Metadata area

Metadata identifies key aspects of the sbspace and each smart large object stored in the sbspace, and enables the database server to manipulate and recover smart large objects stored within.

User-data area

User data is the smart large object data stored in the sbspace by user applications. The chunk has up to two user-data areas.

Reserved area

The database server allocates space from the reserved area to either the metadata or user-data area when more space is required. The chunk has up to two reserved areas.

For information about correctly allocating metadata and user data for sbspaces, see Size sbspace metadata and the *IBM® Informix® Performance Guide*.

When you add a chunk to an sbspace, you can specify whether it contains a metadata area and user-data area or whether to reserve the chunk exclusively for user data. You can add more chunks at any time. If you are updating smart large objects, I/O to the user data is much faster on raw disks than cooked chunk files. For instructions on how to create an sbspace, add chunks to an sbspace, or drop a chunk from an sbspace, see Manage disk space.

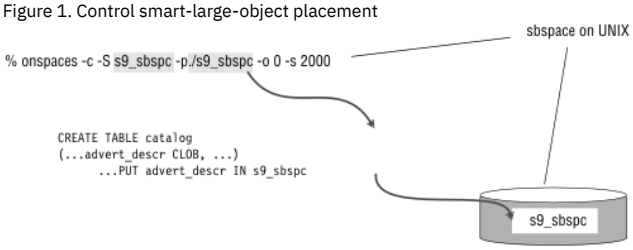
Important: Sbspace metadata is always logged, regardless of the logging setting of the database.

Control of where smart large object data is stored

You specify the data type of a column when you create the table.

For smart large objects, you specify CLOB, BLOB, or user-defined data types. As the following figure shows, to control the placement of smart large objects, you can use the IN *sbspace* option in the PUT clause of the CREATE TABLE statement.

Figure 1. Control smart-large-object placement



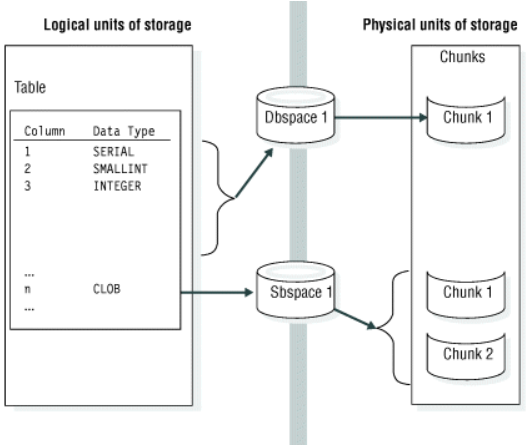
Before you specify an sbspace in a PUT clause, you must first create the sbspace. For more information about how to create an sbspace with the **onspaces -c -S** command, see Adding a chunk to a dbspace or blobspace. For more information about how to specify smart large object characteristics in the PUT clause, see the CREATE TABLE statement in the *IBM® Informix® Guide to SQL: Syntax*.

If you do not specify the PUT clause, the database server stores the smart large objects in the default sbspace that you specify in the SBSPACENAME configuration parameter. For more information about SBSPACENAME, see the configuration parameter topics of the *IBM Informix Administrator's Reference*.

An sbspace includes one or more chunks, as the following figure shows. When an sbspace contains more than one chunk, you cannot specify the chunk in which the data is located.

You can add more chunks at any time. It is a high-priority task of a database server administrator to monitor sbspace chunks for fullness and to anticipate the necessity to allocate more chunks to an sbspace. For more information about monitoring sbspaces, see your *IBM Informix Performance Guide*.

Figure 2. Sbspaces that link logical and physical units of storage



The database server uses sbspaces to store table columns that contain smart large objects. The database server uses dbspaces to store the rest of the table columns.

You can mirror an sbspace to speed recovery in event of a media failure. For more information, see Mirroring.

For information about using **onspaces** to perform the following tasks, see Manage disk space.

- Creating an sbspace
- Adding a chunk to an sbspace
- Altering storage characteristics of smart large objects
- Creating a temporary sbspace
- Dropping an sbspace

Storage characteristics of sbspaces

As the database server administrator, you can use the system default values for these storage characteristics, or you can specify them in the **-Df** tags when you create the sbspace with **onspaces -c**. You can change these sbspace characteristics with the **onspaces -ch** option.

The administrator or programmer can override these default values for storage characteristics and attributes for individual tables.

- Extent sizes for sbspaces
  An extent in an sbspace consists of a collection of contiguous pages that store smart large object data.
- Average smart-large-object size
  Smart large objects usually vary in length. You can provide an average size of your smart large objects to calculate space for an sbspace.
- Buffering mode
  When you create an sbspace, the default buffering mode is on, which means to use the buffer pool in the resident portion of shared memory.
- Last-access time
  When you create an sbspace, you can specify whether the database server must keep the last time that the smart large object was read or updated with the ACCESSTIME tag of the **onspaces -c -Df** option.
- Lock mode
  When you create an sbspace, you can specify whether the database server locks the whole smart large object or a range of bytes within a smart large object with the LOCK_MODE tag of the **onspaces -c -Df** option.
- Logging
  When you create an sbspace, you can specify whether to turn on logging for the smart large objects.

Extent sizes for sbspaces

An extent in an sbspace consists of a collection of contiguous pages that store smart large object data.

The unit of allocation in an sbspace is an extent. The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For example, if an operation asks to write 30 KB, the database server tries to allocate an extent the size of 30 KB.
Important: For most applications, you must use the values that the database server calculates for the extent size.
If you know the size of the smart large object, you can use one of the following functions to set the extent size. The database server allocates the entire smart large object as one extent (if an extent of that size is available in the chunk):

- The DataBlade API **mi_lo_specset_estbytes()** function
  For more information about the DataBlade API functions for smart large objects, see the *IBM® Informix DataBlade API Function Reference*.
- The Informix® ESQL/C **ifx_lo_specset_estbytes** function
  For more information about the Informix ESQL/C functions for smart large objects, see the *IBM Informix ESQL/C Programmer's Manual*.

For information about tuning extent sizes, see smart large objects in the chapter on configuration effects on I/O utilization in your *IBM Informix Performance Guide*.

Average smart-large-object size

Smart large objects usually vary in length. You can provide an average size of your smart large objects to calculate space for an sbspace.

You specify the average size with the AVG_LO_SIZE tag of the **onspaces -c -Df** option.

To specify the size and location of the metadata area, specify the **-Ms** and **-Mo** flags in the **onspaces** command. If you do not use the **-Ms** flag, the database server uses the value of AVG_LO_SIZE to estimate the amount of space to allocate for the metadata area. For more information, see Size sbspace metadata.

Buffering mode

When you create an sbspace, the default buffering mode is on, which means to use the buffer pool in the resident portion of shared memory.

As the database administrator, you can specify the buffering mode with the BUFFERING tag of the **onspaces -c -Df** option. The default is “buffering=ON”, which means to use the buffer pool. If you turn off buffering, the database server uses private buffers in the virtual portion of shared memory.
Important: In general, if read and write operations to the smart large objects are less than 8 KB, do not specify a buffering mode when you create the sbspace. If you are reading or writing short blocks of data, such as 2 KB or 4 KB, leave the default of “buffering=ON” to obtain better performance.
For information about when to use private buffers, see the section on light-weight I/O operations in the topics about configuration effects on I/O utilization in your *IBM® Informix® Performance Guide*.

Last-access time

When you create an sbspace, you can specify whether the database server must keep the last time that the smart large object was read or updated with the ACCESSTIME tag of the **onspaces -c -Df** option.

The default is “ACCESSTIME=OFF”. The database server keeps this last-access time in the metadata area.

For more information about how programmers use this last-access time, see the *IBM® Informix® DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Lock mode

When you create an sbspace, you can specify whether the database server locks the whole smart large object or a range of bytes within a smart large object with the LOCK_MODE tag of the **onspaces -c -Df** option.

The default is “LOCK_MODE=BLOB”, which means to lock the entire smart large object. For more information, see the locking chapter in your *IBM® Informix® Performance Guide*.

Logging

When you create an sbspace, you can specify whether to turn on logging for the smart large objects.

The default is no logging. For more information, see Log sbspaces and smart large objects.
Important: When you use logging databases, turn logging on for the sbspaces. If a failure occurs that requires log recovery, you can keep the smart large objects consistent with the rest of the database.
You specify the logging status with the LOGGING tag of the **onspaces -c -Df** option. The default is “LOGGING=off”. You can change the logging status with the **onspaces -c -Df** option. You can override this logging status with the PUT clause in the SQL statements CREATE TABLE or ALTER TABLE. For more information about these SQL statements, see the *IBM® Informix Guide to SQL: Syntax*.

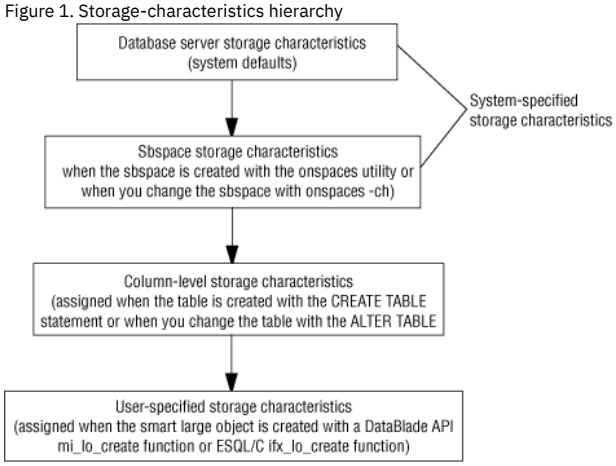
The programmer can override this logging status with functions that the DataBlade API and Informix® ESQL/C provide. For more information about the DataBlade API functions for smart large objects, see the *IBM Informix DataBlade API Function Reference*. For more information about the Informix ESQL/C functions for smart large objects, see the *IBM Informix ESQL/C Programmer's Manual*.

When you turn on logging for an sbspace, the smart large objects pass through the resident portion of shared memory. Although applications can retrieve pieces of a smart large object, you still must consider the larger size of data that might pass through the buffer pool and logical-log buffers. For more information, see Access smart large objects.

Levels of inheritance for sbspace characteristics

The four levels of inheritance for sbspace characteristics are system, sbspace, column, and smart large objects. You can use the system default values for sbspace attributes, or override them for specific sbspaces, columns in a table, or smart large objects.

The following figure shows the storage-characteristics hierarchy for a smart large object.
Figure 1. Storage-characteristics hierarchy



The figure shows that you can override the system default in the following ways:

- Use the **-Df** tags of the **onspaces -c -S** command to override the system default for a specific sbspace.
  You can later change these sbspace attributes for the sbspace with the **onspaces -ch** option. For more information about valid ranges for the **-Df** tags, see the **onspaces** topics in the *IBM® Informix Administrator's Reference*.
- You override the system default for a specific column when you specify these attributes in the PUT clause of the CREATE TABLE or ALTER TABLE statements.

For more information about these SQL statements, see the *IBM Informix Guide to SQL: Syntax*.

- The programmer can override the default values for sbspace attributes for specific smart large objects with functions that the DataBlade API and Informix® ESQL/C programming interface provide.

More information about sbspaces

There are various tasks related to using and managing sbspaces.

The following table lists sources of information tasks related to using and managing sbspaces.

Table 1. Finding information for sbspace tasks

Task	Reference
Setting memory configuration parameters for smart large objects	Manage shared memory
Understanding sbpages	Sbpages
Specifying I/O characteristics for an sbspace	**onspaces** option in Storage characteristics of sbspaces
Allocating space for an sbspace	Creating an sbspace
Adding a chunk to an sbspace	Adding a chunk to an sbspace
Defining or altering storage characteristics for a smart large object	Alter storage characteristics of smart large objects PUT clause of CREATE TABLE or ALTER TABLE statement in *IBM Informix Guide to SQL: Syntax*
Monitoring sbspaces	Monitor sbspaces Topics about table performance considerations in *IBM Informix Performance Guide*
Setting up logging for an sbspace	Log sbspaces and smart large objects
Backing up an sbspace	Back up sbspaces
Checking consistency of an sbspace	Validate metadata
Understanding an sbspace structure	Topics about disk structures in the *IBM Informix Administrator's Reference*
Using **onspaces** for sbspaces	Topics about utilities in the *IBM Informix Administrator's Reference*
Creating a table with CLOB or BLOB data types	*IBM Informix Guide to SQL: Syntax*
Accessing smart large objects in an application	*IBM Informix DataBlade API Programmer's GuideIBM Informix ESQL/C Programmer's Manual*
Calculating the metadata area size Improving metadata I/O Changing storage characteristics	Topics about table performance in *IBM Informix Performance Guide*
Understanding smart-large-object locking	Topics about locking in *IBM Informix Performance Guide*
Configuring sbspaces for temporary smart large objects	Topics about configuration effects on I/O activity in *IBM Informix Performance Guide*

Temporary sbspaces

Use a *temporary sbspace* to store temporary smart large objects without metadata logging and user-data logging.

If you store temporary smart large objects in a standard sbspace, the metadata is logged. Temporary sbspaces are similar to temporary dbspaces. To create a temporary sbspace, use the **onspaces -c -S** command with the **-t** option. For more information, see Creating a temporary sbspace.

You can store temporary large objects in a standard sbspace or temporary sbspace.

- If you specify a temporary sbspace in the SBSPACETEMP parameter, you can store temporary smart large objects there.
- If you specify a standard sbspace in the SBSPACENAME parameter, you can store temporary and permanent smart large objects there.
- If you specify a temporary sbspace name in the CREATE TEMP TABLE statement, you can store temporary smart large objects there.
- If you specify a permanent sbspace name in the CREATE TABLE statement, you can store temporary smart large objects there.
- If you omit the SBSPACETEMP and SBSPACENAME parameters and create a smart large object, error message `-12053` might display.
- If you specify a temporary sbspace in the SBSPACENAME parameter, you cannot store a `permanent` smart large object in that sbspace. You can store temporary smart large objects in that sbspace.

- Comparison of temporary and standard sbspaces
- Temporary smart large objects
  Use *temporary smart large objects* to store text or image data (CLOB or BLOB) that do not require restoring from a backup or log replay in fast recovery.

Comparison of temporary and standard sbspaces

The following table compares standard and temporary sbspaces.

Table 1. Temporary and standard sbspaces

Characteristics	Standard sbspace	Temporary sbspace
Stores smart large objects	Yes	No
Stores temporary smart large objects	Yes	Yes
Logs metadata	Metadata is always logged	Metadata is not logged
Logs user data	User data is not logged for temporary smart large objects but is logged for permanent smart large objects if LOGGING=ON	User data is not logged Creation and deletion of space, and addition of chunks is logged
Fast recovery	Yes	No (the sbspace is emptied when the database server restarts) To set up shared memory without cleaning up temporary smart large objects, specify **oninit -p**. If you keep the temporary large objects, their state is indeterminate.
Backup and restore	Yes	No
Add and drop chunks	Yes	Yes
Configuration parameter	SBSPACENAME	SBSPACETEMP

Temporary smart large objects

Use *temporary smart large objects* to store text or image data (CLOB or BLOB) that do not require restoring from a backup or log replay in fast recovery.

Temporary smart large objects last for the user session and are much faster to update than smart large objects.

You create a temporary smart large object in the same way as a permanent smart large object, except you set the LO_CREATE_TEMP flag in the **ifx_lo_specset_flags** or **mi_lo_specset_flags** function. Use **mi_lo_copy** or **ifx_lo_copy** to create a permanent smart large object from a temporary smart large object. For details on creating temporary smart large objects, see the *IBM® Informix® DataBlade API Programmer's Guide*.
Important: Store pointers to temporary large objects in temporary tables only. If you store them in standard tables and reboot the database server, it results in an error that says that the large object does not exist.
The following table compares standard and temporary smart large objects.

Table 1. Temporary and standard smart large objects

Characteristics	Smart large object	Temporary smart large object
Creation flags	LO_CREATE_LOG or LO_CREATE_NOLOG	LO_CREATE_TEMP
Rollback	Yes	No
Logging	Yes, if turned on	No
Duration	Permanent (until user deletes it)	Deleted at end of user session or transaction
Table type stored in	Permanent or temporary table	Temporary tables

Plogspace

A plogspace is a logical storage unit that is composed of one chunk that stores the physical log. When the physical log is in the plogspace, the database server increases the size of the physical log as needed to improve performance.

If you did not create a server during installation, the physical log is created in the root dbspace. However, you can create the plogspace to move the physical log to a different dbspace to prevent the physical log from filling the root dbspace. For optimal performance, create the plogspace on a different disk from the root dbspace or the location of the logical logs. If you created a server during installation, the plogspace is created automatically with a default size that depends on the value of the AUTO_TUNE_SERVER_SIZE configuration parameter.

By default, the chunk that you assign to the plogspace is extendable, therefore, the initial size of the chunk can be small. The database server automatically expands the chunk when the physical log requires more space.

The plogspace has the following restrictions:

- A database server instance can have only one plogspace.
- The plogspace can contain only the physical log.
- The plogspace can have only one chunk.
- The chunk must have the same page size as the root dbspace.

Related concepts:

Manage the plogspace
Related reference:
Size and location of the physical log
Related information:
AUTO_TUNE_SERVER_SIZE configuration parameter

Extspaces

An *extspace* is a logical name associated with an arbitrary string that signifies the location of external data. The resource that the extspace references depends on a user-defined access method for accessing its contents.

For example, a database user might require access to binary files encoded in a proprietary format. First, a developer creates an *access method*, which is a set of routines that access the data. These routines are responsible for all interaction between the database server and the external file. A DBA then adds an extspace that has the file as its target to the database. After the DBA creates a table in the extspace, the users can access the data in the proprietary files through SQL statements. To locate those files, use the extspace information.

An extspace is not required to be a file name. For example, it can be a network location. The routines that access the data can use information found in the string associated with the extspace in any manner.

For more information about user-defined access methods, see the *IBM® Informix® Virtual-Table Interface Programmer's Guide*. For more information about creating functions and primary access methods, see the *IBM Informix Guide to SQL: Syntax*.

Databases

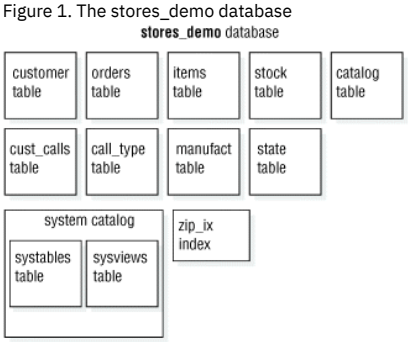
A database is a logical storage unit that contains tables and indexes. Each database also contains a system catalog that tracks information about many of the elements in the database, including tables, indexes, SPL routines, and integrity constraints.

A database is stored in the dbspace that is specified by the IN clause of the CREATE DATABASE statement. When you do not explicitly name a dbspace in the CREATE DATABASE statement, the database is stored in the root dbspace, unless automatic location is enabled. You can enable automatic location by setting the AUTOLOCATE configuration parameter or session environment variable to a positive integer. The database server chooses the dbspaces in which to create new databases and new tables that are created without specified storage locations. Tables are automatically fragmented by round robin in the dbspaces that are chosen by the server.

When you do specify a dbspace in the CREATE DATABASE statement, this dbspace is the location for the following tables:

- Database system catalog tables
- Any table that belongs to the database

The following figure shows the tables that are contained in the **stores_demo** database.
Figure 1. The stores_demo database



The size limits that apply to databases are related to their location in a dbspace. To be certain that all tables in a database are created on a specific physical device, assign only one chunk to the device, and create a dbspace that contains only that chunk. Place your database in that dbspace. When you place a database in a chunk that is assigned to a specific physical device, the database size is limited to the size of that chunk.

Related concepts:
Managing automatic location and fragmentation
Related reference:
Display databases

Tables

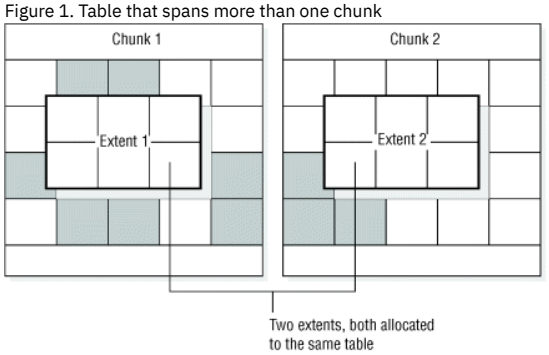
In relational database systems, a *table* is a row of column headings together with zero or more rows of data values. The row of column headings identifies one or more columns and a data type for each column.

When you create a table, the database server allocates disk space for the table in a block of pages that is called an extent. You can specify the size of both the first and any subsequent extents.

You can place the table in a specific dbspace by naming the dbspace when the table is created (with the IN *dbspace* clause of the CREATE TABLE statement). When you do not specify the dbspace, the database server places the table in the dbspace where the database is located. You can fragment a table over more than one dbspace or within a dbspace by specifying a fragmentation distribution scheme. However, if you set the AUTOLOCATE configuration parameter to a positive integer, the database server automatically fragments new tables by round robin, in the dbspaces that are optimal for the table.

A table or table fragment is located completely in the dbspace in which it was created. The database server administrator can use this fact to limit the growth of a table by placing a table in a dbspace and then refusing to add a chunk to the dbspace when it becomes full.

A table, which is composed of extents, can span multiple chunks, as the following figure shows.
Figure 1. Table that spans more than one chunk



Simple large objects are in blobpages in either the dbspace with the data pages of the table or in a separate blobspace.

- Damaged tables
  There are a number of ways you can damage a table.

Related concepts:
Extents
Table fragmentation and data storage
Disk-layout guidelines
Control of where simple large object data is stored

Damaged tables

There are a number of ways you can damage a table.

The following items can damage a table:

- An incorrect buffer flush
- A user error
- Mounting a files system or another chunk on top of a chunk
- Deleting or updating when the scope of the change is not as narrow as you require

Damaged indexes can cause a table to seem damaged, even though it is not.

The **oncheck** commands cannot fix most damaged tables. If a page is damaged, **oncheck** can detect and try to fix the page, but cannot correct the data within the page.

Table types for Informix

You can create logging or nonlogging tables in a logging database on IBM® Informix®. The two table types are STANDARD (logging tables) and RAW (nonlogging tables). The default standard table is like a table created in earlier versions without a special keyword specified. You can create either a STANDARD or RAW table and change tables from one type to another.

In a nonlogging database, both STANDARD tables and RAW tables are nonlogging. In a nonlogging database, the only difference between STANDARD and RAW tables is that RAW tables do not support primary-key constraints, unique constraints, referential constraints, or rollback. However, these tables can be indexed and updated.

The following table lists the properties of the types of tables available with . The flag values are the hexadecimal values for each table type in the **flags** column of **systables**.

Table 1. Table types for

Characteristic	STANDARD	RAW	TEMP
Permanent	Yes	Yes	No
Logged	Yes	No	Yes
Indexes	Yes	Yes	Yes

Characteristic	STANDARD	RAW	TEMP
Constraints	Yes	No referential or unique constraints NULL and NOT NULL constraints are allowed	Yes
Rollback	Yes	No	Yes
Recoverable	Yes	Yes, if not updated	No
Restorable	Yes	Yes, if not updated	No
Loadable	Yes	Yes	Yes
Enterprise Replication servers	Yes	No	No
Primary servers in a high-availability cluster	Yes	Yes, cannot alter logging mode	Yes
Secondary servers in a high-availability cluster	Yes	Yes, but not accessible for any operation	Yes
Flag Value	None	0x10	None

- Standard permanent tables
  A STANDARD table is the same as a table in a logged database that the database server creates.
- RAW tables
  RAW tables are nonlogging permanent tables that are similar to tables in a nonlogging database.
- Temp tables
  Temp tables are temporary, logged tables that are dropped when the user session closes, the database server shuts down, or on reboot after a failure.
- Properties of table types
  These topics explain loading tables, fast recovery, and backup and restore of table types.
- Temporary tables
  The database server must provide disk space for specific types of temporary table.

Standard permanent tables

A STANDARD table is the same as a table in a logged database that the database server creates.

STANDARD tables do not use light appends. All operations are logged, record by record, so STANDARD tables can be recovered and rolled back. You can back up and restore STANDARD tables. Logging enables updates since the last physical backup to be applied when you perform a warm restore or point-in-time restore. Enterprise Replication is allowed on STANDARD tables.

A STANDARD table is the default type on both logging and nonlogging databases. STANDARD tables are logged if stored in a logging database but are not logged if stored in a nonlogging database.

RAW tables

RAW tables are nonlogging permanent tables that are similar to tables in a nonlogging database.

The following statement creates a RAW table called **r_tab**:

```
CREATE RAW TABLE IF NOT EXISTS
   r_tab1 (col1 INT, col2 CHAR(40) NOT NULL);
```

Update, insert, and delete operations on rows in a RAW table are supported but are not logged. You can define indexes on RAW tables, but you cannot define unique constraints, primary-key constraints, or referential constraints on RAW tables.

A RAW table has the same attributes, whether it is stored in a logging database or in a nonlogging database. If you update a RAW table, you cannot reliably restore the data unless you perform a level-0 backup after the update. If the table has not been updated since that backup, you can restore the RAW table from the last physical backup, but backing up only the logical logs is not sufficient for a RAW table to be recoverable. Fast recovery can roll back incomplete transactions on STANDARD tables but not on RAW tables. For information about creating and altering RAW tables, see the *IBM Informix Guide to SQL: Syntax*.

RAW tables are intended for the initial loading and validation of data. To load RAW tables, you can use any loading utility, including **dbexport**, the LOAD statement of DB-Access, or the HPL in express mode. If an error or failure occurs while loading a RAW table, the resulting data is whatever was on the disk at the time of the failure.

Restriction: Do not use RAW tables within a transaction. After you have loaded the data, use the ALTER TABLE statement to change the table to type STANDARD and perform a level-0 backup before you use the table in a transaction.
Restriction: Do not use Enterprise Replication on RAW or TEMP tables.
There are some restrictions when using RAW tables in a high-availability cluster environment. Because modifications made to RAW tables are not logged, and because secondary servers (including HDR, RSS and SDS) use log records to stay synchronized with the primary server, you are restricted from performing certain operations on RAW tables:

- On a primary server, RAW tables can be created, dropped, and accessed. Altering the table mode, however, from unlogged to logged, or from logged to unlogged, is not allowed. Altering the logging mode of a table in a high-availability cluster environment yields error -19845.
- On secondary servers (HDR, SDS, or RSS), RAW tables are not accessible for any operation. Attempting to access a RAW table from SQL yields error -19846.

Temp tables

Temp tables are temporary, logged tables that are dropped when the user session closes, the database server shuts down, or on reboot after a failure.

Temp tables support indexes, constraints, and rollback. You cannot recover, back up, or restore temp tables. Temp tables support bulk operations such as light appends, which add rows quickly to the end of each table fragment. For more information about light appends, see your *IBM Informix Performance Guide*.

For more information, see Temporary tables.

Properties of table types

These topics explain loading tables, fast recovery, and backup and restore of table types.

- Loading of data into a table
  You can use the CREATE RAW TABLE statement to create a RAW table or use the ALTER TABLE statement to change a STANDARD table to RAW before loading the table. After you load the table, run UPDATE STATISTICS on it.
- Fast recovery of table types
  There are fast recovery scenarios for the table types available with IBM Informix.
- Backup and restore of RAW tables
  There are backup scenarios for the table types available on IBM Informix.

Loading of data into a table

You can use the CREATE RAW TABLE statement to create a RAW table or use the ALTER TABLE statement to change a STANDARD table to RAW before loading the table. After you load the table, run UPDATE STATISTICS on it.

IBM® Informix® creates STANDARD tables that use logging by default. Data warehousing applications can have huge tables that take a long time to load. Nonlogging tables are faster to load than logging tables.

For more information about how to improve the performance of loading very large tables, see your *IBM Informix Performance Guide*. For more information about using ALTER TABLE to change a table from logging to nonlogging, see the *IBM Informix Guide to SQL: Syntax*.

Fast recovery of table types

There are fast recovery scenarios for the table types available with IBM® Informix®.

The following table shows fast recovery scenarios for table types.

Table 1. Fast recovery of table types

Table type	Fast recovery behavior
Standard	Fast recovery is successful. All committed log records are rolled forward, and all incomplete transactions are rolled back.
RAW	If a checkpoint completed since the raw table was modified last, all the data is recoverable. Inserts, updates, and deletions that occurred after the last checkpoint are lost. Incomplete transactions in a RAW table are not rolled back.

Backup and restore of RAW tables

There are backup scenarios for the table types available on IBM® Informix®.

The following table explains backup scenarios for table types.

Table 1. Backing up tables on

Table type	Backup allowed?
Standard	Yes.
Temp	No.

Table type	Backup allowed?
RAW	Yes. If you update a RAW table, you must back it up so that you can restore all the data in it. Backing up only the logical logs is not enough.

Important: After you load a RAW table or change a RAW table to type STANDARD, you must perform a level-0 backup.
The following table shows restore scenarios for these table types.

Table 2. Restoring tables on

Table type	Restore allowed?
Standard	Yes. Warm restore, cold restore, and point-in-time restore work.
Temp	No.
RAW	When you restore a RAW table, it contains only data that was on disk at the time of the last backup. Because RAW tables are not logged, any changes that occurred since the last backup are not restored.

Temporary tables

The database server must provide disk space for specific types of temporary table.

Disk space is required for the following temporary tables:

- Temporary tables that you create with an SQL statement, such as CREATE TEMP TABLE. . .
- Temporary tables that the database server creates as it processes a query

Make sure that your database server has configured enough temporary space for both user-created and database server-created temporary tables. Some uses of the database server might require as much temporary storage space as permanent storage space, or more.

By default, the database server stores temporary tables in the root dbspace. If you decide not to store your temporary tables in the root dbspace, use the DBSPACETEMP environment variable or the DBSPACETEMP configuration parameter to specify a list of dbspaces for temporary tables.

- Temporary tables that you create
  You can create temporary tables with some SQL statements.
- Temporary tables that the database server creates
  The database server sometimes creates temporary tables while running queries against the database or backing it up.

Temporary tables that you create

You can create temporary tables with some SQL statements.

Temporary tables can be created with the following SQL statements:

- TEMP TABLE option of the CREATE TABLE statement
- INTO TEMP clause of the SELECT statement, such as `SELECT * FROM customer INTO TEMP cust_temp`

Only the session that creates a temporary table can use the table. When the session exits, the table is dropped automatically.

When you create a temporary table, the database server uses the following criteria:

- If the query used to populate the TEMP table produces no rows, the database server creates an empty, unfragmented table.
- If the rows that the query produces do not exceed 8 KB, the temporary table is located in only one dbspace.
- If the rows exceed 8 KB, the database server creates multiple fragments and uses a round-robin fragmentation scheme to populate them unless you specify a fragmentation method and location for the table.

If you use the CREATE TEMP and SELECT...INTO TEMP SQL statements and DBSPACETEMP has been set:

- LOGGING dbspaces in the list are used to create the tables that specify or imply the WITH LOG clause.
- NON-LOGGING temporary dbspaces in the list are used to create the tables that specify the WITH NO LOG clause.

When CREATE TEMP and SELECT...INTO TEMP SQL statements are used and DBSPACETEMP has not been set or does not contain the correct type of dbspace, IBM® Informix® uses the dbspace of the database to store the temporary table. See the *IBM Informix Guide to SQL: Syntax* for more information.

- Where user-created temporary tables are stored
  If your application lets you specify the location of a temporary table, you can specify either logging spaces or nonlogging spaces that you create exclusively for temporary tables.

Where user-created temporary tables are stored

If your application lets you specify the location of a temporary table, you can specify either logging spaces or nonlogging spaces that you create exclusively for temporary tables.

For information about creating temporary dbspaces, see the **onspaces** topics in the *IBM® Informix® Administrator's Reference*.

If you do not specify the location of a temporary table, the database server stores the temporary table in one of the spaces that you specify as an argument to the DBSPACETEMP configuration parameter or environment variable. The database server remembers the name of the last dbspace that it used for a temporary table. When the database server receives another request for temporary storage space, it uses the next available dbspace to spread I/O evenly across the temporary storage space.

For information about where the database stores temporary tables when you do not list any spaces as an argument to DBSPACETEMP, see the DBSPACETEMP section in the *IBM Informix Administrator's Reference*.

When you use an application to create a temporary table, you can use the temporary table until the application exits or performs one of the following actions:

- Closes the database in which the table was created and opens a database in a different database server
- Closes the database in which the table was created
- Explicitly drops the temporary table

Temporary tables that the database server creates

The database server sometimes creates temporary tables while running queries against the database or backing it up.

The database server might create a temporary table in any of the following circumstances:

- Statements that include a GROUP BY or ORDER BY clause
- Statements that use aggregate functions with the UNIQUE or DISTINCT keywords
- SELECT statements that use auto-index or hash joins
- Complex CREATE VIEW statements
- DECLARE statements that create a scroll cursor
- Statements that contain correlated subqueries
- Statements that contain subqueries that occur within an IN or ANY clause
- CREATE INDEX statements

When the process that initiated the creation of the table is complete, the database server deletes the temporary tables that it creates.

If the database server shuts down without removing temporary tables, the database server removes the temporary tables the next time it is started. To start the database server without removing temporary tables, run the **oninit** command with the **-p** option.

Applications and analytic tools can define queries in which a derived table contains multiple views joined with base tables, potentially including hundreds of columns. The database server attempts to fold views or derived tables into the main query. Any such views or derived tables that cannot be folded are materialized into a temporary table. The temporary table excludes all the columns that are not referenced in the main query. The temporary table is created with only the columns referenced in the Projection clause and in other clauses of the parent query, including the WHERE, HAVING, GROUP BY, and ON clauses.

By excluding from the system-generated temporary table any columns that are not referenced in the main query, this reduced schema can improve query performance by conserving storage resources, and by avoiding unnecessary I/O of data in the unused columns.

In a nested query, however, projected columns from views and derived table are checked only in the parent query, but not in the levels above the immediate parent query.

Important: In addition to temporary tables, the database server uses temporary disk space to store the before images of data records that are overwritten while backups are occurring, and for overflow from query processing that occurs in memory. Make sure that you have correctly set the **DBSPACETEMP** environment variable or the DBSPACETEMP configuration parameter to specify dbspaces with enough space for your needs. If there is not enough room in the specified dbspaces, the backup fails, root dbspace is used, or the backup fails after filling the root dbspace.

- Where database server-created temporary tables are stored
  When the database server creates a temporary table, it stores the temporary table in one of the dbspaces that you specify in the DBSPACETEMP configuration parameter or the DBSPACETEMP environment variable. The environment variable supersedes the configuration parameter.

Where database server-created temporary tables are stored

When the database server creates a temporary table, it stores the temporary table in one of the dbspaces that you specify in the DBSPACETEMP configuration parameter or the DBSPACETEMP environment variable. The environment variable supersedes the configuration parameter.

When you do not specify any temporary dbspaces in DBSPACETEMP, or the temporary dbspaces that you specify have insufficient space, the database server creates the table in a standard dbspace according to the following rules:

- If you created the temporary table with CREATE TEMP TABLE, the database server stores this table in the dbspace that contains the database to which the table belongs.
- If you created the temporary table with the INTO TEMP option of the SELECT statement, the database server stores this table in the root dbspace.

For more information, see Creating a temporary dbspace.

Tblspaces

Database server administrators sometimes must track disk use by a particular table. A *tblspace* contains all the disk space allocated to a given table or table fragment (if the table is fragmented). A separate tblspace contains the disk space allocated for the associated index.

A tblspace, for example, does not correspond to any particular part of a chunk or even to any particular chunk. The indexes and data that make up a tblspace might be scattered throughout your chunks. The tblspace, however, represents a convenient accounting entity for space across chunks devoted to a particular table. (See Tables.)

- Maximum number of tblspaces in a table
  You can specify a maximum of 2**20 (or 1,048,576) tblspaces in a table.
- Table and index tblspaces
  The table tblspace and index tblspace contain certain types of pages.
- Extent interleaving
  The database server allocates the pages that belong to a tblspace as extents. Although the pages within an extent are contiguous, extents might be scattered throughout the dbspace where the table is located (even on different chunks).

Maximum number of tblspaces in a table

You can specify a maximum of 2**20 (or 1,048,576) tblspaces in a table.

Table and index tblspaces

The table tblspace and index tblspace contain certain types of pages.

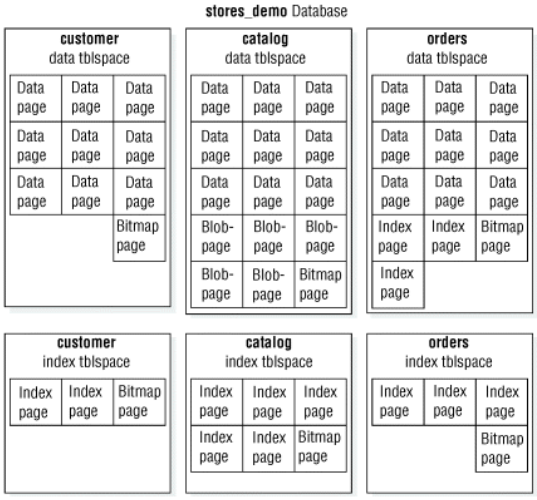
The table tblspace contains the following types of pages:

- Pages allocated to data
- Pages allocated to indexes
- Pages used to store TEXT or BYTE data in the dbspace (but not pages used to store TEXT or BYTE data in a blobspace)
- Bitmap pages that track page use within the table extents

The index tblspace contains the following types of pages:

- Pages allocated to indexes
- Bitmap pages that track page use within the index extents

The following table illustrates the tblspaces for three tables that form part of the **stores_demo** database. Only one table (or table fragment) exists per tblspace. Blobpages represent TEXT or BYTE data stored in a dbspace.
Figure 1. Sample tblspaces in the stores_demo database



Extent interleaving

The database server allocates the pages that belong to a tblspace as extents. Although the pages within an extent are contiguous, extents might be scattered throughout the dbspace where the table is located (even on different chunks).

The following figure depicts this situation with two noncontiguous extents that belong to the tblspace for **table_1** and a third extent that belongs to the tblspace for **table_2**. A **table_2** extent is located between the first table_1 extent and the second **table_1** extent. When this situation occurs, the extents are interleaved. Because sequential access searches across **table_1** require the disk head to seek across the **table_2** extent, performance is worse than if the **table_1** extents were contiguous. For instructions on how to avoid and eliminate interleaving extents, see your *IBM® Informix® Performance Guide*.

Figure 1. Three extents that belong to two different tblspaces in a single dbspace

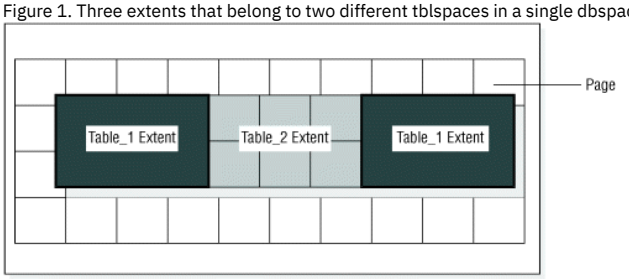


Table fragmentation and data storage

The fragmentation feature gives you more control over where the database stores data. You are not limited to specifying the locations of individual tables and indexes. You can also specify the location of table and index *fragments*, which are different parts of a table or index that are on different storage spaces.

You can fragment a table in the following ways:

- Fragment a table over more than one dbspace. However, you cannot put fragments into dbspaces that have different page sizes. All fragments must have the same page size.
- Create multiple partitions of a fragmented table within a single dbspace if the fragmented table uses an expression-based or round-robin distribution scheme.

You can fragment the following storage spaces:

- Dbspaces
- Sbspaces

Usually you fragment a table when you initially create it. The CREATE TABLE statement takes one of the following forms:

```
CREATE TABLE tablename ... FRAGMENT BY ROUND ROBIN IN dbspace1,
 dbspace2, dbspace3;

CREATE TABLE tablename ...FRAGMENT BY EXPRESSION
   <Expression 1> in dbspace1,
   <Expression 2> in dbspace2,
   <Expression 3> in dbspace3;
```

The FRAGMENT BY ROUND ROBIN and FRAGMENT BY EXPRESSION keywords refer to two different distribution schemes. Both statements associate fragments with dbspaces.

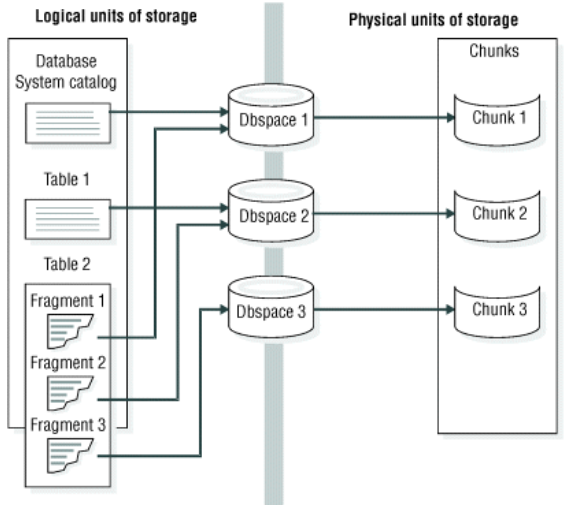
If you set the AUTOLOCATE configuration parameter or session environment variable to a positive integer, and you do not specify a location for the table, new tables are fragmented in round-robin order in dbspaces that are chosen by the database server.

When you fragment a table, you can also create multiple partitions of the table within the same dbspace, as shown in this example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
         PARTITION part1 (a >=0 AND a < 5) in dbs1,
         PARTITION part2 (a >=5 AND a < 10) in dbs1
           ...
       ;
```

The following figure illustrates the role of fragments in specifying the location of data.

Figure 1. Dbspaces that link logical units (including table fragments) and physical units of storage



Related concepts:
Manage disk space
Managing automatic location and fragmentation
Tables
Related information:
Table fragmentation strategies
Fragmentation guidelines

Amount of disk space needed to store data

To determine how much disk space you require, follow these steps:

1. Calculate the size requirements of the root dbspace.
2. Estimate the total amount of disk space to allocate to all the database server databases, including space for overhead and growth.

The following topics explain these steps.

- Size of the root dbspace
  You can calculate the size of the root dbspace, which stores information that describes your database server.
- Amount of space that databases require
  The amount of additional disk space required for the database server data storage depends on the requirements of users, plus overhead and growth.

Size of the root dbspace

You can calculate the size of the root dbspace, which stores information that describes your database server.

The following storage structures can be stored in the root dbspace:

Physical and logical logs (200 KB minimum for each type)
: Although the root dbspace is the default location for the physical log and logical log files, move the log files to other dbspaces. You can set the AUTO_LLOG configuration parameter to specify the dbspace for logical log files. You can store the physical log in a plogspace.
Recommendation: Set up the system with a small physical log and a few small logical logs. For example, create three 1000 KB logical log files, or 3000 KB for the total log space. After the initial setup is complete, create a new dbspace for logical logs in an area that does not compete for I/O with other dbspaces, and set the AUTO_LLOG configuration parameter to that dbspace. Create a set of larger logical-log files in the dbspace for logical logs, and drop the original logs from the root dbspace. Then create a plogspace for the physical log. Make the plogspace large enough to hold your final physical log, and isolate it from other dbspaces as much as possible. This configuration optimizes logging performance and the root dbspace for the following reasons:

  - The unused space that is left in the root dbspace after you move the logs is minimized.
  - The physical and logical logs do not contend for space and I/O on the same disk as each other or the root dbspace.
  - The server automatically increases the total logical log space and the size of the physical log if increasing logs measurably improves performance.

Temporary tables
: Analyze user applications to estimate the amount of disk space that the database server might require for temporary tables. Try to estimate how many of these statements are to run concurrently. The space that is occupied by the rows and columns that are returned provides a good basis for estimating the amount of space required. The largest temporary table that the database server creates during a warm restore is equal to the size of your logical log. You calculate the size of your logical log by adding the sizes of all logical-log files. You must also analyze user applications to estimate the amount of disk space that the database server might require for explicit temporary tables.

Data
: Although the root dbspace is the default location for databases, do not store databases and tables in the root dbspace.

System databases (the size varies between versions)

The **sysmaster**, **sysutils**, **syscdr**, and **sysuuid** databases, and the system catalogs must be stored in the root dbspace. The **sysadmin** database is stored in the root dbspace by default, however, you can move the **sysadmin** database to a different dbspace.

Reserved pages (~24 KB)

The reserved pages contain control and tracking information that is used by the database server. Reserved pages must be stored in the root dbspace.

Tblspace tblspace (100 - 200 KB minimum)

The **tblspace** tblspace contains information about tblspaces. The **tblspace** tblspace must be stored in the root dbspace.

This estimate is the root dbspace size before you initialize the database server. The size of the root dbspace depends on whether you plan to store the physical log, logical logs, and temporary tables in the root dbspace or in another dbspace. The root dbspace must be large enough for the minimum size configuration during disk initialization.

Allow extra space in the root dbspace for the system databases to grow, for the extended reserved pages, and ample free space. The number of extended reserved pages depends on the number of primary chunks, mirror chunks, logical-log files, and storage spaces in the database server.

If you need to make the root dbspace larger after the server is initialized, you can add a chunk to the root dbspace. You can enable automatic space management to expand the root dbspace as needed.

Important: Mirror the root dbspace and other dbspaces that contain critical data such as the physical log and logical logs.

Related concepts:
Automatic space management
Manage dbspaces
Move logical-log files
Related tasks:
Creating a temporary dbspace
Related reference:
Change the physical-log location and size
Related information:
ROOTSIZE configuration parameter
reset sysadmin argument: Move the sysadmin database (SQL administration API)

Amount of space that databases require

The amount of additional disk space required for the database server data storage depends on the requirements of users, plus overhead and growth.

Every application that users run has different storage requirements. The following list suggests some of the steps that you can take to calculate the amount of disk space to allocate (beyond the root dbspace):

- Decide how many databases and tables you must to store. Calculate the amount of space required for each one.
- Calculate a growth rate for each table and assign some amount of disk space to each table to accommodate growth.
- Decide which databases and tables you want to mirror.

For instructions about calculating the size of your tables, see your *IBM® Informix® Performance Guide*.

The storage pool

Every instance of the database server has a storage pool. The storage pool contains information about the directories, cooked files, and raw devices that the server can use if necessary to automatically expand an existing dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

When the storage space falls below a threshold defined in the SP_THRESHOLD configuration parameter, the database server can automatically run a task that expands the space, either by extending an existing chunk in the space or by adding a new chunk.

You can use SQL administration API commands to:

- Add, delete, or modify an entry that describes one directory, cooked file, or raw device in the storage pool. The server can use the specified directory, cooked file, or raw device when necessary to automatically add space to an existing storage space.
- Control how a storage pool entry is used by modifying two different dbspace sizes that are associated with expanding a storage space, the extend size and the create size.
- Mark a chunk as extendable or not extendable.
- Immediately expand the size of a space, when you do not want the database server to automatically expand the space.
- Immediately extend the size of a chunk by a specified minimum amount.
- Create a storage space or chunk from an entry in the storage pool
- Return empty space from a dropped storage space or chunk to the storage pool

The **storagepool** table in **sysadmin** database contains information about all of the entries in a storage pool for a database server instance.

Related concepts:
Extendable chunks
Automatic space management
Related tasks:
Creating and managing storage pool entries

Disk-layout guidelines

The following goals are typical for efficient disk layout:

- Limiting disk-head movement
- Reducing disk contention
- Balancing the load
- Maximizing availability

You must make some trade-offs among these goals when you design your disk layout. For example, separating the system catalog tables, the logical log, and the physical log can help reduce contention for these resources. However, this action can also increase the chances that you must perform a system restore. For detailed disk-layout guidelines, see the *IBM® Informix® Performance Guide*.

- Dbspace and chunk guidelines
  This topic lists some general strategies for disk layout that do not require any information about the characteristics of a particular database.
- Table-location guidelines
  This topic lists some strategies for optimizing the disk layout, given certain characteristics about the tables in a database.

Related concepts:
Partitions and offsets
Tables

Dbspace and chunk guidelines

This topic lists some general strategies for disk layout that do not require any information about the characteristics of a particular database.

- Associate disk partitions with chunks and allocate at least one additional chunk for the root dbspace.
  A disk that is already partitioned might require the use of offsets. For details, see Allocating raw disk space on UNIX.
  Tip: With the 4-terabyte maximum size of a chunk, you can avoid partitioning by assigning a chunk per disk drive.
- Mirror critical dbspaces: the root dbspace, the dbspaces that contain the physical log and the logical-log files. Also mirror high-use databases and tables.
  You specify mirroring at the dbspace level. Mirroring is either on or off for all chunks belonging to a dbspace. Locate the primary and the mirrored dbspaces on different disks. Ideally, different controllers handle the different disks.
- Spread temporary tables and sort files across multiple disks.
  To define several dbspaces for temporary tables and sort files, use **onspaces -t**. When you place these dbspaces on different disks and list them in the DBSPACETEMP configuration parameter, you can spread the I/O associated with temporary tables and sort files across multiple disks. For information about using the DBSPACETEMP configuration parameter or environment variable, see the chapter on configuration parameters in the *IBM® Informix® Administrator's Reference*.
- Keep the physical log in the root dbspace but move the logical logs from the root dbspace. However, if you plan to store the system catalogs in the root dbspace, move the physical log to another dbspace.
  For advice on where to store your logs, see Location of logical-log files. Also see Move logical-log files and Change the physical-log location and size.
- To improve backup and restore performance:
  - Cluster system catalogs with the data that they track.
  - If you use ON-Bar to perform parallel backups to a high-speed tape drive, store the databases in several small dbspaces.
    For additional performance recommendations, see the *IBM Informix Backup and Restore Guide*.

Table-location guidelines

This topic lists some strategies for optimizing the disk layout, given certain characteristics about the tables in a database.

You can implement many of these strategies with a higher degree of control using table fragmentation:

- Isolate high-use tables on a separate disk.
  To isolate a high-use table on its own disk device, assign the device to a chunk, and assign the same chunk to a dbspace. Finally, place the frequently used table in the dbspace just created using the IN *dbspace* option of CREATE TABLE.

  To display the level of I/O operations against each chunk, run the **onstat -g iof** option.
- Fragment high-use tables over multiple disks.
- Group related tables in a dbspace.
  If a device that contains a dbspace fails, all tables in that dbspace are inaccessible. However, tables in other dbspaces remain accessible. Although you must perform a cold restore if a dbspace that contains critical information fails, you must only perform a warm restore if a noncritical dbspace fails.
- Place high-use tables on the middle partition of a disk.
- Optimize table extent sizes.

For more information, see the chapter on table performance considerations in your *IBM® Informix® Performance Guide*. For information about **onstat** options, see the *IBM Informix Administrator's Reference*.

Sample disk layouts

When setting out to organize disk space, the database server administrator usually has one or more of the following objectives in mind:

- High performance
- High availability
- Ease and frequency of backup and restore

Meeting any one of these objectives has trade-offs. For example, configuring your system for high performance usually results in taking risks regarding the availability of data. The sections that follow present an example in which the database server administrator must make disk-layout choices given limited disk resources. These sections describe two different disk-layout solutions. The first solution represents a performance optimization, and the second solution represents an availability-and-restore optimization.

The setting for the sample disk layouts is a fictitious sporting goods database that uses the structure (but not the volume) of the **stores_demo** database. In this example, the database server is configured to handle approximately 350 users and 3 gigabytes of data. The disk space resources are shown in the following table.

Disk drive	Size of drive	High performance
Disk 1	2.5 gigabytes	No
Disk 2	3 gigabytes	Yes
Disk 3	2 gigabytes	Yes
Disk 4	1.5 gigabytes	No

The database includes two large tables: **cust_calls** and **items**. Assume that both of these tables contain more than 1,000,000 rows. The **cust_calls** table represents a record of all customer calls made to the distributor. The items table contains a line item of every order that the distributor ever shipped.

The database includes two high-use tables: **items** and **orders**. Both of these tables are subject to constant access from users around the country.

The remaining tables are low-volume tables that the database server uses to look up data such as postal code or manufacturer.

Table name	Maximum size	Access rate
cust_calls	2.5 gigabytes	Low
items	0.5 gigabytes	High
orders	50 megabytes	High
customers	50 megabytes	Low
stock	50 megabytes	Low
catalog	50 megabytes	Low
manufact	50 megabytes	Low
state	50 megabytes	Low
call_type	50 megabytes	Low

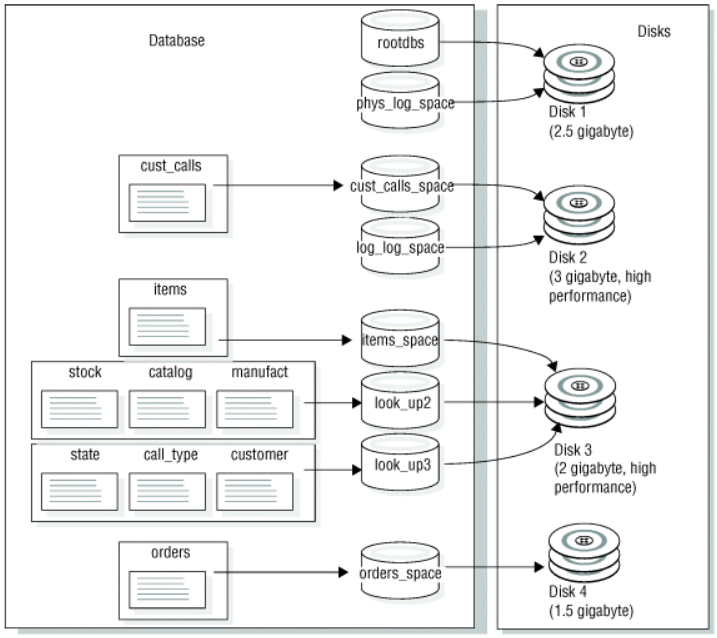
Sample layout when performance is highest priority

To optimize performance, use multiple storage spaces and multiple disks. The following figure shows a disk layout that is optimized for performance. This disk layout uses the following strategies to improve performance:

- Migration of the logical log and physical log files from the root dbspace
  This strategy separates the logical log and the physical log and reduces contention for the root dbspace. For best performance, take advantage of automatic performance tuning for the logical and physical logs:
  - Create a plogspace to enable the automatic expansion of the physical log.
  - Set the AUTO_LLOG configuration parameter to enable the automatic expansion of the logical log in a specified dbspace.

  If you create a server during installation, the plogspace is created and the AUTO_LLOG configuration parameter is set to a non-critical dbspace.
- Location of the two tables that undergo the highest use in dbspaces on separate disks
  Neither of these disks stores the logical log or the physical log. Ideally you might store each of the **items** and **orders** tables on a separate high-performance disk. However, in the present scenario, this strategy is not possible because one of the high-performance disks is required to store the large **cust_calls** table (the other two disks are too small for this task).

Figure 1. Disk layout optimized for performance

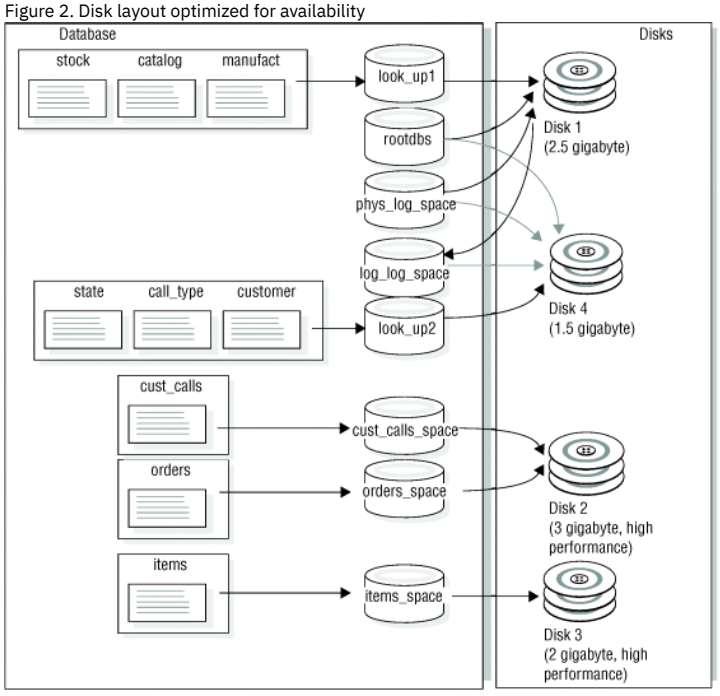


Sample layout when availability is highest priority

The weakness of the previous disk layout is that if either Disk 1 or Disk 2 fails, the whole database server goes down until you restore the dbspaces on these disks from backups. In other words, the disk layout is poor with respect to availability.

An alternative disk layout that optimizes for availability and involves mirroring is shown in following figure. This layout mirrors all the critical data spaces (the system catalog tables, the physical log, and the logical log) to a separate disk. Ideally you might separate the logical log and physical log (as in the previous layout) and mirror each disk to its own mirror disk. However, in this scenario, the required number of disks does not exist; therefore, the logical log and the physical log both are located in the root dbspace.

Figure 2. Disk layout optimized for availability



Logical-volume manager

You can use the logical-volume manager (LVM) utility to manage your disk space through user-defined logical volumes.

Many computer manufacturers ship their computers with a proprietary LVM. You can use the database server to store and retrieve data on disks that are managed by most proprietary LVMs. Logical-volume managers provide some advantages and some disadvantages, as explained in the remainder of this section.

Most LVMs can manage multiple gigabytes of disk space. The database server chunks are limited to a size of 4 terabytes, and this size can be attained only when the chunk being allocated has an offset of zero. Consequently, you must limit the size of any volumes to be allocated as chunks to a size of 4 terabytes.

Because you can use LVMs to partition a disk drive into multiple volumes, you can control where data is placed on a given disk. You can improve performance by defining a volume that consists of the middle-most cylinders of a disk drive and placing high-use tables in that volume. (Technically, you do not place a table directly in a volume. You must first allocate a chunk as a volume, then assign the chunk to a dbspace, and finally place the table in the dbspace. For more information, see Control of where simple large object data is stored.)
Tip: If you choose to use large disk drives, you can assign a chunk to one drive and eliminate the necessity to partition the disk.
You can also improve performance by using a logical volume manager to define a volume that spreads across multiple disks and then placing a table in that volume.

Many logical volume managers also allow a degree of flexibility that standard operating-system format utilities do not. One such feature is the ability to reposition logical volumes after you define them. Thus getting the layout of your disk space right the first time is not so critical as with operating-system format utilities.

LVMs often provide operating-system-level mirroring facilities. For more information, see Alternatives to mirroring.

Manage disk space

You can use several utilities and tools to manage disk spaces and the data that the database server controls.

You can use the following utilities to manage storage spaces:

- The **onspaces** utility commands
- SQL administration API commands

Your *IBM® Informix Performance Guide* also contains information about managing disk space. In particular, it describes how to eliminate interleaved extents, how to reclaim space in an empty extent, and how to improve disk I/O.

You can generate SQL administration API or **onspaces** commands for reproducing the storage spaces, chunks, and logs that exist in a file with the **dbschema** utility.

- Allocate disk space
  This section explains how to allocate disk space for the database server.
- Specify names for storage spaces and chunks
  Chunk names follow the same rules as storage-space names.
- Monitor storage spaces
  You can monitor the status of storage spaces and configure how you are notified when a storage space becomes full.
- Manage dbspaces
  This section contains information about creating standard and temporary dbspaces with and without the default page size, specifying the first and next extent sizes for the tblspace **tblspace** in a dbspace when you create the dbspace, and adding a chunk to a dbspace or blobspace.
- Manage blobspaces
  This section explains how to create a blobspace and determine the blobpage size.
- Manage sbspaces
  This section describes how to create a standard or temporary sbspace, monitor the metadata and user-data areas, add a chunk to an sbspace, and alter storage characteristics of smart large objects.
- Manage the plogspace
  You create or move the plogspace with the **onspaces** utility or equivalent SQL administration API command.
- Automatic space management
  You can configure the server to add more storage space automatically when more space is required. You use space more effectively and ensure that space is allocated as necessary, while reducing out-of-space errors. You reduce the time required to manually monitor your spaces to determine which storage space might run out of free space. If you configure the server to automatically add space, you can also manually expand a space or extend a chunk.
- Drop a chunk
  Use the **onspaces** utility to drop a chunk from a dbspace.
- Drop a storage space
  Use **onspaces** to drop a dbspace, temporary dbspace, blobspace, sbspace, temporary sbspace, or extspace.
- Creating a space or chunk from the storage pool
  If your storage pool contains entries, you can create storage spaces or chunks from free space in the storage pool.
- Returning empty space to the storage pool
  You can return the space from an empty chunk or storage space to the storage pool.
- Manage extspaces
- Skip inaccessible fragments
- Display databases
  You can display the databases that you create with SMI tables.
- Monitor disk usage
- Multitenancy
  You can segregate data, storage space, and processing resources for multiple client organizations by creating multiple tenant databases in a single instance of Informix®.
- Storage optimization
  Data compression and consolidation processes can minimize the disk space that is used by your data and indexes.
- Load data into a table

Related concepts:
Table fragmentation and data storage
Storage space creation and management
Data storage
Related information:
Storage space, chunk, and log creation
SQL Administration API Functions
Managing extents
Managing sbspaces

Allocate disk space

This section explains how to allocate disk space for the database server.

Read the following sections before you allocate disk space:

- Unbuffered or buffered disk access on UNIX
- Amount of disk space needed to store data
- Disk-layout guidelines

Before you can create a storage space or chunk, or mirror an existing storage space, you must allocate disk space for the chunk file. You can allocate either an empty file or a portion of raw disk for database server disk space.

UNIX only: On UNIX, if you allocate raw disk space, you must use the UNIX **ln** command to create a link between the character-special device name and another file name. For more information about this topic, see Create symbolic links to raw devices (UNIX).
Using a UNIX file and its inherent operating-system interface for database server disk space is called using *cooked space*.

Windows only: On Windows, you must use NTFS files for database server disk space. For more information about this recommendation, see Unbuffered or buffered disk access on UNIX.
You can balance chunks over disks and controllers. Placing multiple chunks on a single disk can improve throughput.

- Specify an offset
  When you allocate a chunk of disk space to the database server, specify an offset.
- Allocating cooked file spaces on UNIX
  The following procedure shows an example of allocating disk space for a cooked file.
- Allocating raw disk space on UNIX
  To allocate raw space, you must have a disk partition available that is dedicated to raw space. To create raw disk space, you can either repartition your disks or unmount an existing file system. Back up any files before you unmount the device.
- Create symbolic links to raw devices (UNIX)
  Use symbolic links to assign standard device names and to point to the device.
- Allocating NTFS file space on Windows
  On Windows, the database server uses NTFS files by default. You can use standard file names for unbuffered files in the NTFS file system. If all your partitions are FAT files, you can convert one to NTFS.
- Allocating raw disk space on Windows
  You can configure raw disk space on Windows as a logical drive or physical drive.

Specify an offset

When you allocate a chunk of disk space to the database server, specify an offset.

Specify an offset for one of the following two purposes:

- To prevent the database server from overwriting the partition information
- To define multiple chunks on a partition, disk device, or cooked file

The maximum value for the offset is 4 terabytes.

Many computer systems and some disk-drive manufacturers keep information for a physical disk drive on the drive itself. This information is sometimes called a *volume table of contents* (VTOC) or disk label. The VTOC is commonly stored on the first track of the drive. A table of alternative sectors and bad-sector mappings (also called a *revectoring table*) might also be stored on the first track.

If you plan to allocate partitions at the start of a disk, you might be required to use offsets to prevent the database server from overwriting critical information required by the operating system. For the exact offset required, see your disk-drive manuals.
Important: If you are running two or more instances of the database server, be extremely careful not to define chunks that overlap. Overlapping chunks can cause the database server to overwrite data in one chunk with unrelated data from an overlapping chunk. This overwrite effectively deletes overlapping data.

- Specify an offset for the initial chunk of root dbspace
  For the initial chunk of root dbspace and its mirror, if it has one, specify the offsets with the ROOTOFFSET and MIRROROFFSET parameters, respectively.
- Specify an offset for additional chunks
  To specify an offset for additional chunks of database server space, you must supply the offset as a parameter when you assign the space to the database server.
- Use offsets to create multiple chunks
  You can create multiple chunks from a disk partition, disk device, or file, by specifying offsets and assigning chunks that are smaller than the total space available.

Specify an offset for the initial chunk of root dbspace

For the initial chunk of root dbspace and its mirror, if it has one, specify the offsets with the ROOTOFFSET and MIRROROFFSET parameters, respectively.

For more information, see the topics about configuration parameters in the *IBM® Informix® Administrator's Reference*.

Specify an offset for additional chunks

To specify an offset for additional chunks of database server space, you must supply the offset as a parameter when you assign the space to the database server.

For more information, see Creating a dbspace that uses the default page size.

Use offsets to create multiple chunks

You can create multiple chunks from a disk partition, disk device, or file, by specifying offsets and assigning chunks that are smaller than the total space available.

The offset specifies the beginning location of a chunk. The database server determines the location of the last byte of the chunk by adding the size of the chunk to the offset.

For the first chunk, assign any initial offset, if necessary, and specify the size as an amount that is less than the total size of the allocated disk space. For each additional chunk, specify the offset to include the sizes of all previously assigned chunks, plus the initial offset, and assign a size that is less than or equal to the amount of space remaining in the allocation.

Allocating cooked file spaces on UNIX

The following procedure shows an example of allocating disk space for a cooked file.

,

To allocate disk space for a cooked file called usr/data/my_chunk, on UNIX:

1. Log-in as user **informix**: **su informix**
2. Change directories to the directory where the cooked space will be located: **cd /usr/data**
3. Create your chunk by concatenating null to the file name that the database server will use for disk space: **cat /dev/null > my_chunk**
4. Set the file permissions to `660` (rw-rw----): **chmod 660 my_chunk**
5. You must set both group and owner of the file to **informix**:

```
ls -l my_chunk  -rw-rw----
  1  informix   informix
  0  Oct 12 13:43 my_chunk
```

6. Use **onspaces** to create the storage space or chunk.

For information about how to create a storage space using the file you have allocated, see Creating a dbspace that uses the default page size, Creating a blobspace, and Creating an sbspace.

Allocating raw disk space on UNIX

To allocate raw space, you must have a disk partition available that is dedicated to raw space. To create raw disk space, you can either repartition your disks or unmount an existing file system. Back up any files before you unmount the device.

To allocate raw disk space

1. Create and install a raw device.
   For specific instructions on how to allocate raw disk space on UNIX, see your operating-system documentation and Unbuffered or buffered disk access on UNIX.
2. Change the ownership and permissions of the character-special devices to **informix**.
   The file name of the character-special device usually begins with the letter `r`. For the procedure, see steps 4 and 5 in Allocating cooked file spaces on UNIX.
3. Verify that the operating-system permissions on the character-special devices are `crw-rw----`.
4. Create a symbolic link between the character-special device name and another file name with the UNIX link command, **ln -s**. For details, see Create symbolic links to raw devices (UNIX).

Restriction: After you create the raw device that the database server uses for disk space, do not create file systems on the same raw device that you allocate for the database server disk space. Also, do not use the same raw device as swap space that you allocate for the database server disk space.
Related concepts:
Partitions and offsets

Create symbolic links to raw devices (UNIX)

Use symbolic links to assign standard device names and to point to the device.

To create a link between the character-special device name and another file name, use the UNIX link command (usually **ln**). To verify that both the devices and the links exist, run the UNIX command **ls -l** (**ls -lg** on BSD) on your device directory. The following example shows links to raw devices. If your operating system does not support symbolic links, hard links also work.

```
ln -s /dev/rxy0h /dev/my_root # orig_device link to symbolic_name
ln -s /dev/rxy0a /dev/raw_dev2
ls -l
crw-rw---  /dev/rxy0h
crw-rw---  /dev/rxy0a
lrwxrwxrwx /dev/my_root@->/dev/rxy0h
lrwxrwxrwx /dev/raw_dev2@->/dev/rxy0a
```

Why use symbolic links? If you create chunks on a raw device and that device fails, you cannot restore from a backup until you replace the raw device and use the same path name. All chunks that were accessible at the time of the last backup must be accessible when you perform the restore.

Symbolic links simplify recovery from disk failure and enable you to replace quickly the disk where the chunk is located. You can replace a failed device with another device, link the new device path name to the same file name that you previously created for the failed device, and restore the data. You are not required to wait for the original device to be repaired.

Allocating NTFS file space on Windows

On Windows, the database server uses NTFS files by default. You can use standard file names for unbuffered files in the NTFS file system. If all your partitions are FAT files, you can convert one to NTFS.

To allocate NTFS file space for database server disk space or mirrored space, the first step is to create a null (zero bytes) file.

To allocate NTFS file space:

1. Log in as a member of the **Informix-Admin** group.
2. Open an MS-DOS command shell.
3. Change to the directory where you want to allocate the space, as in the following example:

   ```
   c:> cd \usr\data
   ```

4. If necessary, convert the partition to NTFS by running the following command: **convert /fs:ntfs**
5. Create a null file with the following command: **c:> copy nul my_chunk**
6. If you want to verify that the file was created, use the **dir** command to do so.

After you allocate the file space, you can create the dbspace or other storage space as you normally would, using **onspaces**. For information about how to create a dbspace or a blobspace, see Creating a dbspace that uses the default page size and Creating a blobspace.

Allocating raw disk space on Windows

You can configure raw disk space on Windows as a logical drive or physical drive.

To find the drive letter or disk number, run the **Disk Administrator**. If the drives must be striped (multiple physical disks combined into one logical disk), only logical drive specification would work.

You must be a member of the **Informix-Admin** group when you create a storage space or add a chunk. The raw disk space can be formatted or unformatted disk space.

Important: If you allocate a formatted drive or disk partition as raw disk space and it contains data, the database server overwrites the data when it begins to use the disk space. You must ensure that any data on raw disk space is expendable before you allocate the disk space to the database server.
To specify a logical drive:

1. Assign a drive letter to the disk partition.
2. Specify the following value for ROOTDBS in the onconfig file: `\\.\drive_letter`
3. To create a storage space or add a chunk, specify the logical drive partition.
   This example adds a chunk of 5000 KB on the e: drive, at an offset of 5200 KB, to dbspace **dpspc3**.

   ```
   onspaces -a dbspc3 \\.\e: -o 5200 -s 5000
   ```

To specify a physical drive

1. If the disk partition has *not* been assigned a drive letter, specify the following value for ROOTDBS in the onconfig file: `\\.\PhysicalDrive<number>`
2. To create a storage space or add a chunk, specify the physical drive partition.
   This example adds a chunk of 5000 KB on **PhysicalDrive0**, at an offset of 5200 KB, to dbspace **dpspc3**.

   ```
   onspaces -a dbspc3 \\.\PhysicalDrive0 : -o 5200 -s 5000
   ```


Specify names for storage spaces and chunks

Chunk names follow the same rules as storage-space names.

Specify an explicit path name for a storage space or chunk as follows:

- If you are using raw disks on UNIX, you must use a linked path name. (See Create symbolic links to raw devices (UNIX).)
- If you are using raw disks on Windows, the path name takes the following form, where *x* specifies the disk drive or partition:

  `\\.\x:`

- If you are using a file for database server disk space, the path name is the complete path and file name.

Use these naming rules when you create storage spaces or add a chunk. The file name must have the following characteristics:

- Be unique and not exceed 128 bytes
- Begin with a letter or underscore
- Contain only letters, digits, underscores, or $ characters

The name is not case-sensitive unless you use quotation marks around it. By default, the database server converts uppercase characters in the name to lowercase. If you want to use uppercase in names, put quotation marks around them and set the DELIMIDENT environment variable to ON.

- Specify the maximum size of chunks
  On most platforms, the maximum chunk size is 4 terabytes, but on other platforms, the maximum chunk size is 8 terabytes.
- Specify the maximum number of chunks and storage spaces
  You can specify a maximum of 32,766 chunks for a storage space, and a maximum of 32,766 storage spaces on the database server system.
- Back up after you change the physical schema
  You must perform a level-0 backup of the root dbspace and the modified storage spaces to ensure that you can restore the data in certain circumstances.

Related concepts:
Chunks

Specify the maximum size of chunks

On most platforms, the maximum chunk size is 4 terabytes, but on other platforms, the maximum chunk size is 8 terabytes.

To determine which chunk size your platform supports see your machine notes file. If you have upgraded from a version before version 10.00 and did not run the **onmode -BC 2** command, the maximum chunk size is 2 GB.

Specify the maximum number of chunks and storage spaces

You can specify a maximum of 32,766 chunks for a storage space, and a maximum of 32,766 storage spaces on the database server system.

The storage spaces can be any combination of dbspaces, blobspaces, and sbspaces.

Considering all limits that can apply to the size of an instance of the database server, the maximum size of an instance is approximately 8 petabytes.

If you have upgraded from a version before version 10.00, you must run **onmode -BC 2** to enable the maximum number of chunks and storage spaces.

Back up after you change the physical schema

You must perform a level-0 backup of the root dbspace and the modified storage spaces to ensure that you can restore the data in certain circumstances.

Perform a level-0 backup to ensure that you can restore data when you:

- Add or drop mirroring
- Drop a logical-log file
- Change the size or location of the physical log
- Change your storage-manager configuration
- Add, move, or drop a dbspace, blobspace, or sbspace
- Add, move, or drop a chunk to a dbspace, blobspace, or sbspace

Important: When you add a new logical log, you no longer are required to perform a level-0 backup of the root dbspace and modified dbspace to use the new logical log. However, you must perform the level-0 backup to prevent level-1 and level-2 backups from failing.
You must perform a level-0 backup of the modified storage spaces to ensure that you can restore the unlogged data before you switch to a logging table type:

- When you convert a nonlogging database to a logging database
- When you convert a RAW table to standard

Monitor storage spaces

You can monitor the status of storage spaces and configure how you are notified when a storage space becomes full.

When a storage space or partition becomes full, a message is shown in the online message log file.

You can configure alarms that are triggered when storage spaces become full with the STORAGE_FULL_ALARM configuration parameter. You can specify how often alarms are sent and the minimum severity level of alarms to be sent. By default, the alarm interval is 600 seconds and the alarm severity level is 3. For more information about the STORAGE_FULL_ALARM configuration parameters and event alarms, see the *IBM Informix Administrator's Reference*.

If the primary server in a high-availability cluster encounters an out-of-space condition, and the STORAGE_FULL_ALARM configuration parameter is enabled, the event alarm is triggered and an error status is returned on the primary server but not on any of the secondary servers. This is expected behavior because log records are no longer sent from the primary server to the secondary servers when the primary server encounters an out-of-space condition. In this case, the secondary servers never exceed their storage limits and thus do not trigger an event alarm or return an error status.

You can use the IBM® Informix® Scheduler to set up a task that automatically monitors the status of storage spaces. The properties of the task define the information that Scheduler collects and specifies how frequently the task runs. For example, you might define a task to monitor storage spaces every hour, five days a week. For more information, see The Scheduler and Creating a task.

Manage dbspaces

This section contains information about creating standard and temporary dbspaces with and without the default page size, specifying the first and next extent sizes for the tblspace **tblspace** in a dbspace when you create the dbspace, and adding a chunk to a dbspace or blobspace.

For information about monitoring dbspaces, see Monitor storage spaces.

- Creating a dbspace that uses the default page size
  You can use **onspaces** to create a standard dbspace and a temporary dbspace.
- Creating a dbspace with a non-default page size
  You can specify a page size for a standard or temporary dbspace if you want a longer key length than is available for the default page size.
- Improving the performance of cooked-file dbspaces by using direct I/O
  On UNIX systems, you can improve the performance of cooked files used for dbspace chunks by using direct I/O.
- Storing multiple named fragments in a single dbspace
  For fragmented tables that use expression-based, interval, list, or round-robin distribution schemes, you can create named fragments that can be located within a single dbspace.
- Creating a temporary dbspace
  To specify where to allocate the temporary files, create temporary dbspaces.
- What to do if you run out of disk space
  When the initial chunk of the dbspace that you are creating is a cooked file on UNIX or an NTFS file on Windows, the database server verifies that the disk space is sufficient for the initial chunk.
- Adding a chunk to a dbspace or blobspace
  You add a chunk when a dbspace, blobspace, or sbspace is becoming full or requires more disk space.
- Rename dbspaces
  You can use the **onspaces** utility to rename a dbspace if you are user **informix** or have DBA privileges and the database server is in quiescent mode (and not any other mode).
- Managing automatic location and fragmentation
  You can control whether the database server automatically chooses the location for databases, indexes, and tables and automatically fragments tables. You can control the list of dbspaces in which the database server stores databases, indexes, and table fragments.

Related concepts:
Control of where simple large object data is stored
Size of the root dbspace

Creating a dbspace that uses the default page size

You can use **onspaces** to create a standard dbspace and a temporary dbspace.

For information about creating a dbspace with a non-default page size, see Creating a dbspace with a non-default page size.

Any newly added dbspace (and its mirror, if one exists) is available immediately. If you are using mirroring, you can mirror the dbspace when you create it. Mirroring takes effect immediately.

To create a standard dbspace using **onspaces**:

1. On UNIX, you must be logged in as user **informix** or **root** to create a dbspace.
   On Windows, users in the **Informix-Admin** group can create a dbspace.
2. Ensure that the database server is in online, administration, or quiescent mode.
3. Allocate disk space for the dbspace, as described in Allocate disk space.
4. To create a dbspace, use the **onspaces -c -d** options.
   KB is the default unit for the **-s** *size* and **-o offset** options. To convert KB to megabytes, multiply the unit by 1024 (for example, `10 MB = 10 * 1024 KB`).

See Creating a dbspace with a non-default page size for information about additional **onspaces** options if you are creating a dbspace with a non-default page size.

5. If you do not want to specify the first and next extent sizes for the tblspace **tblspace** in a dbspace, go to 6.
   If you want to specify the first and next extent sizes for the tblspace **tblspace** in a dbspace, see additional information in Specifying the first and next extent sizes for the tblspace tblspace.

6. After you create the dbspace, you must perform a level-0 backup of the root dbspace and the new dbspace.

The following example shows how to create a 10-megabyte mirrored dbspace, **dbspce1**, with an offset of 5000 KB for both the primary and mirror chunks, using raw disk space on UNIX:

```
onspaces -c -d dbspce1 -p /dev/raw_dev1 -o 5000 -s 10240 -m /dev/raw_dev2 5000
```

The following example shows how to create a 5-megabyte dbspace, **dbspc3**, with an offset of 200 KB, from raw disk space (drive e:) on Windows:

```
onspaces -c -d dbspc3 \\.\e: -o 200 -s 5120
```

For more information about creating a dbspace with **onspaces**, see Dbspaces and information about the **onspaces** utility in the *IBM® Informix® Administrator's Reference*.

- Specifying the first and next extent sizes for the tblspace tblspace
  You can specify first and next extent sizes if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you must place the tblspace **tblspace** extents in non-primary chunks. (A *primary chunk* is the initial chunk in a dbspace.)

Specifying the first and next extent sizes for the tblspace tblspace

You can specify first and next extent sizes if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you must place the tblspace **tblspace** extents in non-primary chunks. (A *primary chunk* is the initial chunk in a dbspace.)

You can choose to specify the first extent size, the next extent size, both the first and the next extent size, or neither extent size. If you do not specify first or next extent sizes for the tblspace **tblspace**, IBM® Informix® uses the existing default extent sizes.

You can use the TBLTBLFIRST and TBLTBLNEXT configuration parameters to specify the first and next extent sizes for the tblspace **tblspace** in the root dbspace that is created when the server is initialized.

You can use the **onspaces** utility to specify the first and next extent sizes for the tblspace **tblspace** in non-root dbspaces.

You can only specify the first and next extent sizes when you create dbspace. You cannot alter the specification of the first and next extent sizes after the creation of the dbspace. In addition, you cannot specify extent sizes for temporary dbspaces, sbspaces, blobspaces, or external spaces. You cannot alter the specification of the first and next extents sizes after the creation of the dbspace.

To specify the first and next extent sizes:

1. Determine the total number of pages required in the tblspace **tblspace**. The number of pages is equal to the sum of the number of tables, detached indexes, and table fragments likely to be located in the dbspace plus one page for the tblspace **tblspace**.
2. Calculate the number of KB required for the number of pages. This number depends on the number of KB to a page on the system.
3. Determine the space management requirements on your system by considering the importance of having all of the extents for the tblspace **tblspace** allocated during dbspace creation and whether the extents must be allocated contiguously. The more important these issues are, the larger the first extent size must be. If you are less concerned with having non-contiguous extents, possibly in secondary chunks, then the first and next extent sizes can be smaller.
4. Specify the extent size as follows:
   - If the space requirement is for the root dbspace, specify the first extent size in the TBLTBLFIRST configuration parameter and the next extent size in the TBLTBLNEXT configuration parameter. Then initialize the database server instance.
   - If the space requirement is for a non-root dbspace, indicate the first and next extent sizes on the command line using the **onspaces** utility to create the dbspace.

Extent sizes must be in KB and must be multiples of the page size. When you specify first and next extent sizes, follow these guidelines:

Type of extent	Minimum size	Maximum size
First extent in a non-root dbspace	The equivalent of 50 pages, specified in KB. This is the system default. For example, for a 2 KB page system, the minimum length is 100.	The size of the initial chunk, minus the space required for any system objects such as the reserved pages, the database tblspace, and the physical and logical logs.
First extent in a root dbspace	The equivalent of 250 pages specified in KB. This is the system default.	The size of the initial chunk, minus the space required for any system objects such as the reserved pages, the database tblspace, and the physical and logical logs.
Next Extent	Four times the disk-page size on the system. The default is 50 pages on any type of dbspace.	The maximum chunk size minus three pages.

You use the following **onspaces** utility **-ef** and **-en** options to specify the first and next extent sizes for the tblspace **tblspace** in non-root dbspaces:

- First extent size: **-ef** *size_in_kbytes*
- Next extent size: **-en** *size_in_kbytes*

For example, you can specify:

```
onspaces -c -d dbspace1 -p /usr/data/dbspace1 -o 0 -s 1000000 -e 2000 -n 1000
```

You can use **Oncheck -pt** and **oncheck -pT** to show the first and next extent sizes of a tblspace **tblspace**.

If data replication is being used and a dbspace is created on the primary database server, the first and next extent sizes are passed to the secondary database server through the ADDCHK log record.

For more information about the **onspaces** utility, **oncheck** commands, and specifying the first and next extent sizes for the tblspace **tblspace**, see the *IBM Informix Administrator's Reference*.

Related information:
TBLTBLFIRST configuration parameter
TBLTBLNEXT configuration parameter

Creating a dbspace with a non-default page size

You can specify a page size for a standard or temporary dbspace if you want a longer key length than is available for the default page size.

The root dbspace uses the default page size. If you want to create a dbspace with a different page size, the size must be an integral multiple of the default page size, and cannot be greater than 16 KB.

For systems with sufficient storage, the performance advantages of a larger page size include:

- Reduced depth of B-tree indexes, even for smaller index keys.
- Decreased checkpoint time, which typically occurs with larger page sizes.

Additional performance advantages occur because you can:

- Group on the same page long rows that currently span multiple pages of the default page size.
- Define a different page size for temporary tables, so the temporary tables have a separate buffer pool.

A table can be in one dbspace and the index for that table can be in another dbspace. The page size for these partitions can be different.

To create a dbspace with a non-default page size:

1. If you upgraded from a version before version 10.00, run the **onmode -BC 2** command to enable the large chunk mode. By default, when IBM® Informix® is first initialized or restarted, starts with the large chunk mode enabled.
2. Optional: Create a buffer pool that corresponds to the page size of the dbspace. You can use the **onparams** utility or the BUFFERPOOL configuration parameter. If you create a dbspace with a page size that does not have a corresponding buffer pool, automatically creates a buffer pool using the default values for the BUFFERPOOL configuration parameter as defined in the onconfig file.

   You cannot have multiple buffer pools with the same page size.

3. Define the page size of the dbspace when you create the dbspace. You can use the **onspaces** utility.

Tip: If you use non-default page sizes, you might be required to increase the size of your physical log. If you perform many updates to non-default pages you might require a 150 - 200 percent increase of the physical log size. Some experimentation might be required to tune the physical log. You can adjust the size of the physical log as necessary according to how frequently the filling of the physical log triggers checkpoints.

Improving the performance of cooked-file dbspaces by using direct I/O

On UNIX systems, you can improve the performance of cooked files used for dbspace chunks by using direct I/O.

Direct I/O must be available and the file system must support direct I/0 for the page size used for the dbspace chunk.
You can use IBM® Informix® to use either raw devices or cooked files for dbspace chunks. In general, cooked files are slower because of the additional overhead and buffering provided by the file system. Direct I/O bypasses the use of the file system buffers, and therefore is more efficient for reads and writes that go to disk. You specify direct I/O with the DIRECT_IO configuration parameter. If your file system supports direct I/O for the page size used for the dbspace chunk and you use direct I/O, performance for cooked files can approach the performance of raw devices used for dbspace chunks.

To improve the performance of cooked-file dbspaces by using direct I/O:

1. Verify that you have direct I/O and the file system supports direct I/O for the page size used for the dbspace chunk.
2. Enable direct I/O by setting the DIRECT_IO configuration parameter to `1`.

If you have an AIX® operating system, you can also enable concurrent I/O for to use with direct IO when reading and writing to chunks that use cooked files.

For more information about using direct IO or concurrent IO, see the *IBM Informix Performance Guide*.

Storing multiple named fragments in a single dbspace

For fragmented tables that use expression-based, interval, list, or round-robin distribution schemes, you can create named fragments that can be located within a single dbspace.

Storing multiple table or index fragments in a single dbspace improves query performance over storing each fragment in a different dbspace and simplifies management of dbspaces.

Suppose you are creating a fragmented table using an expression-based distribution scheme in which each expression specifies the data sets that are placed in particular fragments. You might decide to separate the data in the table with data from one month in one dbspace and data from the next 11 months in 11 other dbspaces. However, if you want to use only one dbspace for all the yearly data, you can create named fragments so the data for each month is stored in one dbspace.

If you create a fragmented table with named fragments, each row in the **sysfragments** system catalog table contains a fragment name in the **partition** column. If you create a fragmented table without named fragments, the name of the dbspace is in the **partition** column. The **flags** column in the **sysfragments** system catalog table tells you if the fragmentation scheme has named fragments.

You can create tables and indexes with named fragments, and you can create, drop, and alter named fragments using the PARTITION keyword and the fragment name.

To create a fragmented table with named fragments, use SQL syntax as shown in the following example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
     PARTITION part1 (a >=0 AND a < 5) IN dbspace1,
     PARTITION part2 (a >=5 AND a < 10) IN dbspace1
        ...
      ;
```

If you created a table or index fragment containing named fragments, you must use syntax containing the fragment name when you use the ALTER FRAGMENT statement, as shown in the following examples:

```
ALTER FRAGMENT ON TABLE tb1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (a >=0 AND a < 5) IN dbspace1,
    PARTITION part_2 (a >=5 AND a < 10) IN dbspace1;

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
    PARTITION part_1 (a >=0 AND a < 5) IN dbspace1,
    PARTITION part_2 (a >=5 AND a < 10) IN dbspace1;
```

You can use the `PARTITION BY EXPRESSION` keywords in place of the `FRAGMENT BY EXPRESSION` keywords in the CREATE TABLE, CREATE INDEX, and ALTER FRAGMENT ON INDEX statements, as shown in this example:

```
ALTER FRAGMENT ON INDEX idx1 INIT PARTITION BY EXPRESSION
           PARTITION part1   (a <= 10) IN idxdbspc1,
           PARTITION part2   (a <= 20) IN idxdbspc1,
           PARTITION part3   (a <= 30) IN idxdbspc1;
```

Use ALTER FRAGMENT syntax to change fragmented tables and indexes that do not have named fragments into tables and indexes that have named fragments. The following syntax shows how you might convert a fragmented table with multiple dbspaces into a fragmented table with named fragments:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1,
     (c1=20) IN dbs2;
ALTER FRAGMENT ON TABLE t1 MODIFY dbs2 TO PARTITION part_3 (c1=20)
 IN dbs1
```

The following syntax shows how you might convert a fragmented index into an index that contains named fragments:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
CREATE INDEX ind1 ON t1 (c1) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
     PARTITION part_1 (c1=10) IN dbs1, PARTITION part_2 (c1=20) IN dbs1,
     PARTITION part_3 (c1=30) IN dbs1,
```

See the *IBM® Informix® Performance Guide* for more information about fragmentation, including fragmentation guidelines, procedures for fragmenting indexes, procedures for creating attached and detached indexes with named fragments, and examples of SQL statements used to create attached and detached indexes containing named fragments.

See the *IBM Informix Guide to SQL: Syntax* for more syntax details, including information about named fragments in the GRANT FRAGMENT and REVOKE FRAGMENT statements, and details for using the DROP, DETACH, and MODIFY clauses of the ALTER FRAGMENT statement.

Creating a temporary dbspace

To specify where to allocate the temporary files, create temporary dbspaces.

To define temporary dbspaces:

1. Use the **onspaces** utility with the **-c -d -t** options.
   For more information, see Creating a dbspace that uses the default page size.

   Or, use the **create tempdbspace** command with the SQL administration API **admin()** or **task()** function.

   For more information, see create tempdbspace argument: Create a temporary dbspace (SQL administration API).

2. Use the DBSPACETEMP environment variables or the DBSPACETEMP configuration parameter to specify the dbspaces that the database server can use for temporary storage.
   The DBSPACETEMP configuration parameter can contain dbspaces with a mix of page sizes.

   For further information about DBSPACETEMP, see the topics about configuration parameters in the *IBM® Informix® Administrator's Reference*.

3. If you create more than one temporary dbspace, the dbspaces should be located on separate disks to optimize the I/O.

After you have created a temporary dbspace, you must make the database server aware of the existence of the newly created temporary dbspace by setting the DBSPACETEMP configuration parameter, the DBSPACETEMP environment variable, or both.

The following example shows how to create 5-megabyte temporary dbspace named **temp_space** with an offset of 5000 KB:

```
onspaces -c -t -d temp_space -p /dev/raw_dev1 -o 5000 -s 5120
```

The equivalent SQL administration API statement:

```
EXECUTE FUNCTION task("create tempdbspace", "temp_space", "/dev/raw_dev1", "5 MB", "5000 KB");
```

By allocating the space from a pre-defined storage pool you may avoid specifying the device or offset:

```
EXECUTE FUNCTION task("create tempdbspace from storagepool", "temp_space", "5 MB");
```

For more information, see Temporary dbspaces.

Related concepts:
Size of the root dbspace
Related information:
create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API)
create tempdbspace argument: Create a temporary dbspace (SQL administration API)

What to do if you run out of disk space

When the initial chunk of the dbspace that you are creating is a cooked file on UNIX or an NTFS file on Windows, the database server verifies that the disk space is sufficient for the initial chunk.

If the size of the chunk is greater than the available space on the disk, a message is displayed and no dbspace is created. However, the cooked file that the database server created for the initial chunk is not removed. Its size represents the space left on your file system before you created the dbspace. Remove this file to reclaim the space.

Adding a chunk to a dbspace or blobspace

You add a chunk when a dbspace, blobspace, or sbspace is becoming full or requires more disk space.

Important: The newly added chunk (and its associated mirror, if one exists) is available immediately. If you are adding a chunk to a mirrored storage space, you must also add a mirror chunk.
To add a chunk using **onspaces**:

1. On UNIX, you must be logged in as user **informix** or **root** to add a chunk.
   On Windows, users in the **Informix-Admin** group can add a chunk.
2. Ensure that the database server is in online, administration, or quiescent mode, or the cleanup phase of fast-recovery mode.
3. Allocate disk space for the chunk, as described in Allocate disk space.
4. To add a chunk, use the **-a** option of **onspaces**.
   If the storage space is mirrored, you must specify the path name of both a primary chunk and mirror chunk.

   If you specify an incorrect path name, offset, or size, the database server does not create the chunk and displays an error message. Also see What to do if you run out of disk space.
5. After you create the chunk, you must perform a level-0 backup of the root dbspace and the dbspace, blobspace, or sbspace that contains the chunk.

The following example adds a 10-megabyte mirror chunk to **blobsp3**. An offset of 200 KB for both the primary and mirror chunk is specified. If you are not adding a mirror chunk, you can omit the **-m** option.

```
onspaces -a blobsp3 -p /dev/raw_dev1 -o 200 -s 10240  -m /dev/raw_dev2 200
```

The next example adds a 5-megabyte chunk of raw disk space, at an offset of 5200 KB, to dbspace **dbspc3**.

```
onspaces -a dbspc3 \\.\e: -o 5200 -s 5120
```

You can also define information that Informix® can use to automatically extend the size of a chunk when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space will run out of space and when it will run out of space.

Related concepts:
Automatic space management
Related tasks:
Adding a chunk to an sbspace

Rename dbspaces

You can use the **onspaces** utility to rename a dbspace if you are user **informix** or have DBA privileges and the database server is in quiescent mode (and not any other mode).

To rename a dbspace use the following **onspaces** utility command:

```
onspaces -ren old_dbspace_name-n new_dbspace_name
```

You can rename standard dbspaces and all other spaces, including blobspaces, smart blobspaces, temporary spaces, and external spaces. However, you cannot rename any critical dbspace, such as a root dbspace or a dbspace that contains physical logs.

You can rename a dbspace and an sbspace:

- When Enterprise Replication is enabled
- On a primary database server when data replication is enabled

You cannot rename a dbspace and an sbspace on a secondary database server or when the secondary database server is part of the Enterprise Replication configuration

The rename dbspace operation only changes the dbspace name; it does not reorganize data.

The rename dbspace command updates the dbspace name in all places where that name is stored. This includes reserved pages on disk, system catalogs, the ONCONFIG configuration file, and in-memory data structures.

Important: After renaming a dbspace, perform a level-0 archive of the renamed dbspace and the root dbspace. For information, see the *IBM® Informix® Backup and Restore Guide*.

- Additional actions that may be required after you rename a dbspace
  If you rename a dbspace, you must rewrite and recompile any stored procedure code that references the old dbspace name.

Additional actions that may be required after you rename a dbspace

If you rename a dbspace, you must rewrite and recompile any stored procedure code that references the old dbspace name.

If you have a stored procedure that contains the ALTER FRAGMENT keywords and a reference to the dbspace name, you must rewrite and recompile that stored procedure.

If you rename dbspaces that are specified in the DATASKIP configuration parameter, you must manually update the DATASKIP configuration parameter after renaming the dbspace.

Managing automatic location and fragmentation

You can control whether the database server automatically chooses the location for databases, indexes, and tables and automatically fragments tables. You can control the list of dbspaces in which the database server stores databases, indexes, and table fragments.

If you enable automatic location and fragmentation, the database server performs the following tasks:

- Stores new databases for which you do not specify a location in the optimal dbspace instead of in the root dbspace. By default, all dbspaces except dbspaces that are dedicated to tenant databases are available.
- Stores new tables and indexes for which you do not specify a location in the optimal dbspace instead of in the same dbspace as the database.
- Allocates an initial number of round-robin fragments for new tables. A table fragment does not have an extent until a row is inserted into the fragment, unless you include the FIRST EXTENT clause in the CREATE TABLE statement.
- Adds more table fragments as the table grows.

To enable automatic location and fragmentation, set the AUTOLOCATE configuration parameter or the AUTOLOCATE session environment variable to a positive integer.

Automatic location is not applicable to tenant databases or the tables, fragments, and indexes within tenant databases.

To view the list of available dbspaces, query the **sysautolocate** system catalog table.

To add a dbspace to the list of available dbspaces, run the **task()** or **admin()** SQL administration API function with the **autolocate database**, the **autolocate database add**, or the **autolocate database anywhere** argument.

To remove a dbspace from the list of available dbspaces, run the **task()** or **admin()** SQL administration API function with the **autolocate database remove** argument.

To disable automatic location and fragmentation for tables in a particular database, run the **task()** or **admin()** SQL administration API function with the **autolocate database off** argument.

To disable automatic location and fragmentation of tables in all databases, set the AUTOLOCATE configuration parameter or the AUTOLOCATE session environment variable to 0.

Related concepts:
Table fragmentation and data storage
Storage space creation and management
Databases
Related information:
AUTOLOCATE configuration parameter
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)

autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)
AUTOLOCATE session environment option
SYSAUTOLOCATE

Manage blobspaces

This section explains how to create a blobspace and determine the blobpage size.

The database server stores TEXT and BYTE data in dbspaces or blobspaces, but blobspaces are more efficient. For information about adding a chunk, see Adding a chunk to a dbspace or blobspace.

For information about monitoring blobspaces, see Monitor storage spaces

- Creating a blobspace
  You can use **onspaces** to create a blobspace.
- Prepare blobspaces to store TEXT and BYTE data
  A newly created blobspace is not immediately available for storage of TEXT or BYTE data. Blobspace logging and recovery require that the statement that creates a blobspace and the statements that insert TEXT and BYTE data into that blobspace be created in separate logical-log files.
- Determine blobpage size
  When you create a blobspace, use the size of the most frequently occurring simple large object as the size of the blobpage. In other words, choose a blobpage size that wastes the least amount of space.

Creating a blobspace

You can use **onspaces** to create a blobspace.

Before you create a blobspace:

1. Allocate disk space for the blobspace, as described in Allocate disk space.
2. Determine what blobpage size is optimal for your environment.
   For instructions, see Determine blobpage size.

Specify a blobspace name of up to 128 bytes. The name must be unique and must begin with a letter or underscore. You can use letters, digits, underscores, and $ characters in the name.
Important: You can mirror the blobspace when you create it if mirroring is enabled for the database server. Mirroring takes effect immediately.
To create a blobspace using **onspaces**:

1. To create a blobspace on UNIX, you must be logged in as user **informix** or **root**.
   To create a blobspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is in online, administration, or quiescent mode, or the cleanup phase of fast-recovery mode.
3. To add a blobspace, use the **onspaces -c -b** options.
   a. Specify an explicit path name for the blobspace. If the blobspace is mirrored, you must specify the path name and size of both the primary chunk and mirror chunk.
   b. Use the **-o** option to specify an offset for the blobspace.
   c. Use the **-s** option to specify the size of the blobspace chunk, in KB.
   d. Use the **-g** option to specify the blobpage size in terms of the number of disk pages per blobpages.
   See Determine blobpage size. For example, if your database server instance has a disk-page size of 2 KB, and you want your blobpages to have a size of 10 KB, enter 5 in this field.

   If you specify an incorrect path name, offset, or size, the database server does not create the blobspace and displays an error message. Also see What to do if you run out of disk space.
4. After you create the blobspace, you must perform a level-0 backup of the root dbspace and the new blobspace.

The following example shows how to create a 10-megabyte mirrored blobspace, **blobsp3**, with a blobpage size of 10 KB, where the database server page size is 2 KB. An offset of 200 KB for the primary and mirror chunks is specified. The blobspace is created from raw disk space on UNIX.

```
onspaces -c -b blobsp3 -g 5 -p /dev/raw_dev1 -o 200 -s 10240 -m /dev/raw_dev2 200
```

For reference information about creating a blobspace with onspaces, see information about the **onspaces** utility in the *IBM® Informix® Administrator's Reference*.

Prepare blobspaces to store TEXT and BYTE data

A newly created blobspace is not immediately available for storage of TEXT or BYTE data. Blobspace logging and recovery require that the statement that creates a blobspace and the statements that insert TEXT and BYTE data into that blobspace be created in separate logical-log files.

This requirement is true for all blobspaces, regardless of the logging status of the database. To accommodate this requirement, switch to the next logical-log file after you create a blobspace. (For instructions, see Back up log files to free blobpages.)

Determine blobpage size

When you create a blobspace, use the size of the most frequently occurring simple large object as the size of the blobpage. In other words, choose a blobpage size that wastes the least amount of space.

For information about calculating an optimal blobpage size, see blobpage size considerations in the topics on the effect of configuration on I/O activity in the *IBM® Informix® Performance Guide*.

If a table has more than one TEXT or BYTE column, and the objects are not close in size, store each column in a different blobspace, each with an appropriately sized blobpage. See Tables.

- Determine database server page size
  When you specify the blobpage size, you specify it in terms of the database server base page size.
- Obtain blobspace storage statistics
  To help you determine the optimal blobpage size for each blobspace, you can use database server utility commands

Determine database server page size

When you specify the blobpage size, you specify it in terms of the database server base page size.

You can use one of the following methods to determine the database server page size for your system:

- Run the **onstat -b** utility to display the system page size, given as buffer size on the last line of the output.
- To view the contents of the PAGE_PZERO reserved page, run the **oncheck -pr** utility.

Obtain blobspace storage statistics

To help you determine the optimal blobpage size for each blobspace, you can use database server utility commands

The following database server utility commands to determine the optimal blobpage size:

- **oncheck -pe**
- **oncheck -pB**

The **oncheck -pe** command provides background information about the objects stored in a blobspace:

- Complete ownership information (displayed as *database:owner.table*) for each table that has data stored in the blobspace chunk
- The total number of pages used by each table to store its associated TEXT and BYTE data
- The total free and total overhead pages in the blobspace

The **oncheck -pB** command lists the following statistics for each table or database:

- The number of blobpages used by the table or database in each blobspace
- The average fullness of the blobpages used by each simple large object stored as part of the table or database

For more information, see Monitor blobspace usage with oncheck -pe, Determine blobpage fullness with oncheck -pB, and optimizing blobspace blobpage size in the topics about table performance considerations in the *IBM® Informix® Performance Guide*.

Manage sbspaces

This section describes how to create a standard or temporary sbspace, monitor the metadata and user-data areas, add a chunk to an sbspace, and alter storage characteristics of smart large objects.

For information about monitoring sbspaces, see Monitor storage spaces.

- Creating an sbspace
  Use the **onspaces** utility or IBM® Informix® Server Administrator (ISA) to create an sbspace.
- Size sbspace metadata
  The first chunk of an sbspace must have a metadata area. It is important to size the metadata area for an sbspace correctly to ensure that the sbspace does not run out of metadata space.

- Adding a chunk to an sbspace
  You can add a chunk to an sbspace or temporary sbspace.
- Alter storage characteristics of smart large objects
- Creating a temporary sbspace

Creating an sbspace

Use the **onspaces** utility or IBM® Informix® Server Administrator (ISA) to create an sbspace.

Use the **onspaces** utility to create an sbspace.

To create an sbspace using **onspaces**:

1. To create an sbspace on UNIX, you must be logged in as user **informix** or **root**.
   To create an sbspace on Windows, you must be a member of the **Informix-Admin** group.
2. Ensure that the database server is online, administration, or quiescent mode, or in the cleanup phase of fast-recovery mode.
3. Use the **onspaces -c -S** options to create the sbspace.
   a. Use the -p option to specify the path name, the -o option to specify the offset, and the -s option to specify the sbspace size.
   b. If you want to mirror the sbspace, use the -m option to specify the mirror path and offset.
   c. If you want to use the default storage characteristics for the sbspace, omit the -Df option.
      If you want to specify different storage characteristics, use the -Df option. For more information, see Storage characteristics of sbspaces.
   d. The first chunk in an sbspace must have a metadata area.
      You can specify a metadata area for an sbspace or let the database server calculate the size of the metadata area. For more information, see Size sbspace metadata.
4. After you create the sbspace, you must perform a level-0 backup of the root dbspace and the new sbspace.
5. To start storing smart large objects in this sbspace, specify the space name in the SBSPACENAME configuration parameter.
6. Use **onstat -d**, **onstat -g smb s**, and **oncheck -cs**, **-cS**, **-ps**, or **-pS** to display information about the sbspace.
   For more information, see Monitor sbspaces.

This shows how to create a 20-megabyte mirrored sbspace, **sbsp4**. Offsets of 500 KB for the primary and 500 KB for the mirror chunks are specified, and a metadata size of 150 KB with a 200 KB offset. The AVG_LO_SIZE -Df tag specifies an expected average smart-large-object size of 32 KB.

```
onspaces -c -S sbsp4 -p /dev/rawdev1 -o 500 -s 20480 -m /dev/rawdev2 500
-Ms 150 -Mo 200 -Df "AVG_LO_SIZE=32"
```

For information about creating an sbspace and default options for smart large objects, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*. For information about creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Size sbspace metadata

The first chunk of an sbspace must have a metadata area. It is important to size the metadata area for an sbspace correctly to ensure that the sbspace does not run out of metadata space.

When you add smart large objects and chunks to the sbspace, the metadata area grows. In addition, the database server reserves 40 percent of the user area to be used in case the metadata area runs out of space.

To ensure that this does not happen, you can either:

- Let the database server calculate the size of the metadata area for the new sbspace chunk.
- Specify the size of the metadata area explicitly.

For instructions on estimating the size of the sbspace and metadata area, see table performance considerations in the *IBM® Informix® Performance Guide*. Also see Monitoring the metadata and user-data areas.

Adding a chunk to an sbspace

You can add a chunk to an sbspace or temporary sbspace.

You can specify a metadata area for a chunk, let the database server calculate the metadata area, or use the chunk for user data only.

To add a chunk to an sbspace using **onspaces**:

1. Ensure that the database server is online, administration, or quiescent mode, or in the cleanup phase of fast-recovery mode.
2. Use the **onspaces -a** option to create the sbspace chunk.
   a. Use the **-p** option to specify the path name, the **-o** option to specify the offset, and the **-s** option to specify the chunk size.
   b. If you want to mirror the chunk, use the **-m** option to specify the mirror path and offset.

c. To specify the size and offset of the metadata space, use the **-Mo** and **-Ms** options.
The database server allocates the specified amount of metadata area on the new chunk.

d. To allow the database server to calculate the size of the metadata for the new chunk, omit the **-Mo** and **-Ms** options.
The database server divides the estimated average size of the smart large objects by the size of the user data area.

e. To use the chunk for user data only, specify the **-U** option.
If you use the **-U** option, the database server does not allocate metadata space in this chunk. Instead, the sbspace uses the metadata area in one of the other chunks.

3. After you add a chunk to the sbspace, the database server writes the CHRESERV and CHKADJUP log records.
4. Perform a level-0 backup of the root dbspace and the sbspace.
5. Use **onstat -d** and **oncheck -pe** to monitor the amount of free space in the sbspace chunk.

This example adds a 10-megabyte mirror chunk to **sbsp4**. An offset of 200 KB for both the primary and mirror chunk is specified. If you are not adding a mirror chunk, you can omit the **-m** option. The **-U** option specifies that the new chunk contains user data exclusively.

```
onspaces -a sbsp4 -p /dev/rawdev1 -o 200 -s 10240 -m /dev/rawdev2 200 -U
```

You can also define information that Informix® can use to automatically expand the size of a chunk when additional storage space is required for an application. If you have extendable chunks, you are not required to add new chunks or spend time trying to determine which storage space (dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace) will run out of space and when it will run out of space.

Related concepts:
Automatic space management
Related tasks:
Adding a chunk to a dbspace or blobspace
Related reference:
Monitor sbspaces

Alter storage characteristics of smart large objects

Use the **onspaces -ch** command to change the following default storage characteristics for the sbspace:

- Extent sizes
- Average smart-large-object size
- Buffering mode
- Last-access time
- Lock mode
- Logging

For more information, see Storage characteristics of sbspaces and managing sbspaces in the topics about table performance considerations in your *IBM® Informix® Performance Guide*.

Creating a temporary sbspace

For background information and the rules for determining where temporary smart large objects are stored, see Temporary sbspaces. You can store temporary smart large objects in a standard or temporary sbspace. You can add or drop chunks in a temporary sbspace.

To create a temporary sbspace with a temporary smart large object:

1. Allocate space for the temporary sbspace. For details, see Allocate disk space.
For information about SBSPACETEMP, see the configuration parameters topics in the *IBM® Informix Administrator's Reference*.

2. Create the temporary sbspace as the following example shows:

```
onspaces -c -S tempsbsp -t -p ./tempsbsp -o 0 -s 1000
```

3. You can specify any of the following **onspaces** options:
   a. Specify a metadata area and offset (**-Ms** and **-Mo**).
   b. Specify storage characteristics (**-Df**).
   You cannot turn on logging for a temporary sbspace.

4. Set the SBSPACETEMP configuration parameter to the name of the default temporary sbspace storage area.
Restart the database server.

5. Use **onstat -d** to display the temporary sbspace.
For information and an example of **onstat -d** output, see the **onstat** utility in the *IBM Informix Administrator's Reference*.

6. Specify the LO_CREATE_TEMP flag when you create a temporary smart large object.
Using DataBlade API:

```
mi_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
```

Using Informix® ESQL/C:

```
ifx_lo_specset_flags(lo_spec,LO_CREATE_TEMP);
```

For information about creating smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and *IBM Informix ESQL/C Programmer's Manual*.

Manage the plogspace

You create or move the plogspace with the **onspaces** utility or equivalent SQL administration API command.

To create the plogspace, run the **onspaces -c -P** command or the **admin()** or **task()** SQL administration API function with the **create plogspace** argument.

If you want to change the location of the plogspace to a different chunk, create a new plogspace. The physical log is moved to the new plogspace and the old plogspace is dropped.

You can modify the chunk in the plogspace in the following ways:

- Mark the chunk as not extendable. Run the **admin()** or **task()** SQL administration API function with the **modify chunk extendable off** argument
- Change the extend size of the chunk. The default extend size is 10000 KB. Run the **admin()** or **task()** SQL administration API function with the **modify space sp_sizes** argument.

Related concepts:
Plogspace
Related information:
onspaces -c -P: Create a plogspace
create plogspace: Create a plogspace (SQL administration API)
modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API)
modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)

Automatic space management

You can configure the server to add more storage space automatically when more space is required. You use space more effectively and ensure that space is allocated as necessary, while reducing out-of-space errors. You reduce the time required to manually monitor your spaces to determine which storage space might run out of free space. If you configure the server to automatically add space, you can also manually expand a space or extend a chunk.

When the server expands a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace, the server can add a chunk to the storage space. The server can also extend a chunk in a dbspace, plogspace, or temporary dbspace that is not mirrored. If the storage space is a non-mirrored dbspace or a temporary dbspace, the server can also extend a chunk in the storage space.

To configure for the automatic and manual space management, you run SQL administration API commands to perform these tasks:

1. Create, modify, and delete one or more entries in the storage pool. The storage pool contains entries for available raw devices, cooked files, and directories that Informix® uses to expand a storage space.
2. Mark a chunk as extendable.
3. Modify the create and extend size of a storage space (optional).
4. Change the threshold and wait time for the automatic addition of more space (optional).
5. Configure the frequency of the monitor low storage task (optional).

If your storage pool contains entries, you can also run SQL administration API commands to:

- Manually expand the storage space or extend a chunk, when you do not want to wait for the task that automatically expands the space to run.
- Manually create storage spaces from storage pool entries and return space from empty storage spaces to the storage pool.

By default, the SP_AUTOEXPAND configuration parameter is set to 1 to enable automatic expansion of storage spaces. If you do not want to the server to automatically expand space, set the SP_AUTOEXPAND configuration parameter to 0 to disable the automatic creation or extension of chunks. You can also specify that a chunk is not extendable.

Tip:
In some situations, the database server might not automatically expand a temporary dbspace that is listed in the DBSPACETEMP configuration parameter after you configured the server to automatically expand an existing storage space. If operations (such as an index build or sort) that use the temporary dbspace run out of space, you receive an out of space error. To work around this problem, you must manually add a chunk to the temporary dbspace or use a bigger temporary dbspace.

If you have a storage pool and the database server participates in Enterprise Replication, storage spaces that are necessary for replication are created automatically if needed when you define a replication server.

- Creating and managing storage pool entries
  You can add, modify, and delete the entries in the storage pool.
- Marking a chunk as extendable or not extendable
  You mark a chunk as extendable to enable the automatic or manual extension of the chunk. You can change the mark to not extendable to prevent the automatic or manual extension of the chunk.
- Modifying the sizes of an extendable storage space
  You can control how an extendable storage space in the storage pool grows by modifying the create size and the extend size. By default, the maximum size of an extendable storage space is unlimited. You can limit the size of an extendable storage space by setting a maximum size.
- Changing the threshold and wait time for the automatic addition of more space
  While Informix can react to out-of-space conditions by automatically extending or adding chunks when a storage space is full, you can also configure the server to extend or add chunks before a storage space is full.

- Configuring the frequency of the monitor low storage task
  You can change the frequency of the **mon_low_storage** task, which periodically scans the list of dbspaces to find spaces that fall below the threshold indicated by SP_THRESHOLD configuration parameter. If the task finds spaces that below the threshold, the task attempts to expand the space, by extending an extendable chunk or by using the storage pool to add a chunk.
- Manually expanding a space or extending an extendable chunk
  You can manually expand a space or extend a chunk when necessary, instead of waiting for Informix to automatically expand the space or extend a chunk.
- Example of minimally configuring for and testing the automatic addition of more space
  This example shows how you can minimally configure and then test the automatic addition of more space. You can do this by creating a dbspace, filling the space, adding an entry to the Informix storage pool, and loading tables into the space. When the space fills, Informix automatically expands it.
- Example of configuring for the automatic addition of more space
  This example shows how you can fully configure for the automatic addition of more space by changing some configuration parameter settings, changing the frequency of a task that monitors low storage, and specifying information for extendable spaces and chunks.

Related concepts:
Storage space creation and management
Size of the root dbspace
Extendable chunks
The storage pool
Related tasks:
Adding a chunk to a dbspace or blobspace
Adding a chunk to an sbspace
Creating a tenant database
Related information:
SP_WAITTIME configuration parameter

Creating and managing storage pool entries

You can add, modify, and delete the entries in the storage pool.

Each entry in the storage pool contains information about a directory, cooked file, or raw device that a database server instance can use if necessary to automatically expand an existing storage space.

Creating a storage pool entry

To create a storage pool entry, run the **admin()** or **task()** function with the storagepool add argument, as follows:

```
EXECUTE FUNCTION task("storagepool add", "path", "begin_offset",
"total_size", "chunk size", "priority");
```

Specify the following information:

- The path for the file, directory, or device that the server can use when additional storage space is required.
- The offset in KB into the device where Informix® can begin allocating space.
- The total space available to Informix in this entry. The server can allocate multiple chunks from this amount of space.
- The minimum size in KB of a chunk that can be allocated from the device, file, or directory. The smallest chunk that you can create is 1000 KB. Therefore, the minimum chunk size that you can specify is 1000 KB.
- A number from 1 to 3 for the priority (1 = high; 2 = medium; 3 = low). The server attempts to allocate space from a high-priority entry before it allocates space from a lower priority entry.

The default units for storage pool sizes and offsets are KB. However, you can specify information in any of the ways shown in the following examples:

- "100000"
- "100000 K"
- "100 MB"
- "100 GB"
- "100 TB"

Modifying a storage pool entry

To modify a storage pool entry, run the **admin()** or **task()** function with the storagepool modify argument, as follows:

```
EXECUTE FUNCTION task("storagepool modify", "storage_pool_entry_id",
"new_total_size", "new_chunk size", "new_priority");
```

Deleting storage pool entries

To delete a storage pool entry, run the **admin()** or **task()** function with the storagepool delete argument, as follows:

```
EXECUTE FUNCTION task("storagepool delete", "storage_pool_entry_id");
```

To delete all storage pool entries, run the **admin()** or **task()** function with the storagepool purge all argument, as follows:

```
EXECUTE FUNCTION task("storagepool purge all");
```

To delete all storage pool entries that are full, run the **admin()** or **task()** function with the storagepool purge full argument, as follows:

```
EXECUTE FUNCTION task("storagepool purge full");
```

To delete storage pool entries that have errors, run the **admin()** or **task()** function with the storagepool purge errors argument, as follows:

```
EXECUTE FUNCTION task("storagepool purge errors");
```

Examples

The following command adds a directory named /region2/dbspaces with a beginning offset of 0, a total size of 0, an initial chunk size of 20 MB, and a high priority. In this example the offset of 0 and the total size of 0 are the only acceptable entries for a directory.

```
EXECUTE FUNCTION task("storagepool add", "/region2/dbspaces", "0", "0",
"20000", "1");
```

The following command changes the total size, chunk size, and priority of storage pool entry 8 to 10 GB, 10 MB, and a medium priority.

```
EXECUTE FUNCTION task("storagepool modify", "8", "10 GB", "10000", "2");
```

The following command deletes the storage pool entry with an entry ID of 7:

```
EXECUTE FUNCTION task("storagepool delete", "7");
```

Related concepts:
The storage pool
Related information:
storagepool purge argument: Delete storage pool entries (SQL administration API)
storagepool modify argument: Modify a storage pool entry (SQL administration API)
storagepool delete argument: Delete one storage pool entry (SQL administration API)

Marking a chunk as extendable or not extendable

You mark a chunk as extendable to enable the automatic or manual extension of the chunk. You can change the mark to not extendable to prevent the automatic or manual extension of the chunk.

If a chunk is marked as not extendable:

- The server cannot automatically extend the chunk when there is little or no free space in the chunk. (However, if the storage pool contains entries, the server can expand a storage space by adding another chunk to the storage space.)
- You cannot manually extend the size of the chunk.

Prerequisite: An extendable chunk must be in an unmirrored dbspace or temporary dbspace.

To mark a chunk as extendable:

1. Run the admin() or task() function with the modify chunk extendable argument, as follows:
2. `EXECUTE FUNCTION task("modify chunk extendable", "chunk number");`

To mark a chunk as not extendable:

1. Run the admin() or task() function with the modify chunk extendable off argument, as follows:

   ```
   EXECUTE FUNCTION task("modify chunk extendable off", "chunk number");
   ```

The following command specifies that chunk 12 can be extended:

```
EXECUTE FUNCTION task("modify chunk extendable", "12");
```

Related concepts:
Extendable chunks
Related information:
modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)
modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API)

Modifying the sizes of an extendable storage space

You can control how an extendable storage space in the storage pool grows by modifying the create size and the extend size. By default, the maximum size of an extendable storage space is unlimited. You can limit the size of an extendable storage space by setting a maximum size.

To modify the create, extend, or maximum size of a storage space:

Run the **admin()** or **task()** SQL administration API function with the modify space sp_sizes argument, as follows:

```
EXECUTE FUNCTION task("modify space sp_sizes", "space_name",
 "new_create_size", "new_extend_size", "max_size");
```

The *space_name* is the name of the storage space.

The *new_create_size* is the minimum size that the server can use to create a new chunk in the specified dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace.

The *new_extend_size* is the minimum size that the server can use to extend a chunk in the specified unmirrored dbspace or temporary dbspace.

Specify either sizes with a number (for the number of KB) or a percentage (for a percentage of the total space).

The *max_size* is the maximum size, in KB, to which the server can expand the storage space. A value of 0 indicates unlimited.

The following command sets the create size to 60 MB, the extend size to 10 MB, and the maximum size to 200 MB for a space that is named **dbspace3**:

```
EXECUTE FUNCTION task("modify space sp_sizes", "dbspace3", "60000",
   "10000", "200000");
```

The following command sets the create size to 20 percent and the extend size to 1.5 percent for a space that is named **logdbs**:

```
EXECUTE FUNCTION task("modify space sp_sizes", "logdbs", "20", "1.5");
```

Related information:
modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)

Changing the threshold and wait time for the automatic addition of more space

While Informix® can react to out-of-space conditions by automatically extending or adding chunks when a storage space is full, you can also configure the server to extend or add chunks before a storage space is full.

Specify a threshold for the minimum amount of free KB in a storage space to trigger a task that expands the space.

You can also use the SP_WAITTIME configuration parameter to specify the maximum number of seconds that a thread waits for a space to expand before returning an out-of-space error.

To change the threshold and wait time:

1. Change the value of the threshold specified in the SP_THRESHOLD configuration parameter from 0 (disabled) to a non-zero value. Specify a value from either `1` to `50` for a percentage of a value from `1000` to the maximum size of a chunk in KB.
2. Change the value of the SP_WAITTIME configuration parameter, which specifies the maximum number of seconds that a thread waits for a space to expand before returning an out-of-space error.

Related information:
SP_THRESHOLD configuration parameter
SP_WAITTIME configuration parameter

Configuring the frequency of the monitor low storage task

You can change the frequency of the **mon_low_storage** task, which periodically scans the list of dbspaces to find spaces that fall below the threshold indicated by SP_THRESHOLD configuration parameter. If the task finds spaces that below the threshold, the task attempts to expand the space, by extending an extendable chunk or by using the storage pool to add a chunk.

The default frequency of the **mon_low_storage** task is once per hour, but you can configure the task to run more or less frequently

Prerequisite: Specify a value in the SP_THRESHOLD configuration parameter for the minimum amount of free KB in a storage space.

To configure the mon_low_storage task to run more or less frequently:

Run the following SQL statements, where *minutes* is the number of minutes between each run:

```
DATABASE sysadmin;
UPDATE ph_task set tk_frequency = INTERVAL (minutes)
 MINUTE TO MINUTE WHERE tk_name = "mon_low_storage";
```

For example, to configure the task to run every 10 minutes, run the following SQL statements:

```
DATABASE sysadmin;
UPDATE ph_task set tk_frequency = INTERVAL (10) MINUTE TO MINUTE
 WHERE tk_name = "mon_low_storage";
```


Manually expanding a space or extending an extendable chunk

You can manually expand a space or extend a chunk when necessary, instead of waiting for Informix® to automatically expand the space or extend a chunk.

Prerequisites:

- You can extend a chunk only if it is in an unmirrored dbspace or temporary dbspace.
- The chunk must be marked as extendable before it can be extended. If not, you must run the admin() or task() function with the modify chunk extendable argument to specify that the chunk is extendable.
- If a space cannot be expanded by extending a chunk, the storage pool must contain active entries that the server can use to create new chunks.

To immediately increase your storage space:

Either:

- Manually expand a space by running the admin() or task() function with the modify space expand argument, as follows:

  ```
  EXECUTE FUNCTION task("modify space expand", "space_name", "size");
  ```

  For example, the following command expands space number 8 by 1 gigabyte:

  ```
  EXECUTE FUNCTION task("modify space expand", "8", "1000000");
  ```

  The server expands the space either by extending a chunk in the space or adding a new chunk. The server might round the requested size up, depending on the page size of the storage space and the configured chunk size for any storage pool entry used during the expansion.

- Manually extend a chunk by running the admin() or task() function with the modify chunk extend argument, as follows:

  ```
  EXECUTE FUNCTION task("modify chunk extend", "chunk_number", "extend_amount");
  ```

  For example, the following command extends chunk number 12 by 5000 KB:

  ```
  EXECUTE FUNCTION task("modify chunk extend", "12", "5000");
  ```

  The server might round the requested size up, depending on the page size of the storage space.

Related concepts:
Extendable chunks
Related information:
modify space expand argument: Expand the size of a space (SQL administration API)
modify chunk extend argument: Extend the size of a chunk (SQL administration API)

Example of minimally configuring for and testing the automatic addition of more space

This example shows how you can minimally configure and then test the automatic addition of more space. You can do this by creating a dbspace, filling the space, adding an entry to the Informix® storage pool, and loading tables into the space. When the space fills, Informix automatically expands it.

To minimally configure for and test the automatic addition of more space:

1. Create a dbspace.
   For example, create a dbspace named `expandable_dbs` and allocate an initial chunk using the first 10000 KB of a cooked file named /my_directory/my_chunk, as follows:

   ```
   onspaces -c -d expandable_dbs -p /my_directory/my_chunk -o 0 -s 10000
   ```

2. Fill the dbspace.
   For example, fill the dbspace without loading a row of data. Instead, create a table and allocate a large set of contiguous free pages to the first extent, as follows:

   ```
   CREATE TABLE large_tab (col1 int) IN expandable_dbs EXTENT SIZE 10000000;
   ```

   You can monitor the free pages in your chunks by using the **onstat -d** command . If your dbspace is full, you receive out-of-space errors when attempting to create and load data into another new table.

3. Add an entry to the Informix storage pool.
   For example, add the $INFORMIXDIR/tmp directory to the storage pool, as follows:

   ```
   DATABASE sysadmin;
   EXECUTE FUNCTION task("storagepool add", "$INFORMIXDIR/tmp",
    "0", "0", "10000", "2");
   ```

4. In the SP_THRESHOLD configuration parameter, set a threshold for the minimum amount of free KB that can exist in a storage space before Informix automatically runs a task to expand the space.
5. Create and load new tables into your database.
   Now, if a storage space becomes full, instead of receiving an out-of-space error, Informix automatically creates a cooked file in the $INFORMIXDIR/tmp file and add a chunk to the expandable_dbs database using the new cooked file. As you continue to fill this chunk, the server automatically extends it. The server will always extend chunks if possible before adding new ones to a dbspace.

6. Reduce the free space in a storage space to test the value in the SP_THRESHOLD configuration parameter.
   Allocate enough pages in a storage space to reduce the free space so it is below the threshold indicated by SP_THRESHOLD. However, do not completely fill the space.

   You must see the space automatically expanded the next time that the **mon_low_storage** task runs.

7. Create an out-of-space condition.
   Allocate all pages in a storage space. Then try to allocate more pages. The allocation must be successful and you must not receive an out-of-space error.

   Informix writes messages to the log whenever it extends or adds a chunk and marks new chunks as extendable.

   Run the **onstat -d** command to display all chunks in the instance. Look for extendable chunks, which are marked with an `E` flag. The command output shows that the server automatically expanded the space, either through the addition of a new chunk or by extending the size of an existing chunk.

Example of configuring for the automatic addition of more space

This example shows how you can fully configure for the automatic addition of more space by changing some configuration parameter settings, changing the frequency of a task that monitors low storage, and specifying information for extendable spaces and chunks.

To configure for the automatic addition of more storage space:

1. Add entries to the storage pool.
   For example, add the $INFORMIXDIR/tmp directory to the storage pool, as follows:

   ```
   DATABASE sysadmin;
   EXECUTE FUNCTION task("storagepool add", "$INFORMIXDIR/tmp",
    "0", "0", "10000", "2");
   ```

2. Mark some chunks in unmirrored dbspaces and temporary dbspaces as extendable so that the server can extend the chunks if necessary in the future.
   For example, specify that chunk 12 can be extended:

   ```
   EXECUTE FUNCTION task("modify chunk extendable", "12");
   ```

   You can also change the mark to of an extendable chunk to not extendable. For example, specify that chunk number 10 cannot be extended:

   ```
   EXECUTE FUNCTION task("modify chunk extendable off", "10");
   ```

3. In the SP_THRESHOLD configuration parameter, set a threshold for the minimum amount of free KB that can exist in a storage space before Informix® automatically runs a task to expand the space. Specify either:
   - A value from `1` to `50` for a percentage,
   - A value from `1000` to the maximum size of the chunk in KB

   If an individual storage space fills beyond this threshold that you define and remains that full until the space-monitoring task (**mon_low_storage**) next runs, the server attempts to expand the space by extending an extendable chunk or by using the storage pool to add a chunk.

   For example, suppose the SP_THRESHOLD value is 5.5, which the server treats as 5.5 percent. If a space runs low on free pages, and the free space percentage falls below 5.5 percent and remains below that level until the **mon_low_storage** task runs next, that task attempts to expand the space. If SP_THRESHOLD is set to 50000 and a space has fewer than free 50000 KB, that space is expanded the next time **mon_low_storage** runs.

4. Optional: Change how often the **mon_low_storage** task runs. This task periodically scans the list of dbspaces to find spaces that fall below the threshold indicated by SP_THRESHOLD configuration parameter.
   For example, to configure the task to run every 10 minutes, run the following SQL statements:

   ```
   DATABASE sysadmin;
   UPDATE ph_task set tk_frequency = INTERVAL (10) MINUTE TO MINUTE
    WHERE tk_name = "mon_low_storage";
   ```

5. Optional: Change the value of the SP_WAITTIME configuration parameter, which specifies the maximum number of seconds that a thread waits for a space to expand before returning an out-of-space error.
6. Optional: Change two sizes that are associated with expanding a storage space:
   - The extend size, which is the minimum size that is used when extending a chunk in a dbspace, temporary dbspace, or the plogspace
   - The create size, which is the minimum size that is used when creating a new chunk in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace that is not a mirror space
     For example, the following command sets the create size and extend size to 60 MB and 10 MB, respectively, for space number 3:

     ```
     EXECUTE FUNCTION task("modify dbspace sp_sizes",
      "3", "60000", "10000");
     ```

After you configure for the automatic expansion of a storage space, you can also manually expand the space or extend a chunk in the space, as necessary.

Drop a chunk

Use the **onspaces** utility to drop a chunk from a dbspace.

Before you drop a chunk, ensure that the database server is in the correct mode, using the following table as a guideline.

Table 1. Database server modes for dropping chunks

Chunk type	Database server in online mode	Database server in administration or quiescent mode	Database server in offline mode
Dbspace chunk	Yes	Yes	No
Temporary dbspace chunk	Yes	Yes	No
Blobspace chunk	No	Yes	No
Sbspace or temporary sbspace chunk	Yes	Yes	No

- Verify whether a chunk is empty
- Drop a chunk from a dbspace with onspaces
- Drop a chunk from a blobspace
- Drop a chunk from an sbspace with onspaces

Verify whether a chunk is empty

To drop a chunk successfully from a dbspace with either of these utilities, the chunk must not contain any data. All pages other than overhead pages must be freed.

If any pages remain allocated to nonoverhead entities, the utility returns the following error: `Chunk is not empty.`

In addition, when a dbspace consists of two or more chunks and the additional chunks do not contain user data, the additional chunks cannot be deleted if the chunks contain a tblspace **tblspace**.

If you receive the `Chunk is not empty` message, you must determine which table or other entity still occupies space in the chunk by running **oncheck -pe** to list contents of the extent.

Usually, the pages can be removed when you drop the table that owns them. Then reenter the utility command.

Drop a chunk from a dbspace with onspaces

The following example drops a chunk from **dbsp3** on UNIX. An offset of 300 KB is specified.

```
onspaces -d dbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of a dbspace with the syntax in the previous example. Instead, you must drop the dbspace. Use the **fchunk** column of **onstat -d** to determine which is the initial chunk of a dbspace. For more information about **onstat**, see information about the **onspaces** utility in the *IBM® Informix® Administrator's Reference*.

For information about dropping a chunk from a dbspace with **onspaces**, see the *IBM Informix Administrator's Reference*.

Drop a chunk from a blobspace

The procedure for dropping a chunk from a blobspace is identical to the procedure for dropping a chunk from a dbspace described in Drop a chunk from a dbspace with onspaces except that the database server must be in quiescent or administration mode. Other than this condition, you must substitute the name of your blobspace wherever a reference to a dbspace occurs.

Drop a chunk from an sbspace with onspaces

The following example drops a chunk from **sbsp3** on UNIX. An offset of 300 KB is specified. The database server must be in online administration, or quiescent mode when you drop a chunk from an sbspace or temporary sbspace.

```
onspaces -d sbsp3 -p /dev/raw_dev1 -o 300
```

You cannot drop the initial chunk of an sbspace with the syntax in the previous example. Instead, you must drop the sbspace. Use the **fchunk** column of **onstat -d** to determine which chunk is the initial chunk of an sbspace.

- The -f (force) option
- Delete smart large objects without any pointers

The -f (force) option

You can use the **-f** option of **onspaces** to drop an sbspace chunk without metadata allocated in it. If the chunk contains metadata for the sbspace, you must drop the entire sbspace. Use the **Chunks** section of **onstat -d** to determine which sbspace chunks contain metadata.

```
onspaces -d sbsp3 -f
```

Warning: If you force the drop of an sbspace, you might introduce consistency problems between tables and sbspaces.

Delete smart large objects without any pointers

Each smart large object has a reference count, the number of pointers to the smart large object. When the reference count is greater than 0, the database server assumes that the smart large object is in use and does not delete it.

Rarely, a smart large object with a reference count of 0 remains. You can use the **onspaces -cl** command to delete all smart large objects that have a reference count of 0, if it is not open by any application.

For information about using **onspaces -cl**, see information about the **onspaces** utility in the *IBM® Informix® Administrator's Reference*.

Drop a storage space

Use **onspaces** to drop a dbspace, temporary dbspace, blobspace, sbspace, temporary sbspace, or extspace.

On UNIX, you must be logged in as root or **informix** to drop a storage space. On Windows, you must be a member of the **Informix-Admin** group to drop a storage space.

You can drop a storage space only when the database server is in online, administration, or quiescent mode.

Preparation for dropping a storage space

Before you drop a dbspace, you must first drop all databases and tables that you previously created in that dbspace. You cannot drop the root dbspace.

Before you drop a blobspace, you must drop all tables that have a TEXT or BYTE column that references the blobspace.

Run **oncheck -pe** to verify that no tables or log files are located in the dbspace or blobspace.

Before you drop an sbspace, you must drop all tables that have a CLOB or BLOB column that reference objects that are stored in the sbspace. For sbspaces, you are not required to delete columns that point to an sbspace, but these columns must be null; that is, all smart large objects must be deallocated from the sbspace.
Tip: If you drop tables on dbspaces where light appends are occurring, the light appends might be slower than you expect. The symptom of this problem is physical logging activity. If light appends are slower than you expect, make sure that no tables are dropped in the dbspace either before or during the light appends. If you have dropped tables, force a checkpoint with **onmode -c** before you perform the light append.
Important: Dropping a chunk or a dbspace triggers a blocking checkpoint, which forces all database updates to wait while all the buffer pools are flushed to disk. This update blocking can be significantly longer during a blocking checkpoint than during a non-blocking checkpoint, especially if the buffer pool is large.

Drop a mirrored storage space

If you drop a storage space that is mirrored, the mirror spaces are also dropped.

If you want to drop only a storage-space mirror, turn off mirroring. (See End mirroring.) This action drops the dbspace, blobspace, or sbspace mirrors and frees the chunks for other uses.

Drop a storage space with onspaces

To drop a storage space with **onspaces**, use the **-d** option as illustrated in the following examples.

This example drops a dbspace called **dbspce5** and its mirrors.

```
onspaces -d dbspce5
```

This example drops a dbspace called **blobsp3** and its mirrors.

```
onspaces -d blobsp3
```

Use the **-d** option with the **-f** option if you want to drop an sbspace that contains data. If you omit the **-f** option, you cannot drop an sbspace that contains data. This example drops an sbspace called **sbspc4** and its mirrors.

```
onspaces -d sbspc4 -f
```

Warning: If you use the **-f** option, the tables in the database server might have dead pointers to the deleted smart large objects.
For information about dropping a storage space with **onspaces**, see information about the **onspaces** utility in the *IBM® Informix® Administrator's Reference*.

Back up after dropping a storage space

If you create a storage space with the same name as the deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see the *IBM® Informix® Backup and Restore Guide*.
Important: After you drop a dbspace, blobspace, or sbspace, the newly freed chunks are available for reassignment to other dbspaces, blobspaces, or sbspaces. However, before you reassign the newly freed chunks, you must perform a level-0 backup of the root dbspace and the modified storage space. If you do not perform this backup, and you subsequently must perform a restore, the restore might fail because the backup reserved pages are not up-to-date.

Creating a space or chunk from the storage pool

If your storage pool contains entries, you can create storage spaces or chunks from free space in the storage pool.

Prerequisite: The storage pool must contain entries (a directory, cooked file, or raw device).

To create a storage space or chunk from the storage pool:

Run the admin() or task() function with one of the following arguments for creating a space from the storage pool. The elements you use in the command vary, depending on the type of space that you are creating.

- ```
  EXECUTE FUNCTION task("create dbspace from storagepool", "space_name",
   "size", "page_size", "mirroring_flag", "first_extent", "next_extent");
  ```
- ```
  EXECUTE FUNCTION task("create tempdbspace from storagepool",
   "space_name", "size", "page_size");
  ```
- ```
  EXECUTE FUNCTION task("create blobspace from storagepool", "space_name",
   "size", "page_size", "mirroring_flag",);
  ```
- ```
  EXECUTE FUNCTION task("create sbspace from storagepool", "space_name",
   "size", "log_flag", "mirroring_flag",);
  ```
- ```
  EXECUTE FUNCTION task("create tempsbspace from storagepool",
   "space_name", "size",);
  ```
- ```
  EXECUTE FUNCTION task("create chunk from storagepool",
   "space_name", "size",);
  ```

Examples

The following command creates a mirrored blobspace named `blobspace1`. The new blobspace has a size of 100 gigabytes and a blobpage size of 100 pages.

```
EXECUTE FUNCTION task("create blobspace from storagepool", "blobspace1", "100 GB",
 "100", "1");
```

The following command adds a chunk to the dbspace named `logdbs`. The new chunk has a size of 200 megabytes.

```
EXECUTE FUNCTION task("create chunk from storagepool", "logdbs", "200 MB");
```

Related information:
create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API)
create tempdbspace from storagepool argument: Create a temporary dbspace from the storage pool (SQL administration API)
create blobspace from storagepool argument: Create a blobspace from the storage pool (SQL administration API)
create sbspace from storagepool argument: Create an sbspace from the storage pool (SQL administration API)
create tempsbspace from storagepool argument: Create a temporary sbspace from the storage pool (SQL administration API)
create chunk from storagepool argument: Create a chunk from the storage pool (SQL administration API)

Returning empty space to the storage pool

You can return the space from an empty chunk or storage space to the storage pool.

To return storage space from an empty chunk, dbspace, temporary dbspace, blobspace, sbspace, or temporary sbspace to the storage pool:

Run the admin() or task() function with one of the following arguments for returning space to the storage pool. The elements you use in the command vary, depending on the type of object that you are dropping.

- ```
  EXECUTE FUNCTION task("drop chunk to storagepool", "space_name",
   "chunk_path", "chunk_offset")
  ```
- ```
  EXECUTE FUNCTION task("drop dbspace to storagepool", "space_name");
  ```
- ```
  EXECUTE FUNCTION task("drop tempdbspace to storagepool", "space_name");
  ```
- ```
  EXECUTE FUNCTION task("drop blobspace to storagepool", "space_name");
  ```
- ```
  EXECUTE FUNCTION task("drop sbspace to storagepool", "space_name");
  ```
- ```
  EXECUTE FUNCTION task("drop tempsbspace to storagepool", "space_name");
  ```

Examples

The following command drops an empty blobspace named `blob4` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop blobspace to storagepool", "blob4");
```

The following command drops an empty chunk in a dbspace named `health` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop chunk to storagepool", "health",
 "/health/rawdisk23", "100 KB");
```

Related information:
drop chunk to storagepool argument: Return space from an empty chunk to the storage pool (SQL administration API)
drop dbspace to storagepool argument: Return space from an empty dbspace to the storage pool (SQL administration API)
drop tempdbspace to storagepool argument: Return space from an empty temporary dbspace to the storage pool (SQL administration API)
drop blobspace to storagepool argument: Return space from an empty blobspace to the storage pool (SQL administration API)
drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API)
drop tempsbspace to storagepool argument: Return space from an empty temporary sbspace to the storage pool (SQL administration API)

Manage extspaces

An extspace does not require allocation of disk space. You create and drop extspaces using the **onspaces** utility. For more information about extspaces, see Extspaces.

- Create an extspace
- Drop an extspace

Create an extspace

You create an extspace with the **onspaces** utility. But you must first have a valid data source and a valid access method with which to access that data source. Although you can create an extspace without a valid access method or a valid data source, any attempts to retrieve data from the extspace generate an error. For information about access methods, see the *IBM® Informix® Virtual-Table Interface Programmer's Guide*.

To create an extspace with **onspaces**, use the -c option as illustrated in the following example. The following example shows how to create an extspace, **pass_space**, that is associated with the UNIX password file.

```
onspaces -c -x pass_space -l /etc/passwd
```

Specify an extspace name of up to 128 bytes. The name must be unique and begin with a letter or underscore. You can use letters, digits, underscores, and **$** characters in the name.
Important: The preceding example assumes that you have coded a routine that provides functions for correctly accessing the file passwd and that the file itself exists. After you have created the extspace, you must use the appropriate commands to allow access to the data in the file passwd. For more information about user-defined access methods, see the *IBM Informix Virtual-Table Interface Programmer's Guide*.
For reference information about creating an extspace with **onspaces**, see information about the **onspaces** utility in the *IBM Informix Administrator's Reference*.

Drop an extspace

To drop an extspace with **onspaces**, use the **-d** option as illustrated in the following examples. An extspace cannot be dropped if it is associated with an existing table or index.

This example drops an extspace called **pass_space**.

```
onspaces -d pass_space
```


Skip inaccessible fragments

One benefit that fragmentation provides is the ability to skip table fragments that are unavailable during an I/O operation. For example, a query can proceed even when a fragment is located on a chunk that is currently down as a result of a disk failure. When this situation occurs, a disk failure affects only a portion of the data in the fragmented table. By contrast, tables that are not fragmented can become completely inaccessible if they are located on a disk that fails.

This function is controlled as follows:

- By the database server administrator with the DATASKIP configuration parameter
- By individual applications with the SET DATASKIP statement

The DATASKIP configuration parameter

You can set the DATASKIP parameter to OFF, ALL, or ON *dbspace_list.* OFF means that the database server does not skip any fragments. If a fragment is unavailable, the query returns an error. ALL indicates that any unavailable fragment is skipped. ON *dbspace_list* instructs the database server to skip any fragments that are located in the specified dbspaces.

The dataskip feature of onspaces

Use the dataskip feature of the onspaces utility to specify the dbspaces that are to be skipped when they are unavailable. For example, the following command sets the DATASKIP parameter so that the database server skips the fragments in **dbspace1** and **dbspace3**, but not in **dbspace2**:

```
onspaces -f ON dbspace1 dbspace3
```

For the complete syntax of this **onspaces** option, see information about the **onspaces** utility in the *IBM® Informix® Administrator's Reference*.

Use onstat to check dataskip status

Use the **onstat** utility to list the dbspaces currently affected by the dataskip feature. The **-f** option lists both the dbspaces that were set with the DATASKIP configuration parameter and the **-f** option of the **onspaces** utility.

When you run **onstat -f**, you receive a message that tells you whether the DATASKIP configuration parameter is set to `on` for all dbspaces, `off` for all dbspaces, or `on` for specific dbspaces.

The SQL statement SET DATASKIP

An application can use the SQL statement SET DATASKIP to control whether a fragment is skipped if it is unavailable. Applications must include this statement only in limited circumstances, because it causes queries to return different results, depending on the availability of the underlying fragments. Like the configuration parameter DATASKIP, the SET DATASKIP statement accepts a list of dbspaces that indicate to the database server which fragments to skip. For example, suppose that an application programmer included the following statement at the beginning of an application:

```
SET DATASKIP ON dbspace1, dbspace5
```

This statement causes the database server to skip **dbspace1** or **dbspace5** whenever both of these conditions are met:

- The application attempts to access one of the dbspaces.
- The database server finds that one of the dbspaces is unavailable.

If the database server finds that both **dbspace1** and **dbspace5** are unavailable, it skips both dbspaces.

A database server administrator can use the DEFAULT setting for the SET DATASKIP statement to control the dataskip feature. Suppose that an application developer includes the following statement in an application:

```
SET DATASKIP DEFAULT
```

When a query is run subsequent to this SQL statement, the database server checks the value of the configuration parameter DATASKIP. A database server administrator can encourage users to use this setting to specify which dbspaces are to be skipped as soon as the database server administrator becomes aware that one or more dbspaces are unavailable.

Effect of the dataskip feature on transactions

If you turn the dataskip feature on, a SELECT statement always executes. In addition, an INSERT statement always succeeds if the table is fragmented by round-robin and at least one fragment is online. However, the database server does not complete operations that write to the database if a possibility exists that such operations might compromise the integrity of the database. The following operations fail:

- All UPDATE and DELETE operations where the database server cannot eliminate the down fragments
  If the database server can eliminate the down fragments, the update or delete is successful, but this outcome is independent of the DATASKIP setting.
- An INSERT operation for a table fragmented according to an expression-based distribution scheme where the appropriate fragment is down
- Any operation that involves referential constraint checking if the constraint involves data in a down fragment
  For example, if an application deletes a row that has child rows, the child rows must also be available for deletion.
- Any operation that affects an index value (for example, updates to a column that is indexed) where the index in question is located in a down chunk

Determine when to use dataskip

Use this feature sparingly and with caution because the results are always suspect. Consider using it in the following situations:

- You can accept the compromised integrity of transactions.
- You can determine that the integrity of the transaction is not compromised.

The latter task can be difficult and time consuming.

- Determine when to skip selected fragments
- Determine when to skip all fragments

Determine when to skip selected fragments

In certain circumstances, you might want the database server to skip some fragments, but not others. This usually occurs in the following situations:

- Fragments can be skipped because they do not contribute significantly to a query result.
- Certain fragments are down, and you decide that skipping these fragments and returning a limited amount of data is preferable to canceling a query.

When you want to skip fragments, use the ON *dbspace-list* setting to specify a list of dbspaces with the fragments that the database server must skip.

Determine when to skip all fragments

Setting the DATASKIP configuration parameter to ALL causes the database server to skip all unavailable fragments. Use this option with caution. If a dbspace becomes unavailable, all queries initiated by applications that do not issue a SET DATASKIP OFF statement before they execute can be subject to errors.

Monitor fragmentation use

The database administrator might find the following aspects of fragmentation useful to monitor:

- Data distribution over fragments
- I/O request balancing over fragments
- The status of chunks that contain fragments

The administrator can monitor the distribution of data over table fragments. If the goal of fragmentation is improved administration response time, it is important for data to be distributed evenly over the fragments. To monitor fragmentation disk use, you must monitor database server tblspaces, because the unit of disk storage for a fragment is a tblspace. (For information about how to monitor the data distribution for a fragmented table, see Monitor tblspaces and extents.)

The administrator must monitor I/O request queues for data that is contained in fragments. When I/O queues become unbalanced, the administrator must work with the DBA to tune the fragmentation strategy. (For an explanation of how to monitor chunk use, including the I/O queues for each chunk, see Monitor chunks.)

The administrator must monitor fragments for availability and take appropriate steps when a dbspace that contains one or more fragments fails. For how to determine if a chunk is down, see Monitor chunks.

Display databases

You can display the databases that you create with SMI tables.

- SMI tables

Related concepts:
Databases

SMI tables

Query the **sysdatabases** table to display a row for each database managed by the database server. For a description of the columns in this table, see the **sysdatabases** information in the topics about the **sysmaster** database in the *IBM® Informix® Administrator's Reference*.

Monitor disk usage

These topics describe methods of tracking the disk space used by various database server storage units.

For background information about internal database server storage units mentioned in this section, see the chapter about disk structures and storage in the *IBM® Informix® Administrator's Reference*.

- Monitor chunks
- Monitor tblspaces and extents
- Monitor simple large objects in a blobspace
- Monitor sbspaces

Related concepts:
Control of where simple large object data is stored

Monitor chunks

You can monitor chunks for the following information:

- Chunk size
- Number of free pages
- Tables within the chunk

You can use this information to track the disk space used by chunks, monitor chunk I/O activity, and check for fragmentation.

- The onstat -d utility
- The onstat -d update option
- The onstat -D option
- Monitor chunk I/O activity with the onstat -g iof command
  Use the **onstat -g iof** command to monitor chunk I/O activity, including the distribution of I/O requests against the different fragments of a fragmented table.
- The oncheck -pr command
- The oncheck -pe command
- SMI tables

The onstat -d utility

The **onstat -d** utility lists all dbspaces, blobspaces, and sbspaces and the following information for the chunks within those spaces.

- The address of the chunk
- The chunk number and associated dbspace number
- The offset into the device (in pages)
- The size of the chunk (in pages)
- The number of free pages in the chunk
- The path name of the physical device

If you issue the **onstat -d** command on an instance with blobspace chunks, the number of free pages shown is out of date. The tilde (~) that precedes the `free` value indicates that this number is approximate. The **onstat -d** command does not register a blobpage as available until the logical login which a deletion occurred is backed up and the blobpage is freed. Therefore, if you delete 25 simple large objects and immediately run **onstat -d**, the newly freed space is not in the **onstat** output.

To obtain an accurate number of free blobpages in a blobspace chunk, issue the **onstat -d update** command. For details, see The onstat -d update option.

In **onstat -d update** output, the **flags** column in the **chunk** section provides the following information:

- Whether the chunk is the primary chunk or the mirror chunk
- Whether the chunk is online, is down, is being recovered, or is a new chunk

For an example of **onstat -d** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

Important: You must perform a level-0 backup of the root dbspace and the modified dbspace before mirroring can become active and after turning off mirroring.

The onstat -d update option

The **onstat -d update** option displays the same information as **onstat -d** and an accurate number of free blobpages for each blobspace chunk.

The onstat -D option

The **onstat -D** option displays the same information as **onstat -d**, plus the number of pages read from the chunk (in the **page Rd** field).

Monitor chunk I/O activity with the onstat -g iof command

Use the **onstat -g iof** command to monitor chunk I/O activity, including the distribution of I/O requests against the different fragments of a fragmented table.

The **onstat -g iof** command displays:

- The number of reads from each chunk and the number of writes to each chunk
- I/0 by service level, broken down by individual operation
- The type of operation
- The number of times the operation occurred
- The average time the operation took to complete

If one chunk has a disproportionate amount of I/O activity against it, this chunk might be a system bottleneck.

For an example of **onstat -g iof** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

The oncheck -pr command

The database server stores chunk information in the reserved pages PAGE_1PCHUNK and PAGE_2PCHUNK.

To list the contents of the reserve pages, run **oncheck -pr**. The following example shows sample output for **oncheck -pr**. This output is essentially the same as the **onstat -d** output; however, if the chunk information has changed since the last checkpoint, these changes are not in the **oncheck -pr** output.

```
Validating PAGE_1DBSP & PAGE_2DBSP...
          Using dbspace page PAGE_2DBSP.

    DBspace number                  1
    DBspace name                    rootdbs
    Flags                           0x20001          No mirror chunks
    Number of chunks                2
    First chunk                     1
    Date/Time created               07/28/2008 14:46:55
    Partition table page number     14
    Logical Log Unique Id           0
    Logical Log Position            0
    Oldest Logical Log Unique Id    0
    Last Logical Log Unique Id      0
    Dbspace archive status          No archives have occurred
.
.
Validating PAGE_1PCHUNK & PAGE_2PCHUNK...
          Using primary chunk page PAGE_2PCHUNK.

    Chunk number                    1
    Flags                           0x40       Chunk is online
    Chunk path                      /home/server/root_chunk
    Chunk offset                    0 (p)
    Chunk size                      75000 (p)
    Number of free pages            40502
    DBSpace number                  1
.
```

.
.

The oncheck -pe command

To obtain the physical layout of information in the chunk, run **oncheck -pe**. The dbspaces, blobspaces, and sbspaces are listed. The following example shows sample output for **oncheck -pe**.

The following information is displayed:

- The name, owner, and creation date of the dbspace
- The size in pages of the chunk, the number of pages used, and the number of pages free
- A listing of all the tables in the chunk, with the initial page number and the length of the table in pages

The tables within a chunk are listed sequentially. This output is useful for determining chunk fragmentation. If the database server is unable to allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly fragmented.

```
DBSpace Usage Report:  rootdbs          Owner:  informix  Created: 08/08/2006

Chunk  Pathname                                 Size     Used     Free
    1  /home/server/root_chunk                 75000    19420    55580

Description                                           Offset    Size
--------------------------------------------         --------------------
RESERVED PAGES                                             0         12
CHUNK FREELIST PAGE                                       12          1
rootdbs:'informix'.TBLSpace                               13        250
PHYSICAL LOG                                             263       1000
FREE                                                    1263       1500
LOGICAL LOG: Log file 2                                 2763       1500
LOGICAL LOG: Log file 3                                 4263       1500
...
sysmaster:'informix'.sysdatabases                      10263          4
sysmaster:'informix'.systables                         10267          8
...

Chunk  Pathname                                 Size     Used     Free
    2  /home/server/dbspace1                   5000       53     4947

Description                                           Offset    Size
--------------------------------------------         --------------------
RESERVED PAGES                                             0          2
CHUNK FREELIST PAGE                                        2          1
dbspace1:'informix'.TBLSpace                               3         50
FREE                                                      53       4947
```

Related reference:
Change the physical-log location and size

SMI tables

Query the **syschunks** table to obtain the status of a chunk. The following columns are relevant.

chknum
Number of the chunk within the dbspace
dbsnum
Number of the dbspace
chksize
Total size of the chunk in pages
nfree
Number of pages that are free
is_offline
Whether the chunk is down
is_recovering
Whether the chunk is recovering
mis_offline
Whether the mirror chunk is down
mis_recovering
Whether the mirror chunk is being recovered

The **syschkio** table contains the following columns.

pagesread
Number of pages read from the chunk
pageswritten
Number of pages written to the chunk

Monitor tblspaces and extents

Monitor tblspaces and extents to determine disk usage by database, table, or table fragment. Monitoring disk usage by table is particularly important when you are using table fragmentation, and you want to ensure that table data and table index data are distributed appropriately over the fragments.

Run **oncheck -pt** to obtain extent information. The **oncheck -pT** option returns all the information from the **oncheck -pt** option and the additional information about page and index usage.

- SMI tables

SMI tables

Query the **systabnames** table to obtain information about each tblspace. The **systabnames** table has columns that indicate the corresponding table, database, and table owner for each tblspace.

Query the **sysextents** table to obtain information about each extent. The **sysextents** table has columns that indicate the database and the table that the extent belongs to, and the physical address and size of the extent.

Monitor simple large objects in a blobspace

Monitor blobspaces to determine the available space and whether the blobpage size is optimal.

- Determine blobpage fullness with oncheck -pB
- Monitor blobspace usage with oncheck -pe
- Monitor simple large objects in a dbspace with oncheck -pT

Determine blobpage fullness with oncheck -pB

The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobspace.

Run **oncheck -pB** with either a database name or a table name as a parameter. The following example retrieves storage information for all simple large objects stored in the table **sriram.catalog** in the **stores_demo** database:

```
oncheck -pB stores_demo:sriram.catalog
```

For detailed information about interpreting the **oncheck -pB** output, see optimizing blobspace blobpage size in the chapter on table performance considerations in the *IBM® Informix® Performance Guide*.

Monitor blobspace usage with oncheck -pe

The **oncheck -pe** command provides information about blobspace usage:

- Names of the tables that store TEXT and BYTE data, by chunk
- Number of disk pages (not blobpages) used, by table
- Number of free disk pages remaining, by chunk
- Number of overhead pages used, by chunk

The following example shows sample **oncheck -pe** output.

```
BLOBSpace Usage Report:  fstblob          Owner:  informix  Created: 03/01/08
     Chunk: 3   /home/server/blob_chunk                Size     Used     Free
                                                       4000      304     3696

         Disk usage for Chunk 3                    Total Pages
         ------------------------------------------------------
         OVERHEAD                                            8
         stores_demo:chrisw.catalog                        296
         FREE                                             3696
```

Monitor simple large objects in a dbspace with oncheck -pT

Use **oncheck -pT** to monitor dbspaces to determine the number of dbspace pages that TEXT and BYTE data use.

This command takes a database name or a table name as a parameter. For each table in the database, or for the specified table, the database server displays a general tblspace report.

Following the general report is a detailed breakdown of page use in the extent, by page type. See the **Type** column for information about TEXT and BYTE data.

The database server can store more than one simple large object on the same blobpage. Therefore, you can count the number of pages that store TEXT or BYTE data in the tblspace, but you cannot estimate the number of simple large objects in the table.

The following example shows sample output.

```
TBLSpace Usage Report for mydemo:chrisw.catalog

    Type                 Pages      Empty  Semi-Full       Full  Very-Full
    ---------------- ---------- ---------- ---------- ---------- ----------
    Free                     7
    Bit-Map                  1
    Index                    2
    Data (Home)              9
    Data (Remainder)         0          0          0          0          0
    Tblspace BLOBs           5          0          0          1          4
                     ----------
    Total Pages             24

    Unused Space Summary

        Unused data bytes in Home pages               3564
        Unused data bytes in Remainder pages             0
        Unused bytes in Tblspace Blob pages           1430

Index Usage Report for index  111_16 on mydemo:chrisw.catalog

                     Average    Average
    Level    Total No. Keys Free Bytes
    ----- -------- -------- ----------
        1        1       74       1058
    ----- -------- -------- ----------
    Total        1       74       1058

Index Usage Report for index  111_18 on mydemo:chrisw.catalog

                     Average    Average
    Level    Total No. Keys Free Bytes
    ----- -------- -------- ----------
        1        1       74        984
    ----- -------- -------- ----------
    Total        1       74        984
```


Monitor sbspaces

One of the most important areas to monitor in an sbspace is the metadata page use. When you create an sbspace, you specify the size of the metadata area. Also, any time that you add a chunk to the sbspace, you can specify that metadata space be added to the chunk.

If you attempt to insert a new smart large object, but no metadata space is available, you receive an error. The administrator must monitor metadata space availability to prevent this situation from occurring.

Use the following commands to monitor sbspaces.

Command	Description
onstat -g smb s	Displays the storage attributes for all sbspaces in the system: • sbspace name, flags, owner • Logging status • Average smart-large-object size • First extent size, next extent size, and minimum extent size • Maximum I/O access time • Lock mode

Command	Description
onstat -g smb c	Displays the following information for each sbspace chunk: • Chunk number and sbspace name • Chunk size and path name • Total user data pages and free user data pages • Location and number of pages in each user-data and metadata areas
oncheck -ce oncheck -pe	Displays the following information about sbspace use: • Names of the tables that store smart-large-object data, by chunk • Number of disk pages (not sbpages) used, by table • Number of free user-data pages that remain, by chunk • Number of reserved user-data pages that remain, by chunk • Number of metadata pages used, by chunk The output provides the following totals: • Total number of used pages for all user-data areas and metadata area. The system adds 53 pages for the reserved area to the totals for the user-data area and metadata area. • Number of free pages that remain in the metadata area • Number of free pages that remain in all user-data areas
onstat -d	Displays the following information about the chunks in each sbspace: • Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data areas • Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data areas
oncheck -cs oncheck -ps	Validates and displays information about the metadata areas for sbspaces..
oncheck -cS	Displays information about smart-large-object extents and user-data areas for sbspaces.
oncheck -pS	Displays information about smart-large-object extents, user-data areas, and metadata areas for sbspaces. For more information about **oncheck -cS** and **-pS**, see managing sbspaces in the topics on table performance considerations in your *IBM® Informix® Performance Guide*.

- The onstat -d option
- The oncheck -ce and oncheck -pe options
- The oncheck -cs option
- The oncheck -ps option
- Monitoring the metadata and user-data areas

Related tasks:
Adding a chunk to an sbspace
Monitoring the metadata and user-data areas
Related reference:
The oncheck -ce and oncheck -pe options
The onstat -d option
The oncheck -ps option
The oncheck -cs option
Related information:
onstat -g smb command: Print sbspaces information

The onstat -d option

Use the **onstat -d** option to display the following information about the chunks in each sbspace:

- Number of free sbpages in each sbspace chunk, in the metadata area, and in the user-data area
- Total number of sbpages in each sbspace chunk, in the metadata area, and in the user-data area

For an example of **onstat -d** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

To find out the total amount of used space, run the **oncheck -pe** command. For more information, see The oncheck -ce and oncheck -pe options.

The **onstat -d** option does not register an sbpage as available until the logical login which a deletion occurred is backed up and the sbpage is freed. Therefore, if you delete 25 smart large objects and immediately run **onstat -d**, the newly freed space is not in the **onstat** output.

Related reference:
Monitor sbspaces

The oncheck -ce and oncheck -pe options

Run **oncheck -ce** to display the size of each sbspace chunk, the total amount of used space, and the amount of free space in the user-data area. The **oncheck -pe** option displays the same information as **oncheck -ce** plus a detailed listing of chunk use. First the dbspaces are listed and then the sbspaces. The **-pe** output provides the

following information about sbspace use:

- Names of the tables that store smart-large-object data, by chunk
- Number of disk pages (not sbpages) used, by table
- Number of free user-data pages that remain, by chunk
- Number of metadata pages used, by chunk

The output provides the following totals:

- Total number of used pages for the user-data area, metadata area, and reserved area
  The system adds 53 extra pages for the reserved area to the totals for the user-data area and metadata area.
- Number of free pages that remain in the metadata area
- Number of free pages that remain in the user-data area

Tip: The **oncheck -pe** option provides information about sbspace use in terms of database server pages, not sbpages.
The following example shows sample output. In this example, the sbspace **s9_sbspc** has a total of 214 used pages, 60 free pages in the metadata area, and 726 free pages in the user-data area.

```
Chunk Pathname                                    Size      Used      Free
      2 /ix/ids9.2/./s9_sbspc                       1000       940        60

    Description                                     Offset     Size
    ------------------------------------------------ -------- --------
    RESERVED PAGES                                         0        2
    CHUNK FREELIST PAGE                                    2        1
    s9_sbspc:'informix'.TBLSpace                           3       50
    SBLOBSpace LO [2,2,1]                                 53        8
    SBLOBSpace LO [2,2,2]                                 61        1
...
SBLOBSpace LO [2,2,79]                                   168        1
    SBLOBSpace FREE USER DATA                            169      305
    s9_sbspc:'informix'.sbspace_desc                     474        4
    s9_sbspc:'informix'.chunk_adjunc                     478        4
    s9_sbspc:'informix'.LO_hdr_partn                     482        8
    s9_sbspc:'informix'.LO_ud_free                       490        5
    s9_sbspc:'informix'.LO_hdr_partn                     495       24
    FREE                                                 519       60
    SBLOBSpace FREE USER DATA                            579      421

    Total Used:                                  214
    Total SBLOBSpace FREE META DATA:              60
    Total SBLOBSpace FREE USER DATA:             726
```

You can use CHECK EXTENTS as the SQL administration API *command* equivalent to **oncheck -ce**. For information about using SQL API commands, see Remote administration with the SQL administration API and the *IBM® Informix® Administrator's Reference*.

Related reference:
Monitor sbspaces

The oncheck -cs option

The **oncheck -cs** and the **oncheck -Cs** options validate the metadata area of an sbspace. The following example shows an example of the **-cs** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** checks and displays the metadata for all sbspaces.

Use the **oncheck -cs** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area. To find the number of used pages in the metadata area, total the numbers in the **Used** column. To find the number of free pages in the metadata area, total the numbers in the **Free** column.

For example, based on the field values displayed in the following figure, the total number of used pages in the metadata area for **s9_sbspc** is 33 2 KB pages (or 66 KB). The metadata area contains a total of 62 free pages (or 124 KB).

```
Validating space 's9_sbspc' ...

SBLOBspace Metadata Partition                Partnum      Used      Free
s9_sbspc:'informix'.TBLSpace                 0x200001        6        44
s9_sbspc:'informix'.sbspace_desc             0x200002        2         2
s9_sbspc:'informix'.chunk_adjunc             0x200003        2         2
s9_sbspc:'informix'.LO_hdr_partn             0x200004       21        11
s9_sbspc:'informix'.LO_ud_free               0x200005        2         3
```

Related reference:
Monitor sbspaces

The oncheck -ps option

The **oncheck -ps** option validates and displays information about the metadata areas in sbspace partitions. The following example shows an example of the **-ps** output for **s9_sbspc**. If you do not specify an sbspace name on the command line, **oncheck** validates and displays tblspace information for all storage spaces.

To monitor the amount of free metadata space, run the following command:

```
oncheck -ps spacename
```

The **-ps** output includes information about the locking granularity, **partnum**, number of pages allocated and used, extent size, and number of rows in the metadata area. Use the **oncheck -ps** output to see how much space is left in the metadata area. If it is full, allocate another chunk with adequate space for the metadata area.

If you run **oncheck -ps** for the dbspace that contains the tables where the smart large objects are stored, you can find the number of rows in the table.

```
Validating space 's9_sbspc' ...

TBLSpace Report for
    TBLspace Flags                  2801       Page Locking
                                               TBLspace use 4 bit bit-maps
                                               Permanent System TBLspace

    Partition partnum               0x200001
    Number of rows                  92
    Number of special columns       0
    Number of keys                  0
    Number of extents               1
    Current serial value            1
    First extent size               50
    Next extent size                50
    Number of pages allocated       50
    Number of pages used            6
    Number of data pages            0
    Number of rows                  0
    Partition lockid                2097153

    Current SERIAL8 value           1
    Current REFID value             1
    Created                         Thu May 24 14:14:33 2007
```

Related reference:
Monitor sbspaces

Monitoring the metadata and user-data areas

The database server reserves 40 percent of the user-data area as a *reserved area*. The database server uses this reserved space for either the metadata or user data. The metadata area gets used up as smart large objects are added to that sbspace. When the database server runs out of metadata or user-data space, it moves a block of the reserved space to the corresponding area.

When all of the reserve area is used up, the database server cannot move space to the metadata area, even if the user-data area contains free space.

1. As you add smart large objects to the sbspace, use **oncheck -pe** or **onstat -g smb c** to monitor the space in the metadata area, user-data area, and reserved area.
   For an example, see The oncheck -ce and oncheck -pe options.
2. Use the message log to monitor metadata stealing.
   The database server prints messages about the number of pages allocated from the reserved area to the metadata area.
3. Add another chunk to the sbspace before the sbspace runs out of space in the metadata and reserved areas.
   For more information, see Adding a chunk to an sbspace.
4. The database server writes the FREE_RE and CHKADJUP log records when it moves space from the reserve area to the metadata or user-data area.

For more information, see Size sbspace metadata.

Related reference:
Monitor sbspaces

Multitenancy

You can segregate data, storage space, and processing resources for multiple client organizations by creating multiple tenant databases in a single instance of Informix®.

For example, assume that you want to provide payroll services to small businesses. You sell the use of the payroll application as a service to small business clients. Instead of providing a separate Informix instance to each client, you can configure a tenant database for each client in a single Informix instance.

When you configure multitenancy, you segregate the following aspects of a database server:

Data
: You create a separate tenant database for each client.

Storage spaces
: Each tenant database has dedicated storage spaces to store data. Tables, fragments, and indexes that are created in the tenant database must be created in the dedicated storage spaces. Only the tenant database can use the dedicated storage spaces.
  You can limit the amount of permanent storage space that is available to a tenant database to conserve system resources.

  Temporary storage spaces can be dedicated to a specific tenant database or shared between databases.

You can encrypt tenant storage spaces if the DISK_ENCRYPTION configuration parameter is set. Each encrypted storage space has a separate encryption key.

Users
: You can set permissions for client users to access each tenant database. You can grant certain users permission to create, modify, or drop tenant databases. By default, only a DBA or user **informix** can create a tenant database.

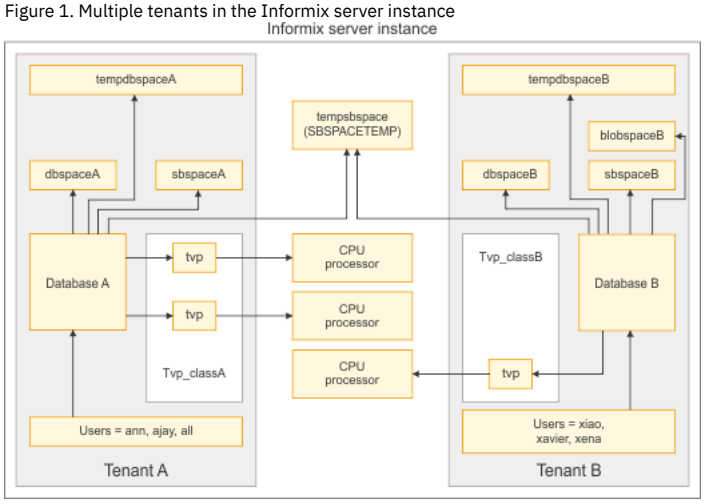
Processing resources
: You can segregate CPU resources for a tenant database by defining a tenant virtual processor class and creating virtual processors for running the session threads for the tenant database. Otherwise, the session threads for tenant databases have access to all CPU virtual processors.

Session limits
: You can set the following limits for tenant sessions:

- The number of locks a tenant session can acquire.
- The amount of memory that can be allocated for a session.
- The amount of temporary storage space that can be allocated for a session.
- The size of transactions within a session, based on the amount of log space that individual transactions would fill.
- The amount of time that a transaction is allowed to run within a session.
- The amount of shared memory for all sessions that are connected to the tenant database.
- The number of client connections to a tenant database.

The following illustration shows a possible configuration for two clients in the Informix server instance. Each client has a database and users who are allowed to access the tenant database. Each tenant database has their own storage spaces. Both tenant databases share the default temporary sbspace. Tenant A has a tenant virtual processor class with two virtual processors, while Tenant B has a virtual process class with one virtual processor.

Figure 1. Multiple tenants in the Informix server instance



Replication and tenant databases

You can replicate tenant databases with Enterprise Replication and high-availability clusters.

You can run the commands to create, modify, or delete tenant databases through an Enterprise Replication grid.

You cannot run the commands to create, modify, or delete tenant databases from an updatable secondary server in a high-availability cluster.

Backup and restore of tenant databases

You can back up tenant databases as part of a database server backup or by specifying the tenant storage spaces in the backup command. You can restore a single tenant database with ON-Bar by specifying the **-T** option in the **onbar -r** command.

If storage space encryption is enabled, you can encrypt all storage spaces that are assigned to a tenant during a restore. Whether storage space encryption is enabled or not enabled, you can decrypt all tenant storage spaces during a restore.

- Creating a tenant database
  You can create a tenant database to segregate data, storage, and processing resources to a specific client organization.
- Managing tenant databases
  You can view the properties of tenant databases, update the properties of tenant databases, and delete tenant databases.
- Restoring a tenant database to a point in time
  You can restore a tenant database to a point in time with the ON-Bar utility.

Related concepts:
Storage space creation and management
Tenant virtual processor class
Related information:
Storage space encryption
DISK_ENCRYPTION configuration parameter

Creating a tenant database

You can create a tenant database to segregate data, storage, and processing resources to a specific client organization.

You must be user **informix**, a DBSA, or have the TENANT privilege to create a tenant database.
You cannot convert an existing database to a tenant database. You cannot convert a tenant database to a non-tenant database. You cannot run the CREATE DATABASE statement to create a tenant database.
To create a tenant database:

1. Create the storage spaces for the tenant database. All dedicated storage spaces must be empty when you create the tenant database. You can create the following types of dedicated spaces for a tenant database:

   dbspaces
   : You must create at least one dbspace for the tenant database. The tenant database must be stored in one or more dedicated dbspaces.

   blobspaces
   : If the tenant database will contain simple large objects, you must create one or more blobspaces.

   sbspaces
   : If the tenant database will contain smart large objects, you must create one or more sbspaces. Smart large objects can include BLOB or CLOB data, or data and table statistics that are too large to fit in a row. Some Informix® features, such as Enterprise Replication, spatial data, and basic text searching, require sbspaces.

   temporary dbspaces
   : Optional: Create one or more temporary dbspaces to store temporary tables. Otherwise, temporary tables are stored in the temporary dbspaces that are specified by the DBSPACETEMP configuration parameter or environment variable.

   temporary sbspaces
   : Optional: Create one or more temporary sbspaces to store temporary smart large objects. Otherwise, temporary smart large objects are stored in the temporary sbspaces that are specified by the SBSPACETEMP configuration parameter.

2. Optional: Set limits for the tenant database so that it cannot monopolize system resources. Tenant database limits do not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. You can set the following limits for a tenant database:

   Locks available to a session
   : Set the **session_limit_locks** property to specify the maximum number of locks available to a session.

   Logspace available to transactions in a session
   : Set the **session_limit_logspace** property to specify the maximum amount of log space that a session can use for individual transactions.

   Memory available to a session
   : Set the **session_limit_memory** property to specify the maximum amount of memory that a session can allocate.

   Temporary table space available to a session
   : Set the **session_limit_tempspace** property to specify the maximum amount of temporary table space that a session can allocate.

   Amount of time that a transaction can run
   : Set the **session_limit_txn_time** property to specify the maximum amount of time that a transaction can run in a session

   Total space available to a tenant database
   : Set the **tenant_limit_space** property to specify the maximum amount of storage space available to a tenant user.

3. Optional: Set up a storage pool so that storage spaces can grow automatically. You can specify maximum sizes for extendable storage spaces to limit the growth of tenant databases.
4. Optional: Provide TENANT privileges to specific users to create, modify, and delete tenant databases. For example, the following command gives the user **jsmith** TENANT privileges:

   ```
   EXECUTE FUNCTION task("grant admin", "jsmith", "tenant");
   ```

5. Create a tenant database and define its properties by running the **admin()** or **task()** SQL administration API function with the **tenant create** argument. For example, the following statement creates a tenant database that is named **companyA**:

   ```
   EXECUTE FUNCTION task('tenant create', 'companyA',
         '{dbspace:"companyA_dbs1,companyA_dbs2", sbspace:"companyA_sbs1",
           vpclass:"tvp_A,num=2", logmode:"ansi"}');
   ```

   The tenant database has two dbspaces, an sbspace, two tenant virtual processors, and the ANSI logging mode.

When you explicitly specify storage locations during the creation or altering of tables and indexes in the tenant database, you must specify the dbspaces that are listed in the tenant database definition. Otherwise, the statement fails. If you do not explicitly specify storage for tables or indexes, they are created in the first dbspace that is listed in the tenant definition.

Note: Improve the security of your databases by performing the following tasks:

- Run GRANT and REVOKE statements to control user access to databases.
- Set the DBCREATE_PERMISSION configuration parameter to restrict the ability to create non-tenant databases.

Related concepts:
Automatic space management
Related information:
tenant create argument: Create a tenant database (SQL Administration API)
The tenant table

Managing tenant databases

You can view the properties of tenant databases, update the properties of tenant databases, and delete tenant databases.

Viewing tenant database properties

To view the tenant database definition, query the **tenant** table in the **sysadmin** database. For example, the following statement lists the tenant databases and their properties:

```
SELECT hex(tenant_id),tenant_dbsname,tenant_resources::json,
       tenant_create_time,tenant_last_updated
FROM tenant;
```

The **tenant_resources** column, which contains the tenant properties, is of type BSON, so you must cast the column to JSON to view the properties.

Updating the properties of tenant databases

To update properties, run the **admin()** or **task()** SQL administration API function with the **tenant update** argument. The updates take effect for new sessions.

You can append dbspaces, blobspaces, or sbspaces to the existing lists of storage spaces for a tenant database. The storage spaces must be empty. You must have DBA or TENANT privileges to change tenant database properties.

You cannot remove dedicated storage spaces from a tenant database unless you delete the database.

When you specify new values for the following tenant database properties, existing values are replaced.

- **dbspacetemp** (temporary dbspaces that are assigned to the tenant)
- **session_limit_logspace** (limit on log space for individual transactions)
- **session_limit_memory** (limit on memory that is allocated per session)
- **session_limit_tempspace** (limit on temporary table space per session)
- **session_limit_txn_time** (limit on the length of time a transaction can run)
- **sbspacetemp** (temporary sbspaces that are assigned to the tenant)
- **session_limit_locks** (limit on the number of locks per session)
- **tenant_limit_space** (limit on total storage space)
- **vpclass** (virtual processor classes names and quantities)

Deleting tenant databases

To delete a tenant database, run the **admin()** or **task()** SQL administration API function with the **tenant drop** argument. You must have DBA or TENANT privileges to delete tenant databases. You cannot delete a tenant database with the DROP DATABASE statement. All dedicated storage spaces for the tenant database are emptied and become available. Any tenant virtual processors that are not shared with other tenant databases are dropped.

Related information:
tenant update argument: Modify tenant database properties (SQL Administration API)
tenant drop argument: Drop a tenant database (SQL Administration API)
The tenant table

Restoring a tenant database to a point in time

You can restore a tenant database to a point in time with the ON-Bar utility.

You must meet the following prerequisites before you start a tenant restore:

- The tenant database storage spaces must have a physical backup.
- The database server must be online.
- All tenant storage spaces that are listed in the tenant definition must exist.
- Tenant storage spaces must have unique names for existing backups. For example, if a tenant database that has backups is dropped and storage spaces with the same names are used in a new tenant database, the backups of the dropped tenant database must be removed.
- No other warm restores or tenant restores are in progress.

A tenant database point-in-time restore has the following additional prerequisites if the tenant database is in a high-availability cluster:

- The cluster cannot include shared-disk secondary servers or updatable secondary servers. The cluster can include only read-only HDR secondary servers and remote stand-alone secondary servers.
- All secondary servers must be online.
- You can run the restore command only on the primary server.
  However, during the tenant point in time restore process, internally generated commands are run on the secondary servers to restore the new state of the tenant database spaces on the secondaries. A tenant point in time restore command includes new physical backups of the tenant spaces in their new state. The tenant point in time restore command that is run on the primary server does not return until all secondary servers acknowledge the completion of the automatically generated restores of the new tenant space backups.

Like other warm restores, the logical logs that are required for the tenant database restore are restored to the temporary spaces that are specified by the DBSPACETEMP configuration parameter. If the DBSPACETEMP configuration parameter is not set, temporary files are created in the root dbspace.
To restore a tenant database to a point in time, run the following command:

```
onbar -r -T tenant_database -t "time" -O
```

Substitute *tenant_database* with the name of the tenant database. Substitute *time* with the time of the last transaction to be restored from the logical logs.

If you omit the **-O** option, all the permanent tenant spaces must be marked as down before you run the restore command. Temporary spaces are never backed up or restored.

If the restore fails, fix the problem and run the restore again. Until the restore succeeds, the tenant database is blocked from accepting connections. During the restore, the value of the **tenant_state** field is set to `restoring` in the **tenant_resources** column of the **sysadmin:tenant** table. When a tenant database is blocked, the value of the **tenant_state** field is set to `blocked`. You can view the value of the **tenant_state** field by running the following query:

```
SELECT bson_value_lvarchar(tenant_resources, 'tenant_state') AS tenant_state
   FROM sysadmin:tenant
   WHERE tenant_dbsname = 'tenant_dbname';
```

Substitute *tenant_dbname* with the name of the tenant database.

Example

The following command restores a tenant database that is named **tenant1** to the specified point in time:

```
onbar -r -T tenant1 -t "2015-10-10 11:35:57" -O
```

Related information:
The tenant table
onbar -r syntax: Restoring data

Storage optimization

Data compression and consolidation processes can minimize the disk space that is used by your data and indexes.

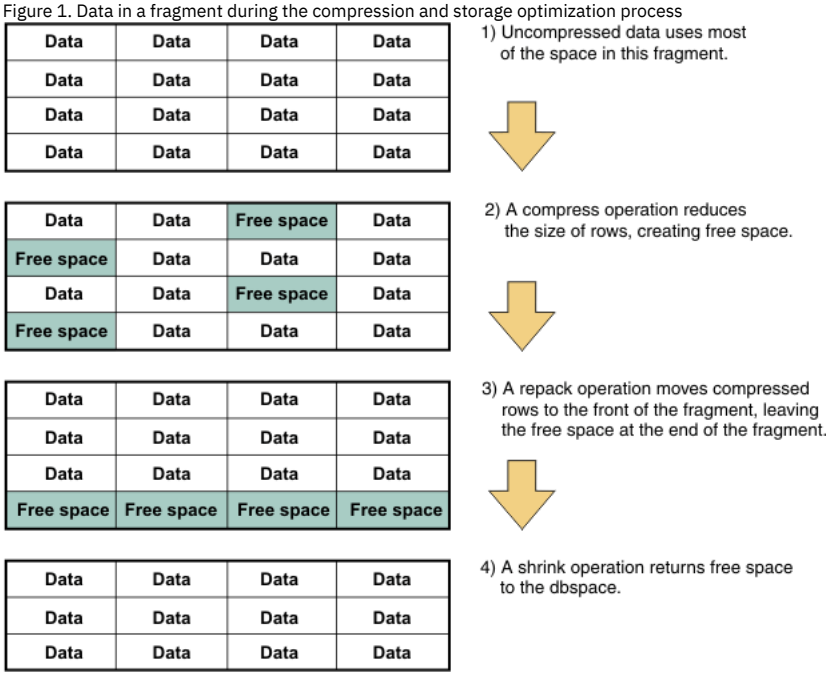
The following table describes the processes that you can use to reduce the amount of disk space that is used by data in rows, simple large objects in dbspaces, and index keys. You can automate any or all of these processes or do them as needed.

Table 1. Storage optimization processes

Storage optimization process	Purpose	When to use
Compressing	Compresses data in tables and fragments, compress simple large objects in dbspaces, and compresses keys in indexes. Reduces the amount of required disk space. After you enable compression, new data or index keys is automatically compressed.	When you want to reduce the size of 2000 or more rows of data, simple large objects in dbspaces, or 2000 or more index keys
Repacking	Consolidates free space in tables, fragments, and indexes.	After you compress or when you want to consolidate free space
Shrinking	Returns free space to the dbspace.	After you compress or repack or when you want to return free space to the dbspace
Defragmenting	Brings data rows or index keys closer together in contiguous, merged extents.	When frequently updated tables or indexes become scattered among multiple non-contiguous extents

The following illustration shows uncompressed data that uses most of the space in a fragment, free space that is created when the data is compressed, free space that is moved to the end of the fragment after a repack operation, and data that remains in the fragment after a shrink operation. The process for storage optimization of indexes is the same.

Figure 1. Data in a fragment during the compression and storage optimization process



- Storage optimization methods
  You can optimize individual tables, fragments, or indexes. You can schedule the automatic optimization of all tables and fragments.
- Scheduling data optimization
  You can configure the automatic compressing, shrinking, repacking, and defragmenting of all tables and extents by enabling the **auto_crsd** Scheduler task.
- Example: Optimizing data storage on demand
  In this example, you learn how to run SQL administration API commands to determine how much space you can save by compressing a table, how to compress the table, and how to optimize storage on demand. You also learn how to uncompress the table and remove the compression dictionaries.
- Partition defragmentation
  You can improve performance by defragmenting partitions to merge non-contiguous extents.
- Compression
  You can compress and uncompress row data in tables and fragments and simple large objects in dbspaces. You can compress B-tree indexes. You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Before you compress data, you can estimate the amount of disk space that you can save.

Related concepts:
Storage space creation and management

Storage optimization methods

You can optimize individual tables, fragments, or indexes. You can schedule the automatic optimization of all tables and fragments.

You can use the COMPRESSED option in the CREATE TABLE statement to enable automatic compression of the table when the table has at least 2000 rows.

You can use the COMPRESSED option in the CREATE INDEX statement to enable automatic compression of the index if the index has 2000 or more keys. Compression is not enabled if the index has fewer than 2000 keys.

You can use the SQL administration API **task** or **admin** function to perform any type of storage optimization on a table, fragment, or index.

You can enable the **auto_crsd** Scheduler task to automatically compress, repack, shrink, and defragment all tables and table fragments.

Table 1. Methods of storage optimization

Goal	SQL statement	SQL administration API argument	Scheduler task	OAT page
Automatically compress data for a table or fragment	CREATE TABLE with the COMPRESSED option	**table compress** or **fragment compress**		Storage
Automatically compress data for all tables and fragments			**auto_crsd**	Server Optimization Policies
Repack and shrink a table or fragment		**table repack shrink** or **fragment repack shrink**		Storage
Automatically repack and shrink all tables and fragments			**auto_crsd**	Server Optimization Policies
Automatically compress a B-tree index	CREATE INDEX with the COMPRESSED option	**index compress**		Storage
Repack and shrink a B-tree index		**index repack shrink**		Storage
Defragment a table of fragment		**defragment**		Storage
Automatically defragment all tables and fragments			**auto_crsd**	Server Optimization Policies

Related concepts:
Data that you can compress
Methods for viewing compression information
Compression
Related information:
COMPRESSED option for tables
COMPRESSED option for indexes
defragment argument: Dynamically defragment partition extents (SQL administration API)
Table and fragment compress and uncompress operations (SQL administration API)
index compress repack shrink arguments: Optimize the storage of B-tree indexes (SQL administration API)

Scheduling data optimization

You can configure the automatic compressing, shrinking, repacking, and defragmenting of all tables and extents by enabling the **auto_crsd** Scheduler task.

You can enable and configure the **auto_crsd** task by updating Scheduler tables in the **sysadmin** database.
To enable the **auto_crsd** task by updating the Scheduler tables:

1. Connect to the **sysadmin** database as user **informix** or another authorized user.
2. Enable the **auto_crsd** Scheduler task by using an UPDATE statement on the **ph_task** table to set the value of the **tk_enable** column to T. For example, the following statement enables the **auto_crsd** task:

```
UPDATE ph_task
  SET tk_enable = 'T'
  WHERE tk_name = 'auto_crsd';
```

3. Optional: Change the frequency of when the task is run by running an UPDATE statement on the **ph_task** table to change the value of the **tk_frequency** column. The default value is 7 00:00:00, which indicates that the task runs once a week. For example, the following statement changes the frequency to once a day:

```
UPDATE ph_task
  SET tk_frequency = '1 00:00:00'
  WHERE tk_name = 'auto_crsd';
```

4. Optional: Disable individual operations by using an UPDATE statement on the **ph_threshold** table to set the **value** column for a threshold to `F`:
   - AUTOCOMPRESS_ENABLED: controls compression
   - AUTOREPACK_ENABLED: controls repacking
   - AUTOSHRINK_ENABLED: controls shrinking
   - AUTODEFRAG_ENABLED: controls defragmenting

   For example, the following statement disables just the defragmentation operation of the **auto_crsd** task:

```
UPDATE ph_threshold
  SET value = 'F'
  WHERE name = 'AUTODEFRAG_ENABLED';
```

5. Optional: Change the thresholds of individual operations by using and UPDATE statement on the **ph_threshold** table to change the value of the **value** column for a threshold:
   - AUTOCOMPRESS_ROWS: The threshold for compression is the number of uncompressed rows. The default threshold is 50 000 rows. A table is compressed when the number of uncompressed rows exceeds 50 000.
   - AUTOREPACK_SPACE: The threshold for repacking a table is the percentage of noncontiguous space. The default is 90%. A table is repacked when more than 90% of the space the table occupies is noncontiguous.
   - AUTOSHRINK_UNUSED: The threshold for shrinking a table or fragment is the percentage of unused, allocated space. The default is 50%. A table or fragment is shrunk when more than 50% of the allocated space is unused.
   - AUTODEFRAG_EXTENTS: The threshold for defragmenting table or fragment extents is the number of extents. The default is 100. A table or fragment is defragmented when the number of extents exceeds 100.

   For example, the following statement changes the compression threshold to 5000 rows:

```
UPDATE ph_threshold
  SET value = '5000'
  WHERE name = 'AUTOCOMPRESS_ROWS';
```

When a threshold for an operation that you enabled is exceeded, the Scheduler runs the operation.

Related concepts:
Partition defragmentation
The Scheduler

Related information:
The Scheduler tables

Example: Optimizing data storage on demand

In this example, you learn how to run SQL administration API commands to determine how much space you can save by compressing a table, how to compress the table, and how to optimize storage on demand. You also learn how to uncompress the table and remove the compression dictionaries.

Assume that you have a table named **rock** in a database named **music** that is owned by user **mario**. The **rock** table is not fragmented. You can run the same operations on a table fragment as you can on a whole table, but the syntax is slightly different.

Prerequisites:

- There must be at least 2,000 rows in each fragment of the table, not just a total of 2,000 rows in the whole table.
- You must be able to connect to the **sysadmin** database (by default only user **informix**), and you must be a DBSA.
- Logical and physical logs are large enough to handle normal processing and compression operations. Compression, repacking, and uncompressing, operations can use large amounts of logs.

To compress both row data and simple large objects in dbspaces:

1. You run the following command to check how much space you might save by compressing the table:

```
EXECUTE FUNCTION task("table estimate_compression", "rock", "music", "mario");
```

   You review the resulting report, which indicates you can save 75 percent of the space that is used by the **rock** table. You decide to compress the table.

2. Before you compress data, you want to create a compression dictionary, which contains information that IBM® Informix® uses to compress data in the **rock** table. You run the following command

```
EXECUTE FUNCTION task("table create_dictionary", "rock", "music", "mario");
```

   Tip: If you do not create the compression dictionary as a separate step, creates the dictionary automatically when you compress data.
3. You decide that you want to compress data in the **rock** table and simple large objects in dbspaces, consolidate the data, and then return the free space to the dbspace. You run the following command:

```
EXECUTE FUNCTION task("table compress repack shrink", "rock", "music", "mario");
```

   You can perform the same operations faster by running them in parallel. You run the following command:

```
EXECUTE FUNCTION task("table compress repack shrink parallel", "rock",
"music", "mario");
```

You can adjust the command by specifying what you want to compress or shrink. For example:

- To compress only row data, specify:

```
EXECUTE FUNCTION task("table compress rows parallel","rock","music","mario");
```

- To compress only row data and then repack and shrink the data, specify:

```
EXECUTE FUNCTION task("table compress repack shrink rows parallel",
"rock","music","mario");
```

- To compress only simple large objects in the dbspace, specify:

```
EXECUTE FUNCTION task("table compress blobs parallel","rock","music","mario");
```

After the existing rows and simple large objects are compressed, consolidates the free space that is left at the end of the table, and then removes the free space from the table, returning that space to the dbspace.

If the simple large objects or rows are not smaller when compressed, the database server does not compress them.

4. Now suppose that you want to uncompress the data. You run the following command:

```
EXECUTE FUNCTION task("table uncompress parallel", "rock", "music", "mario");
```

5. You want to remove the compression dictionary.
   a. Verify that Enterprise Replication does not require the dictionary.
      If you do require the dictionaries for Enterprise Replication, do not remove compression dictionaries for uncompressed or dropped tables and fragments.
   b. Archive the dbspace that contains the table or fragment with a compression dictionary.
   c. Run this command:

```
EXECUTE FUNCTION task("table purge_dictionary", "rock", "music", "mario");
```

To run compression and other storage optimization commands on table fragments, include the **fragment** argument instead of the **table** argument and the fragment partition number instead of the table name.

```
EXECUTE FUNCTION task("fragment command_arguments", "partnum_list");
```

Related concepts:
Compression
Related information:
table or fragment arguments: Compress data and optimize storage (SQL administration API)

Partition defragmentation

You can improve performance by defragmenting partitions to merge non-contiguous extents.

A frequently updated table can become fragmented over time, which degrades performance every time the table is accessed by the server. Defragmenting a table brings data rows closer together and avoids partition header page overflow problems. Defragmenting an index brings the entries closer together, which improves the speed at which the table information is accessed.

Before you defragment a table, index, or partition, be sure that none of the following conflicting operations are in progress:

- An existing defragment operation on the table, index, or dbspace.
- DDL statements, such as DROP TABLE or ALTER FRAGMENT, are being run on the table or partition.
- The table is being truncated.
- The table is being compressed or uncompressed.
- An online index build is running.

You cannot defragment the following objects:

- Pseudo tables, such as virtual-table interface (VTI) tables
- Tables with virtual-index interface (VII) indexes
- Tables with functional indexes
- Temporary tables
- Sort files
- A table that has exclusive access set

To determine how many extents a table, index, or partition has, you can run the **oncheck -pt** command.

To defragment a table, index, or partition, run the SQL administration API **task()** or **admin()** function with the defragment argument or the defragment partnum argument and specify the table name, index, or partition number that you want to defragment.

You cannot stop a defragment request after you run the command.

If there are problems in completing a defragment request, error messages are sent to the online log file.

Related tasks:
Scheduling data optimization
Related information:
oncheck -pt and -pT: Display tblspaces for a Table or Fragment
defragment argument: Dynamically defragment partition extents (SQL administration API)

Compression

You can compress and uncompress row data in tables and fragments and simple large objects in dbspaces. You can compress B-tree indexes. You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Before you compress data, you can estimate the amount of disk space that you can save.

Compressing data, simple large objects, or indexes, consolidating data, and returning free space have the following benefits:

- Significant savings in disk storage space
- Reduced disk usage for compressed fragments
- Significant saving of logical log usage, which saves more space and can prevent bottlenecks for high-throughput OLTP after the compression operation is completed.
- Fewer page reads because more rows can fit on a page
- Smaller buffer pools because more data fits in the same size pool
- Reduced I/O activity:
  - More compressed rows than uncompressed rows fit on a page
  - Log records for insert, update, and delete operations of compressed rows are smaller
- Ability to compress older fragments of time-fragmented data that are not often accessed, while leaving more recent data that is frequently accessed in uncompressed form
- Ability to free space no longer required for a table
- Faster backup and restore

If your applications run with high buffer cache hit ratios and high performance is more important than space usage, you might not want to compress your data, because compression might slightly decrease performance.

You can compress data and indexes in parallel.

Queries can access data in a compressed table.

Because compressed data covers fewer pages and has more rows per page than uncompressed data, the query optimizer might choose different plans after compression.

If you use Enterprise Replication, compressing data on one replication server does not affect the data on any other replication server.

If you use high-availability clusters, data that is compressed in the source table is compressed in the target table. You cannot perform compression operations on secondary servers, because secondary servers must have the same data and physical layout as the primary server.

The main alternative to compression is to buy more physical storage. The main alternative for reducing bottlenecks in IO-bound workloads is to buy more physical memory to enable the expansion of the buffer pools.

- Data that you can compress
  You can compress data in rows and simple large objects in dbspaces. However, you might not want to compress all the types of data that you can compress.
- Data that you cannot compress
  You cannot compress data in rows in some types of tables and fragments.
- B-tree index compression
  You can compress detached B-tree indexes. You can also consolidate free space in the index and you can return free space at the end of the index to the dbspace. Before you compress an index, you can estimate the amount of disk space that you can save.
- Compression ratio estimates
  The compression ratio depends on the data that is being compressed. Before you compress a table or table fragment, you can estimate the amount of space you can save if data is compressed. Compression estimates are based on samples of row data. The actual ratio of saved space might vary.
- Compression dictionaries
  A compression dictionary is a library of frequently occurring patterns in data or index keys and the symbol numbers that replace the patterns.
- Tools for moving compressed data
  You can use the High-Performance Loader (HPL) and other IBM® Informix® data migration utilities to move compressed data between databases.
- BLOBspace Blob Compression
  A *partition BLOB* (binary large object) column is a TEXT or BYTE column that is stored in a DBspace, in the same RSAM partition as the home row. A *BLOBspace BLOB* column is the same data type (TEXT or BYTE), but the BLOB data is stored in a BLOBspace.
- Methods for viewing compression information
  You can display compression statistics, information about compression dictionaries, and the compression dictionary.

Related concepts:
Storage optimization methods
Related tasks:
Example: Optimizing data storage on demand

Data that you can compress

You can compress data in rows and simple large objects in dbspaces. However, you might not want to compress all the types of data that you can compress.

You can compress the following types of data:

- The contents of data rows, including any remainder pieces for rows that span pages, and the images of those rows that are contained in logical log records.
- Simple large objects (TEXT or BYTE data types) that are stored in dbspaces.

Table or table-fragment data with frequently repeating long patterns is very compressible. Certain types of data, such as text, might be more compressible than other types of data, such as numeric data, because data types like text might contain longer and more frequently repeating patterns.

I/O-bound tables, for example, tables that have bad cache hit ratios, are good candidates for compression. In OLTP environments, compressing I/O-bound tables can improve performance.

IBM® Informix® can compress any combination of data types, because it treats all data to be compressed as unstructured sequences of bytes. Thus, the server can compress patterns that span columns, for example, in city, state, and zip code combinations. (The server uncompresses a sequence of bytes in the same sequence that existed before the data was compressed.)

Related concepts:
Storage optimization methods

Data that you cannot compress

You cannot compress data in rows in some types of tables and fragments.

You cannot compress data in rows in the following database objects:

- Tables or fragments that are in the **sysmaster**, **sysutils**, **sysuser**, **syscdr**, and **syscdcv1** databases
- Catalogs
- Temporary tables
- Virtual-table interface tables
- The tblspace **tblspace**
- Internal partition tables
- Dictionary tables (these tables, one per dbspace, hold compression dictionaries for the fragments or tables that are compressed in that dbspace and metadata about the dictionaries.)

You cannot compress a table while an online index build is occurring on the table.

You cannot compress simple large objects in blobspaces.

Encrypted data, data that is already compressed by another algorithm, and data without long repeating patterns compresses poorly or does not compress. Try to avoid placing columns with data that compresses poorly between columns that have frequent patterns to prevent the potential disruption of column-spanning patterns.

IBM® Informix® compresses images of the rows only if the images of the compressed rows are smaller than the uncompressed images. Even if compressed rows are only slightly smaller than their uncompressed images, a small saving of space can enable the server to put more rows onto pages.

Very small tables are not good candidates for compression, because you might not be able to gain back enough space from compressing the rows to offset the storage cost of the compression dictionary.

cannot compress an individual row to be smaller than four bytes long. The server must leave room in case the row image later grows beyond what the page can hold. Therefore, you must not try to compress fragments or non-fragmented tables with rows that contain four bytes or are shorter than four bytes.

B-tree index compression

You can compress detached B-tree indexes. You can also consolidate free space in the index and you can return free space at the end of the index to the dbspace. Before you compress an index, you can estimate the amount of disk space that you can save.

You can compress a detached B-tree index that is on a fragmented or non-fragmented table.

An index must have at least 2000 keys to be compressed.

You cannot compress the following types of indexes:

- An index that is not a B-tree index
- An attached B-tree index
- Virtual B-tree indexes
- An index that does not have at least 2000 keys

The compression operation compresses only the leaves (bottom level) of the index.

You cannot uncompress a compressed index. If you no longer need the compressed index, you can drop the index and then re-create it as an uncompressed index.

You can compress a new index when you create it by including the COMPRESSED option in the CREATE INDEX statement. You compress an existing index with an SQL administration API command.

Related information:
index compress repack shrink arguments: Optimize the storage of B-tree indexes (SQL administration API)
index estimate_compression argument: Estimate index compression (SQL administration API)
CREATE INDEX statement
Creation of Root and Leaf Nodes

Compression ratio estimates

The compression ratio depends on the data that is being compressed. Before you compress a table or table fragment, you can estimate the amount of space you can save if data is compressed. Compression estimates are based on samples of row data. The actual ratio of saved space might vary.

The compression algorithm that IBM® Informix® uses is a dictionary-based algorithm that performs operations on the patterns of the data that were found to be the most frequent, weighted by length, in the data that was sampled at the time the dictionary was built.

If the typical data distribution skews away from the data that was sampled when the dictionary was created, compression ratios can decrease.

The maximum compression ratio is 90 percent. The maximum compression of any sequence of bytes occurs by replacing each group of 15 bytes with a single 12-bit symbol number, yielding a compressed image that is ten percent of the size of the original image. However, the 90 percent ratio is never achieved because adds a single byte of metadata to each compressed image.

IBM Informix estimates the compression ratios by random sampling of row data and then summing up the sizes of the following items:

- Uncompressed row images
- Compressed row images, based on a new compression dictionary that is temporarily created by the estimate compression command
- Compressed row images, based on the existing dictionary, if there is one. If there is no existing dictionary, this value is the same as the sum of the sizes of the uncompressed row images.

The actual space saving ratios that are achieved might vary from the compression estimates due to a sampling error, the type of data, how data fits in data pages, or whether other storage optimization operations are also run.

Some types of data compress more than other types of data:

- Text in different languages or character sets might have different compression ratios, even though the text is stored in CHAR or VARCHAR columns.
- Numeric data that consists mostly of zeros might compress well, while more variable numeric data might not compress well.
- Data with long runs of blank spaces compresses well.
- Data that is already compressed by another algorithm and data that is encrypted might not compress well. For example, images and sound samples in rows might already be compressed, so compressing the data again does not save more space.

Compression estimates are based on raw compressibility of the rows. The server generally puts a row onto a single data page. How the rows fit on data pages can affect how much the actual compression ratio varies from the estimated compression ratio:

- When each uncompressed row nearly fills a page and the compression ratio is less than 50 percent, each compressed row fills more than half a page. The server puts each compressed row on a separate page. In this case, although the estimated compression ratio might be 45 percent, the actual space savings is nothing.
- When each uncompressed row fills slightly more than half a page and the compression ratio is low, each compressed row might be small enough to fit in half a page. The server puts two compressed rows on a page. In this case, even though the estimated compression ratio might be as low as 5 percent, the actual space savings is 50 percent.

does not store more than 255 rows on a single page. Thus, small rows or large pages can reduce the total savings that compression can achieve. For example, if 200 rows fit onto a page before compression, no matter how small the rows are when compressed, the maximum effective compression ratio is approximately 20 percent, because only 255 rows can fit on a page after compression.

If you are using a page size that is larger than the minimum page size, one way to increase the realized compression space savings is to switch to smaller pages, so that:

- The 255 row limit can no longer be reached.
- If this limit is still reached, there is less unused space on the pages.

More (or less) space can be saved, compared to the estimate, if the compress operation is combined with a repack operation, shrink operation, or repack and shrink operation. The repack operation can save extra space only if more compressed rows fit on a page than uncompressed rows. The shrink operation can save space at the dbspace level if the repack operation frees space.

Related information:
Output of the estimate compression operation (SQL administration API)

Compression dictionaries

A compression dictionary is a library of frequently occurring patterns in data or index keys and the symbol numbers that replace the patterns.

One compression dictionary exists for each compressed fragment, each compressed non-fragmented table, each compressed simple large object in a dbspace, and each compressed index partition.

A compression dictionary is built using data that is sampled randomly from a fragment or non-fragmented table that contains at least 2,000 rows, or an index that has at least 2,000 keys. Typically, approximately 100 KB of space is required for storing the compression dictionary.

The compression dictionary can store a maximum of 3,840 patterns, each of which can be from two to 15 bytes in length. (Patterns that are longer than seven bytes reduce the total number of patterns that the dictionary can hold.) Each of these patterns is represented by a 12-bit symbol number in a compressed row. To be compressed, a sequence of bytes in the input row image must exactly match a complete pattern in the dictionary. A row that does not have enough pattern matches against the dictionary might not be compressible because each byte of an input row that did not completely match is replaced in the compressed image by 12 bits (1.5 bytes).

attempts to capture the best compressible patterns (the frequency of the pattern that is multiplied by the length). Data is compressed by replacing occurrences of the patterns with the corresponding symbol numbers from the dictionary, and replacing occurrences of bytes that do not match any pattern with special reserved symbol numbers.

All dictionaries for the tables or fragments in a dbspace are stored in a hidden dictionary table in that dbspace. The **syscompdicts_full** table and the **syscompdicts** view in the **sysmaster** database provide information about the compression dictionaries.

Related information:
syscompdicts_full

Tools for moving compressed data

You can use the High-Performance Loader (HPL) and other IBM® Informix® data migration utilities to move compressed data between databases.

You cannot use the **onunload** and **onload** utilities to move compressed data from one database to another. You must uncompress data in compressed tables and fragments before you use the **onunload** and **onload** utilities.

The **dbexport** utility uncompresses compressed data. Therefore, you must recompress the data after you use the **dbimport** utility to import the data.

Related information:
Data migration utilities
High-Performance Loader User's Guide

BLOBspace Blob Compression

A *partition BLOB* (binary large object) column is a TEXT or BYTE column that is stored in a DBspace, in the same RSAM partition as the home row. A *BLOBspace BLOB* column is the same data type (TEXT or BYTE), but the BLOB data is stored in a BLOBspace.

All TEXT and BYTE BLOBs may be compressed, whether they are stored in the partition or a BLOBspace. If a table containing TEXT or BYTES columns is created with the COMPRESSED keyword, the BLOB data will be automatically compressed along with the home row data once the number of rows has reached a certain threshold (2000 by default). Separate compression dictionaries are built for the home rows and each BLOB column.

- By using the "compress" sysadmin task() command rather than the auto compress feature, it is possible to compress only the BLOB data in a table, or to compress only the home row data without compressing BLOBs.
- ER is able to replicate BLOB data whether it is compressed or uncompressed.

Methods for viewing compression information

You can display compression statistics, information about compression dictionaries, and the compression dictionary.

The following table describes the different methods that you can use to view compression information.

Table 1. Methods to view compression information

Method	Description
oncheck -pT or **oncheck -pt** command	Displays statistics on any compressed items in the "Compressed Data Summary" section of the output. If no items are compressed, the "Compressed Data Summary" section does not appear in the output. For example, for row data, **oncheck -pT** displays the number of any compressed rows in a table or table fragment and the percentage of table or table-fragment rows that are compressed.
onlog -c option	Uses the compression dictionary to expand compressed data and display the uncompressed contents of compressed log records.
onstat –g dsk option	Displays information about the progress of currently running compression operations.
onstat -g ppd option	Displays information about the active compression dictionaries that exist for currently open compressed fragments (also referred to as partitions). This option shows the same information as the **syscompdicts** view in the **sysmaster** database.
syscompdicts_full table in the **sysmaster** database	Displays metadata about the compression dictionary and the compression dictionary binary object. Only user **informix** can access this table.
syscompdicts view in the **sysmaster** database	Displays the same information as the **syscompdicts_full** table, except that for security reasons, it excludes the **dict_dictionary** column, which contains the compression dictionary binary object.
`UNLOAD TO 'compression_dictionary_file'` `SELECT * FROM sysmaster:syscompdicts_full;` SQL statement	View the compression dictionary in a file.

Related concepts:
Storage optimization methods
Related information:
onstat -g dsk command: Print the progress of the currently running compression operation
onstat -g ppd command: Print partition compression dictionary information
oncheck -pt and -pT: Display tblspaces for a Table or Fragment
The onlog utility

syscompdicts_full

Load data into a table

You can load data into an existing table in the following ways.

Method to load data	TEXT or BYTE data	CLOB or BLOB data	Reference
DB-Access LOAD statement	Yes	Yes	LOAD statement in the *IBM® Informix Guide to SQL: Syntax*
dbload utility	Yes	Yes	*IBM Informix Migration Guide*
dbimport utility	Yes	Yes	*IBM Informix Migration Guide*
Informix® ESQL/C programs	Yes	Yes	*IBM Informix ESQL/C Programmer's Manual*
Insert MERGE, using an EXTERNAL source table	Yes	Yes	*IBM Informix Guide to SQL: Syntax*
onload utility	No	No	*IBM Informix Migration Guide*
onpladm utility	Yes, deluxe mode	Yes, deluxe mode	*IBM Informix High-Performance Loader User's Guide*
High-Performance Loader (HPL)	Yes, deluxe mode	Yes, deluxe mode	*IBM Informix High-Performance Loader User's Guide*

Important: The database server does not contain any mechanisms for compressing TEXT and BYTE data after the data has been loaded into a database.

Moving data with external tables

You can use external tables to load and unload database data.

You issue a series of SQL statements that perform the following functions:

- Transfer operational data efficiently to or from other systems
- Transfer data files across platforms in IBM® Informix® internal data format
- Use the database server to convert data between delimited ASCII, fixed-ASCII, and IBM Informix internal (raw) representation
- Use SQL INSERT and SELECT statements to specify the mapping of data to new columns in a database table
- Provide parallel standard INSERT operations so that data can be loaded without dropping indexes
- Use named pipes to support loading data to and unloading data from storage devices, including tape drives and direct network connections
- Maintain a record of load and unload statistics during the run
- Perform express (high-speed) and deluxe (data-checking) transfers

You can issue the SQL statements with DB-Access or embed them in an ESQL/C program.

- External tables
  An *external table* is a data file that is not managed by the IBM Informix database server. The definition of the external table includes data-formatting type, external data description fields, and global parameters.
- Defining external tables
  To define an external table, you use SQL statements to describe the data file, define the table, and then specify the data to load or unload.
- Map columns to other columns
- Load data from and unload to a named pipe
  You can use a named pipe, also called a first-in-first-out (FIFO) data file, to load from and unload to a nonstandard device, such as a tape drive.
- Monitor the load or unload operations
  You can monitor the status of an external table load or unload operation.
- External tables in high-availability cluster environments
  You use external tables on secondary servers in much the same way they are used on the primary server.
- System catalog entries for external tables
  You can query system catalog tables to determine the status of external tables.
- Performance considerations when using external tables
  Use external tables when you want to manipulate data in an ASCII file using SQL commands, or when loading data from an external data file to a RAW database table.
- Manage errors from external table load and unload operations
  You can manage errors that occur during external table load and unload operations.

External tables

An *external table* is a data file that is not managed by the IBM® Informix® database server. The definition of the external table includes data-formatting type, external data description fields, and global parameters.

To map external data to internal data, the database server views the external data as an external table. Treating the external data as a table provides a powerful method for moving data into or out of the database and for specifying transformations of the data.

When the database server runs a load task, it reads data from the external source and performs the conversion required to create the row and then inserts the row into the table. The database server writes errors to a reject file.

If the data in the external table cannot be converted, you can specify that the database server write the record to a *reject file*, along with the reason for the failure. To do this, you specify the REJECTFILE keyword in the CREATE EXTERNAL TABLE statement.

The database server provides a number of different conversion mechanisms, which are performed within the database server and therefore provide maximum performance during the conversion task. The database server optimizes data conversion between ASCII and IBM Informix data representations, in both fixed and delimited formats.

To perform customized conversions, you can create a filter program that writes converted data to a named pipe. The database server then reads its input from the named pipe in one of the common formats.

Defining external tables

To define an external table, you use SQL statements to describe the data file, define the table, and then specify the data to load or unload.

To set up loading and unloading tasks, you issue a series of SQL statements:

- CREATE EXTERNAL TABLE to describe the data file to load or unload
- CREATE TABLE to define the table to load
- INSERT...SELECT to load and unload

The following steps outline the load process:

1. The CREATE EXTERNAL TABLE statement describes the location of the various external files, which can be on disk or come from a pipe (tape drive or direct network connection), and the format of the external data. The following example is a CREATE EXTERNAL TABLE statement:

```
CREATE EXTERNAL TABLE emp_ext
   ( name CHAR(18) EXTERNAL CHAR(18),
     hiredate DATE EXTERNAL CHAR(10),
     address VARCHAR(40) EXTERNAL CHAR(40),
     empno INTEGER EXTERNAL CHAR(6) )
USING (
   FORMAT 'FIXED',
    DATAFILES
      ("DISK:/work2/mydir/emp.fix")
    );
```

2. The CREATE TABLE statement defines the table to load. The following sample CREATE TABLE statement defines the **employee** table:

```
CREATE TABLE employee
   FRAGMENT BY ROUND ROBIN IN dbspaces;
```

3. The INSERT...SELECT statement maps the movement of the external data from or to the database table. The following sample INSERT statement loads the **employee** table from the external table:

```
INSERT INTO employee SELECT * FROM emp_ext
```

Important: If you specify more than one *INSERT...SELECT statement* to unload data, each subsequent INSERT statement overwrites the data file. Use absolute paths for data files.
When you load data into the database, the FROM *table* portion of the SELECT clause is the external table that the CREATE EXTERNAL statement defined. When you unload data to an external file, the SELECT clause controls the retrieval of the data from the database.

Unlike a TEMP table, the external table has a definition that remains in the catalog until it is dropped. When you create an external table you can save the external description of the data for reuse. This action is particularly helpful when you unload a table into the IBM® Informix® internal data representation because you can later use the same external table description to reload that data.

On Windows systems, if you use the **DB-Access** utility or the **dbexport** utility to unload a database table into a file and then plan to use the file as an external table datafile, you must define RECORDEND as '\012' in the CREATE EXTERNAL TABLE statement.

The external table definition contains all the information required to define the data in the external data file as follows:

- The description of the fields in the external data.
- The DATAFILES clause.
  This clause specifies:
  - Whether the data file is located on disk or a named pipe.
  - The path name of the file.
- The FORMAT clause.
  This clause specifies the type of data formatting in the external data file. The database server converts external data from several data formats, including delimited and fixed ASCII, and IBM Informix internal.
- Any global parameters that affect the format of the data.

If you map the external table directly into the internal database table in delimited format, you can use the CREATE EXTERNAL TABLE statement to define the columns and add the clause SAMEAS *internal-table* instead of enumerating the columns explicitly.

Map columns to other columns

If the data file is to have fields in a different order (for example, **empno**, **name**, **address**, **hiredate**), you can use the INSERT statement to map the columns. First, create the table with the columns in the order in which they are found in the external file.

```
CREATE EXTERNAL TABLE emp_ext
   (
   f01 INTEGER,
   f02 CHAR(18),
   f03 VARCHAR(40),
   f04 DATE
   )
USING (
   DATAFILES ("DISK:/work2/mydir/emp.dat"),
   REJECTFILE "/work2/mydir/emp.rej"
   );
INSERT INTO employee (empno, name, address, hiredate)
   SELECT * FROM emp_ext;
```

With this method, the insert columns are mapped to match the field order of the external table.

Another way to reorder columns is to use the SELECT clause to match the order of the database table.

```
INSERT INTO employee
   SELECT f02, f04, f03, f01 FROM emp_ext;
```


Load data from and unload to a named pipe

You can use a named pipe, also called a first-in-first-out (FIFO) data file, to load from and unload to a nonstandard device, such as a tape drive.

Unlike ordinary operating-system files, named pipes do not have a 2-gigabyte size limitation. The operating system opens and checks for the end of file differently for named pipes than for ordinary files.

- Loading data with named pipes
  You can use a named pipe to load data from external tables.
- FIFO virtual processors
  The database server uses FIFO virtual processors (VPs) to read and write to external tables on named pipes.
- Unloading data with named pipes
  You can use a named pipe to unload data from the database to external tables.
- Copying data from one instance to another using the PIPE option
  You can use a named pipe to copy data from one Informix® instance to another without writing the data to an intermediate file.

Loading data with named pipes

You can use a named pipe to load data from external tables.

To use a named pipe to load data from an external table, follow these steps:

1. Specify the named pipes in the DATAFILES clause of the CREATE EXTERNAL TABLE statement in SQL.
2. Create the named pipes that you specified in the DATAFILES clause. Use operating-system commands to create the named pipes.
   Use the **mknod** UNIX command with the **-p** option to create a named pipe. To avoid blocking open problems for pipes on UNIX, start separate UNIX processes for pipe-readers and pipe-writers or open the pipes with the O_NDELAY flag set.
3. Open the named pipes with a program that reads the named pipe.
4. Execute the INSERT statement in SQL.

   ```
   INSERT INTO employee SELECT * FROM emp_ext;
   ```

Important: If you do not create and open the named pipes before you execute the INSERT statement, the INSERT succeeds, but no rows are loaded.

FIFO virtual processors

The database server uses FIFO virtual processors (VPs) to read and write to external tables on named pipes.

The default number of FIFO virtual processors is `1`.

The database server uses one FIFO VP for each named pipe that you specify in the DATAFILES clause of the CREATE EXTERNAL TABLE statement. For example, suppose you define an external table with the following SQL statement:

```
CREATE EXTERNAL TABLE ext_items
   SAMEAS items
   USING (
      DATAFILES("PIPE:/tmp/pipe1",
                "PIPE:/tmp/pipe2",
                "PIPE:/tmp/pipe3"
     ));
```

If you use the default value of `1` for FIFO VPs, the database server does not read from **pipe2** until it finishes reading all the data from **pipe1**, and does not read from **pipe3** until it finishes reading all the data from **pipe2**.

Unloading data with named pipes

You can use a named pipe to unload data from the database to external tables.

To use named pipes to unload data to external tables, follow these steps:

1. Specify the named pipe in the DATAFILES clause of either the CREATE EXTERNAL TABLE statement or the SELECT INTO EXTERNAL statement of SQL.

   ```
   DATAFILES ("PIPE:/usr/local/TAPE")
   ```

2. Create the named pipes that you specified in the DATAFILES clause. Use operating-system commands to create the named pipes.
3. Open the named pipes with a program that writes to the named pipe.
4. Unload data to the named pipe.

   ```
   CREATE EXTERNAL TABLE emp_ext
      ( name CHAR(18) EXTERNAL CHAR(20),
        hiredate DATE EXTERNAL CHAR(10),
        address VARCHAR(40) EXTERNAL CHAR(40),
        empno INTEGER EXTERNAL CHAR(6) )
      USING (
         FORMAT 'FIXED',
         DATAFILES
         ("PIPE:/usr/local/TAPE")
          );

   INSERT INTO emp_ext SELECT * FROM employee;
   ```

Important: If you do not create and open the named pipes before you execute the SELECT or INSERT statement, the unload fails with the *ENXIO* error message (`no such device or address`).

Copying data from one instance to another using the PIPE option

You can use a named pipe to copy data from one Informix® instance to another without writing the data to an intermediate file.

You can use a named pipe to unload data from one Informix instance and load it into another instance without writing data to an intermediate file. You can also use a named pipe to copy data from one table to another on the same Informix instance. In the following example, data is copied from a source table on one instance to a destination table on a second instance.

Depending on the hardware platform you are using, you must first create a named pipe using one of the following commands. For this example, the named pipe is called pipe1.

```
% mkfifo /work/pipe1
% mknod /work/pipe1
```

Follow these steps to copy data from a table on a source instance to a table on a destination instance on the same computer.

1. Create the source table on the source instance. In this example, the source table is called source_data_table:

   ```
   CREATE TABLE source_data_table
   (
     empid    CHAR(5),
     empname  VARCHAR(40),
     empaddr  VARCHAR(100)
   );
   ```

2. Create the external table on the source instance. In this example, the external table is named ext_table:

   ```
   CREATE EXTERNAL TABLE ext_table
   (
     empid    CHAR(5),
     empname  VARCHAR(40),
     empaddr  VARCHAR(100)
   )
   USING
   (DATAFILES
     (
       'PIPE:/work/pipe1'
     )
   );
   ```

3. Create the destination table on the destination instance. In this example, the destination table is called destin_data_table:

```
CREATE TABLE destin_data_table
(
  empid    CHAR(5),
  empname  VARCHAR(40),
  empaddr  VARCHAR(100)
);
```

4. Create the external table on the destination instance. In this example, the external table is named ext_table:

```
CREATE EXTERNAL TABLE ext_table
(
  empid    CHAR(5),
  empname  VARCHAR(40),
  empaddr  VARCHAR(100)
)
USING
(DATAFILES
  (
    'PIPE:/work/pipe1_1'
  )
);
```

5. Run the following command from a UNIX shell. The command redirects data from /work/pipe1 to /work/pipe1_1

```
cat /work/pipe1 > /work/pipe1_1
```

6. Run the following command on the destination instance to direct data from the named pipe to the destination table:

```
INSERT INTO destin_data_table SELECT * FROM ext_table;
```

7. Run the following command on the source instance to spool data to the named pipe:

```
INSERT INTO ext_table SELECT * FROM source_data_table;
```

You can use more than one pipe by inserting multiple PIPE statements in the DATAFILES clause and creating a named pipe for each.

Monitor the load or unload operations

You can monitor the status of an external table load or unload operation.

You might want to monitor the load or unload operations for the following situations:

- If you expect to load and unload the same table often to build a data mart or data warehouse, monitor the progress of the job to estimate the time of similar jobs for future use.
- If you load or unload from named pipes, monitor the I/O queues to determine if you have a sufficient number of FIFO virtual processors.

- Monitor frequent load and unload operations
  Use the **onstat -g iof** command to find the global file descriptor (gfd) in the file that you want to examine. Then the **onstat -g sql** command to monitor load and unload operations.
- Monitor FIFO virtual processors

Monitor frequent load and unload operations

Use the **onstat -g iof** command to find the global file descriptor (gfd) in the file that you want to examine. Then the **onstat -g sql** command to monitor load and unload operations.

The following example shows sample **onstat -g iof** command output.

```
AIO global files:
gfd path name      bytes read   page reads   bytes write   page writes  io/s
3   rootdbs        1918976      937          145061888     70831        36.5

      op type     count          avg. time
      seeks       0              N/A
      reads       937            0.0010
      writes      4088           0.0335
      kaio_reads  0              N/A
      kaio_writes 0              N/A
```

To determine if a load or unload operation can use parallel execution, execute the SET EXPLAIN ON statement before the INSERT statement. The SET EXPLAIN output shows the following counts:

- Number of parallel SQL operators that the optimizer chooses for the INSERT statement
- Number of rows to be processed by each SQL operator

To monitor a load operation, run **onstat -g sql** to obtain the session ID.

Monitor FIFO virtual processors

You can monitor the effective usage of FIFO VPs with **onstat** commands.

Use the **onstat -g ioq** option to display the length of each FIFO queue that is waiting to perform I/O requests. The following example shows sample output.

```
AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
 fifo   0      0      0        0        0        0        0
  adt   0      0      0        0        0        0        0
  msc   0      0      1      153        0        0        0
  aio   0      0      9     3499     1013       77        0
  pio   0      0      2        3        0        2        0
  lio   0      0      2     2159        0     2158        0
  gfd   3      0     16    39860       38    39822        0
  gfd   4      0     16    39854       32    39822        0
  gfd   5      0      1        2        2        0        0
  gfd   6      0      1        2        2        0        0
...
  gfd  19      0      1        2        2        0        0
```

The **q name** field in the sample output in the previous example shows the type of the queue, such as **fifo** for a FIFO VP or **aio** for an AIO VP. If the **q name** field shows **gfd** or **gfdwq**, it is a queue for a file whose global file descriptor matches the **id** field of the output. Disk files have both read and write requests in one queue. One line per disk file displays in the **onstat -g ioq** output. Pipes have separate read and write queues. Two lines per pipe display in the output: **gfd** for read requests and **gfdwq** for write requests.

The **len** or **maxlen** field has a value of up to `4` for a load or `4 * number_of_writer_threads` for an unload. The **xuwrite** operator controls the number of writer threads.

Use the values in the **totalops** field rather than the **len** or **maxlen** field to monitor the number of read or write requests done on the file or pipe. The **totalops** field represents 34 KB of data read from or written to the file. If **totalops** is not increasing, it means the read or write operation on a file or pipe is stalled (because the FIFO VPs are busy).

To improve performance, use the **onmode -p** command to add more FIFO VPs. The default number of FIFO VPs is 1. In this sample output, the FIFO queue does not contain any data. For example, if you usually define more than two pipes to load or unload, increase the number of FIFO VPs with the following sample **onmode** command:

```
onmode -p +2 FIFO
```

For more information, see *IBM® Informix® Administrator's Reference*.

External tables in high-availability cluster environments

You use external tables on secondary servers in much the same way they are used on the primary server.

You can perform the following operations on the primary and on secondary servers:

- Unload data from a database table to an external table:

  ```
  INSERT INTO external_table SELECT * FROM base_table WHERE ...
  ```

- Load data from an external table into a database table:

  ```
  INSERT INTO base_table SELECT * FROM external_table WHERE ...
  ```

Loading data on SDS, RSS, or HDR secondary servers is slower than loading data on the primary server.

The CREATE EXTERNAL TABLE statement and the SELECT ... INTO EXTERNAL ... statement are not supported on secondary servers.

When unloading data from a database table to an external table, data files are created on the secondary server but not on the primary server. External table data files created on secondary servers are not automatically transferred to the primary server, nor are external table data files that are created on the primary server automatically transferred to secondary servers.

When creating an external table on a primary server, only the schema of the external table is replicated to the secondary servers, not the data file.

To synchronize external tables between the primary server and a secondary server, you can either copy the external table file from the primary server to the secondary servers, or use the following steps:

1. On the primary server:
   a. Create a temporary table with the same schema as the external table.
   b. Populate the temporary table:

      ```
      INSERT INTO dummy_table SELECT * FROM external_table
      ```

2. On the secondary server:
   Use the following command to populate the external table:

   ```
   INSERT INTO external_table SELECT * FROM dummy_table
   ```

System catalog entries for external tables

You can query system catalog tables to determine the status of external tables.

IBM® Informix® updates the **sysexternal** and **sysextdfiles** system catalog tables each time an external table is created. The **sysextcols** system catalog table is updated when the external format type (**fmttype**) FIXED is specified.

Table 1. System catalog tables that describe external table files

Table name	Description
sysexternal	Stores name and attributes of each external table file
sysextdfiles	Stores file-path and directory of each external table file
sysextcols	Stores attributes of each column in FIXED-type external tables

See the *IBM Informix Guide to SQL: Reference* for more information.

A row is inserted into the **systables** system catalog when an external table is created; however, the **nrows** (number of rows) and the **npused** (number of data pages used) columns might not accurately reflect the number of rows and the number of data pages used by the external table unless the NUMROWS clause was specified when the external table was created.

When an external table is created without specifying a value for the NUMROWS clause, is unable to determine the number of rows in the external table because the data exists outside the database in data files. updates the **nrows** column in the **systables** system catalog by inserting a large value (MAXINT – 1), and computes the number of data pages used based on the **nrows** value. The values stored in **npused** and **nrows** are later used by the optimizer to determine the most efficient execution plan. While the NUMROWS clause is not required to be specified precisely, the more accurately it is specified, the more accurate the values for **nrows** and **npused** are.

Performance considerations when using external tables

Use external tables when you want to manipulate data in an ASCII file using SQL commands, or when loading data from an external data file to a RAW database table.

There are several ways to load information into a database, including:

- LOAD FROM ... INSERT INTO... DB-Access command
- **dbimport** utility
- High-Performance Loader utility
- External tables

The High Performance Loader utility provides best performance for loading external data into a database table with indexes.

External tables provide the best performance for loading data into a RAW table with no indexes.

Note: Locking an external table prior to loading data increases the load performance

Manage errors from external table load and unload operations

You can manage errors that occur during external table load and unload operations.

These topics describe how to use the reject file and error messages to manage errors, and how to recover data loaded into the database.

- Reject files
  Rows that have conversion errors during a load are written to a reject file on the server that performs the conversion.
- External table error messages
  Most of the error messages related to external tables are in the -26151 to -26199 range.
- Recoverability of table types for external tables
  The database server checks the recoverability level of the table when loading of data.

Reject files

Rows that have conversion errors during a load are written to a reject file on the server that performs the conversion.

The REJECTFILE keyword in the CREATE EXTERNAL TABLE statement determines the name given to the reject file.

Instead of using a reject file, you can use the MAXERRORS keyword in the CREATE EXTERNAL TABLE statement to specify the number of errors that are allowed before the database server stops loading data. (If you do not set the MAXERRORS keyword, the database server processes all data regardless of the number of errors.)

The database server removes the reject files, if any, at the beginning of a load. The reject files are recreated and written only if errors occur during the load.

Reject file entries are single lines with the following comma-separated fields:

```
file name, record, reason-code, field-name: bad-line
```

file name
Name of the input file
record
Record number in the input file where the error was detected
reason-code
Description of the error
field-name
The external field name where the first error in the line occurred or <none> if the rejection is not specific to a particular column
bad-line
For delimited or fixed-ASCII files only, the bad line itself

The load operation writes *file name*, *record*, *field-name*, and *reason-code* in ASCII.

The *bad-line* information varies with the type of input file:

- For delimited files or fixed text files, the entire bad line is copied directly into the reject file. However, if the delimited format table has TEXT or BYTE columns, the reject file does not include any bad data. The load operation generates only a header for each rejected row.
- For IBM® Informix® internal data files, the bad line is not placed in the reject file because you cannot edit the binary representation in a file. However, the *file name*, *record*, *reason-code*, and *field-name* are still reported in the reject file so that you can isolate the problem.

The following types of errors can cause a row to be rejected.

CONSTRAINT *constraint name*
This constraint was violated.
CONVERT_ERR
Any field encounters a conversion error.
MISSING_DELIMITER
No delimiter was found.
MISSING_RECORDEND
No record end was found.
NOT NULL
A null was found in *field-name*.
ROW_TOO_LONG
The input record is longer than 2 GB.

External table error messages

Most of the error messages related to external tables are in the -26151 to -26199 range.

Additional messages are -615, -999, -23852, and -23855. In the messages, *n macro* and *r macro* refer to the values generated from the substitution character %r (*first..last*) . For a list of error messages, see *IBM® Informix® Error Messages* or use the **finderr** utility. For information about the violations table error messages, see your *IBM Informix Administrator's Reference*.

Recoverability of table types for external tables

The database server checks the recoverability level of the table when loading of data.

- If the logging type of the table is RAW, the database server can use light append (or EXPRESS) mode to load data and to process check constraints. However, if the database server crashes while inserting data rows into a RAW table in EXPRESS mode, this unlogged light append operation is not rolled back, and the table might be left in an unknown state.
- Only DELUXE mode supports data recoverability. DELUXE mode uses logged, regular inserts. To recover data after a failed express-mode load, revert to the most recent level-0 backup. The table type must be STANDARD for this level of recoverability.

For information about restoring tables of RAW or STANDARD logging types, see the *IBM® Informix® Backup and Restore Guide*.

Storage space encryption

You can encrypt storage spaces (dbspaces, blobspaces and smart blobspaces) with Informix Dynamic Server. The data in encrypted storage spaces is unintelligible without the encryption key. Encrypting storage spaces is an effective way to protect sensitive information that is stored on disk.

You must have IBM® Global Security Kit (GSKit) installed to enable storage space encryption. GSKit is installed by default when you install the database server.

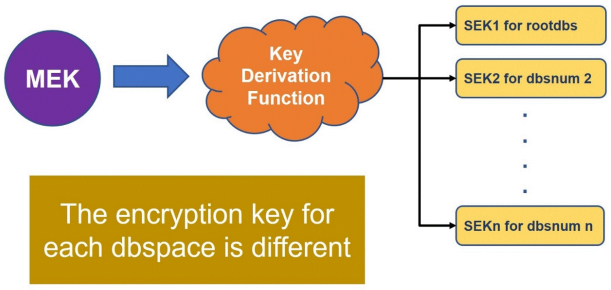
You can enable storage space encryption by setting the DISK_ENCRYPTION configuration parameter.

Each storage space is encrypted separately with its own encryption key. By default, the encryption cipher is set to AES with 128-bit keys. You can specify a stronger encryption key by including the cipher option in the DISK_ENCRYPTION configuration parameter value.

Any storage space that you create when storage space encryption is enabled is automatically encrypted, unless you explicitly specify to create it as unencrypted with the onspaces utility. If you initialize a new database server before setting the DISK_ENCRYPTION configuration parameter, the root dbspace and all storage spaces created before setting DISK_ENCRYPTION are not encrypted. However, you can encrypt unencrypted storage spaces, including the root dbspace, by running a restore that encrypts the spaces.

As mentioned above, each storage space is encrypted with its own *Space Encryption Key (SEK)*. The SEKs are generated by the system (oninit) based on a *Master Encryption Key (MEK)*. The MEK is created by the onkstore utility and can be stored locally in the keystore created by the onkstore, or remotely in a Remote Key Server (RKS). In both cases, you must use the onkstore utility to create a keystore that will contain a MEK or the credentials necessary to reach the MEK at a Remote Key Server.

Figure 1. Storage Space Encryprtion



See The onkstore utility for information on creating and managing keystore files.

Once you have created and verified your keystore file, you enable storage space encryption by setting the DISK_ENCRYPTION configuration parameter to point to the keystore you created and then restarting the database server. The value of the DISK_ENCRYPTION parameter is a comma-separated list of attributes, one of which points to your keystore file. For example:

```
DISK_ENCRYPTION keystore=/work/ifmx/keystores/my_ks,cipher=aes192
```

See The DISK_ENCRYPTION configuration parameter for information on setting this important value correctly.

Using a locally stored Master Encryption Key

When using a local Master Encryption Key (MEK), the MEK is created locally and stored inside the keystore file pointed by the DISK_ENCRYPTION configuration parameter. In this scenario, the MEK is generated locally by the onkstore and placed inside the keystore file.

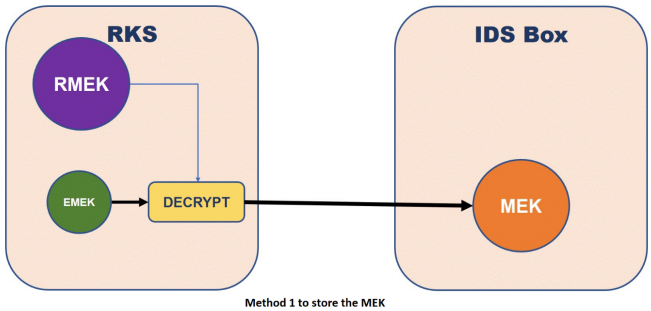
Using a remotely stored Master Encryption Key

The Master Encryption Key can also be created and/or stored in a Remote Key Server. Currently, it is possible to use servers that conform with the KMIP standard and also, selected Cloud Providers (Amazon AWS KMS and Azure KeyVault at this moment). If using a Remote Key Server, the credentials necessary to access the Remote Key Server are recorded inside the keystore file pointed by the DISK_ENCRYPTION configuration parameter.

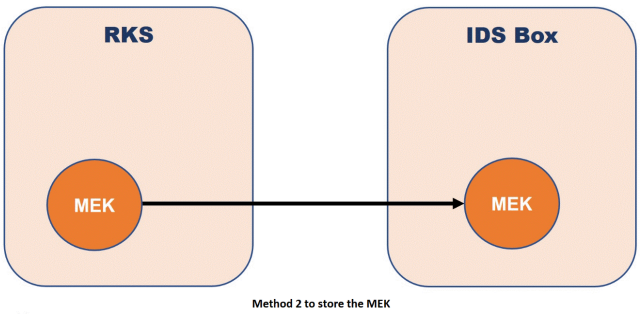
The credentials needed to reach the RKS change depending on the type of RKS you are using. You can see details about this on the description of the onsktore utilty.

If you are using a RKS, there are two options on how we manage the MEK:

- **Option 1**: The MEK is stored at the RKS encrypted using a a Remote Master Encryption Key. The Remote Master Encryption Key (RMEK) is a managed key (managed by the RKS) and never leaves the RKS. Currently, we will use this method with AWS-KMS and Azure KeyVault RKS. In the case of AWS, the MEK is generated remotely at the AWS infrastructure. In the case of Azure, the MEK is generated locally. In bothe cases the MEK will the be encrypted with RMEK and the result stored at the Cloud server.
  Figure 2. Method 1 to manage the MEK



- **Option 2**: The MEK is the Managed Key (Managed by the RKS), its encrypted and protected internally by the RKS. Currently, this method is used with KMIP RKS.
  Figure 3. Method 2 to manage the MEK



Method 2 to store the MEK

Keystore Files May Be Shared

Once you have a valid keystore file, whether a local or network type, it can be used with any number of IDS instances. You are not required to create a new keystore file for an HDR secondary, for example. It can utilize a copy of the keystore file (along with any associated stash file) from the primary, or an entirely different keystore file. There is no forced connection between the encryption keys used for a primary node and HDR or RSS secondaries. However, the SDS secondary must use the same keystore file used by its primary, as they are reading from the same disk. That file may be a duplicate or the same inode, as long as the contents are identical between the file used by the primary and the one used by the SDS secondary.

- Creating encrypted or unencrypted storage spaces
  When storage space encryption is enabled, the storage spaces that you create are encrypted by default, however, you can specify to create unencrypted storage spaces. When storage space encryption is disabled, you cannot create encrypted storage spaces.
- Changing storage space encryption during a restore
  When storage space encryption is enabled, storage spaces are restored with the same encryption state as during the back up, by default. However, you can specify to restore storage spaces as encrypted or unencrypted.
- Monitoring the encryption of storage spaces
  You can monitor whether storage space encryption is enabled and which storage spaces are encrypted.

Creating encrypted or unencrypted storage spaces

When storage space encryption is enabled, the storage spaces that you create are encrypted by default, however, you can specify to create unencrypted storage spaces. When storage space encryption is disabled, you cannot create encrypted storage spaces.

To create an encrypted storage space, set the DISK_ENCRYPTION configuration parameter, restart the database server, and then run the appropriate **onspaces** or SQL administration API command.

To create an unencrypted storage space when storage space encryption is enabled, use one of the following methods:

- Run the appropriate **onspaces** command for the storage space and include the **-u** option.
- Run the appropriate SQL administration API command for the storage space and include the **unencrypt** option.

Changing storage space encryption during a restore

When storage space encryption is enabled, storage spaces are restored with the same encryption state as during the back up, by default. However, you can specify to restore storage spaces as encrypted or unencrypted.

The encryption state of storage spaces on disk does not affect the encryption state of backups. Storage spaces that are encrypted on disk are decrypted before they are sent to the backup front end (on-Bar/ontape). To encrypt the backup, you can use the Integrated backup encryption feature. When you restore a storage space that was encrypted on disk before its backup, the storage space is encrypted during the restore, unless you specify to restore the space as unencrypted. Similarly, you can restore a storage space that was not encrypted on disk by specifying to encrypt the space during the restore.

You can choose to restore some or all storage spaces as encrypted or unencrypted.

The following table shows the ways you can encrypt and decrypt storage spaces during a restore with the ON-Bar or **ontape** utilities when storage space encryption is enabled.

Table 1. Storage space encryption options during a restore

Task	Method
Encrypt or decrypt all existing storage spaces	Run a full restore with the **-encrypt** or **-decrypt** option.
Encrypt or decrypt critical storage spaces	Run a cold restore with the **-encrypt** or **-decrypt** option and specify the spaces with the **-D** option.
Encrypt or decrypt some non-critical storage spaces	Run a warm restore with the **-encrypt** or **-decrypt** option and specify the spaces with the **-D** option.
Encrypt or decrypt all storage spaces for a tenant database (ON-Bar only)	Run a tenant restore with the **onbar -T** command and include the **-encrypt** or **-decrypt** option.

Task	Method
Encrypt or decrypt storage spaces that are created by a roll-forward of logical logs	Include the **rollfwd_create_dbs=encrypt** or **rollfwd_create_dbs=decrypt** option in the DISK_ENCRYPTION configuration parameter value.

When you run a full or a cold restore, new keystore and stash files are created. If you receive an error message that the restore failed because of existing keystore and stash files, follow the instructions in the message and rerun the restore.

During an external restore, storage spaces are restored to the same encryption state as during the backup. You cannot change the encryption state of storage spaces during an external restore.

When storage space encryption is not enabled, you see the following behavior:

- If you attempt to encrypt storage spaces during a restore with the **-encrypt** option, the restore fails.
- If you restore encrypted storage spaces, the storage spaces are restored as unencrypted.

Examples

The following command encrypts all existing storage spaces during a whole-system restore:

```
onbar -r -encrypt -w
```

The following command encrypts two storage spaces during a physical restore:

```
ontape -p -encrypt -D space1 space2
```

The following command decrypts all storage spaces that belong to a tenant database:

```
onbar -T tenant1 -decrypt -t "08-08-2016 00:00:00"
```


Monitoring the encryption of storage spaces

You can monitor whether storage space encryption is enabled and which storage spaces are encrypted.

To determine whether storage space encryption is enabled, run the **oncheck -pr** command. If storage space encryption is enabled, the output includes the following line, which identifies the encryption cipher:

```
Encryption-at-rest is enabled using cipher 'cipher'
```

If the root dbspace is encrypted, the output of the **oncheck -pr** command contains the following line:

```
The ROOT Dbspace is encrypted
```

To see which storage spaces are encrypted, use one of the following methods:

- Run the **onstat -d** command. Position 6 of the `flags` field contains an E if the storage space is encrypted.
- Query the **sysdbspaces** table in the **sysmaster** database. The **is_encrypted** column shows whether a storage space is encrypted.

Logging and log administration

Logging

These topics describe logging of IBM® Informix® databases and addresses the following questions:

- Which database server processes require logging?
- What is transaction logging?
- What database server activity is logged?
- What is the database-logging status?
- Who can set or change the database logging status?

All the databases managed by a single database server instance store their log records in the same logical log, regardless of whether they use transaction logging. Most database users might be concerned with whether transaction logging is buffered or whether a table uses logging.

If you want to change the database-logging status, see Settings or changes for logging status or mode.

Database server processes that require logging

As IBM® Informix® operates, processing transactions, tracking data storage, and ensuring data consistency, automatically generates *logical-log records* for some of the actions that it takes. Most of the time the database server makes no further use of the logical-log records. However, when the database server is required to roll back a transaction, to run a fast recovery after a system failure, for example, the logical-log records are critical. The logical-log records are at the heart of the data-recovery mechanisms.

The database server stores the logical-log records in a *logical log*. The logical log is made up of *logical-log files* that the database server manages on disk until they have been safely transferred offline (*backed up*). The database server administrator keeps the backed up logical-log files until they are required during a data restore, or until the administrator decides that the records are no longer required for a restore. See Logical log for more information about logical logs.

The logical-log records themselves are variable length. This arrangement increases the number of logical-log records that can be written to a page in the logical-log buffer. However, the database server often flushes the logical-log buffer before the page is full. For more information about the format of logical-log records, see the topics about interpreting logical-log records in the *IBM Informix Administrator's Reference*.

The database server uses logical-log records when it performs various functions that recover data and ensure data consistency, as follows:

Transaction rollback
: If a database is using transaction logging and a transaction must be rolled back, the database server uses the logical-log records to reverse the changes made during the transaction. For more information, see Transaction logging.

Fast recovery
: If the database server shuts down in an uncontrolled manner, the database server uses the logical-log records to recover all transactions that occurred since the oldest update not yet flushed to disk and to roll back any uncommitted transactions. (When all the data in shared memory and on disk are the same, they are *physically consistent*.) The database server uses the logical-log records in fast recovery when it returns the entire database server to a state of logical consistency up to the point of the most recent logical-log record. (For more information, see Fast recovery after a checkpoint.)

Data restoration
: The database server uses the most recent storage-space and logical-log backups to recreate the database server system up to the point of the most recently backed-up logical-log record. The logical restore applies all the log records since the last storage-space backup.

Deferred checking
: If a transaction uses the SET CONSTRAINTS statement to set checking to DEFERRED, the database server does not check the constraints until the transaction is committed. If a constraint error occurs while the transaction is being committed, the database server uses logical-log records to roll back the transaction. For more information, see SET Database Object Mode in the *IBM Informix Guide to SQL: Syntax*.

Cascading deletes
: Cascading deletes on referential constraints use logical-log records to ensure that a transaction can be rolled back if a parent row is deleted and the system fails before the children rows are deleted. For information about table inheritance, see the *IBM Informix Database Design and Implementation Guide*. For information about primary key and foreign key constraints, see the *IBM Informix Guide to SQL: Tutorial*.

Distributed transactions
: Each database server involved in a distributed transaction keeps logical-log records of the transaction. This process ensures data integrity and consistency, even if a failure occurs on one of the database servers that is performing the transaction. For more information, see Two-phase commit and logical-log records.

Data Replication
: Data Replication environments that use HDR secondary, SD secondary, and RS secondary servers use logical-log records to maintain consistent data on the primary and secondary database servers so that one of the database servers can be used quickly as a backup database server if the other fails. For more details, see How data replication works.

Enterprise Replication
: You must use database logging with Enterprise Replication because it replicates the data from the logical-log records. For more information, see the .

Transaction logging

A database or table is said to have or use transaction logging when SQL data manipulation statements in a database generate logical-log records.

The database-logging *status* indicates whether a database uses transaction logging. The *log-buffering mode* indicates whether a database uses buffered or unbuffered logging, or ANSI-compliant logging. For more information, see Database-logging status and Manage the database-logging mode.

When you create a database, you specify whether it uses *transaction logging* and, if it does, what log-buffering mechanism it uses. After the database is created, you can turn off database logging or change to buffered logging, for example. Even if you turn off transaction logging for all databases, the database server always logs some events. For more information, see Activity that is always logged and Database logging in an X/Open DTP environment.

You can use logging or nonlogging tables within a database. The user who creates the table specifies the type of table. Even if you use nonlogging tables, the database server always logs some events. For more information, see Table types for Informix.

Logging of SQL statements and database server activity

Three types of logged activity are possible in the database server:

- Activity that is always logged
- Activity logged for databases with transaction logging
- Activity that is not logged
  Some SQL statements are not logged.

Activity that is always logged

Some database operations always generate logical-log records, even if you turn off transaction logging or use nonlogging tables.

The following operations are always logged for permanent tables:

- Certain SQL statements, including SQL data definition statements
- Storage-space backups
- Checkpoints
- Administrative changes to the database server configuration such as adding a chunk or dbspace
- Allocation of new extents to tables
- A change to the logging status of a database
- Smart-large-object operations:
  - Creating
  - Deleting
  - Allocating and deallocating extents
  - Truncating
  - Combining and splitting chunk free list pages
  - Changing the LO header and the LO reference count
- Sbspace metadata
- Blobspaces

The following table lists statements that generate operations that are logged even if transaction logging is turned off.

- ALTER ACCESS_METHOD
- ALTER FRAGMENT
- ALTER FUNCTION
- ALTER INDEX
- ALTER PROCEDURE
- ALTER ROUTINE
- ALTER SECURITY LABEL COMPONENT
- ALTER SEQUENCE
- ALTER TABLE
- ALTER TRUSTED CONTEXT
- ALTER USER
- CLOSE DATABASE
- CREATE ACCESS_METHOD
- CREATE AGGREGATE
- CREATE CAST
- CREATE DATABASE
- CREATE DISTINCT TYPE
- CREATE EXTERNAL TABLE
- CREATE FUNCTION
- CREATE FUNCTION FROM
- CREATE INDEX
- CREATE OPAQUE TYPE
- CREATE OPCLASS
- CREATE PROCEDURE
- CREATE PROCEDURE FROM
- CREATE ROLE
- CREATE ROUTINE FROM
- CREATE ROW TYPE
- CREATE SCHEMA
- CREATE SECURITY LABEL
- CREATE SECURITY LABEL COMPONENT
- CREATE SECURITY POLICY
- CREATE SEQUENCE
- CREATE SYNONYM
- CREATE TABLE
- CREATE TEMP TABLE
- CREATE TRIGGER
- CREATE TRUSTED CONTEXT
- CREATE USER
- CREATE VIEW
- CREATE XADATASOURCE

- CREATE XADATASOURCE TYPE
- DROP ACCESS_METHOD
- DROP AGGREGATE
- DROP CAST
- DROP DATABASE
- DROP FUNCTION
- DROP INDEX
- DROP OPCLASS
- DROP PROCEDURE
- DROP ROLE
- DROP ROUTINE
- DROP ROW TYPE
- DROP SECURITY
- DROP SEQUENCE
- DROP SYNONYM
- DROP TABLE
- DROP TRIGGER
- DROP TRUSTED CONTEXT
- DROP TYPE
- DROP USER
- DROP VIEW
- DROP XADATASOURCE
- DROP XADATASOURCE TYPE
- GRANT
- GRANT FRAGMENT
- RENAME COLUMN
- RENAME DATABASE
- RENAME INDEX
- RENAME SECURITY
- RENAME SEQUENCE
- RENAME TABLE
- RENAME TRUSTED CONTEXT
- RENAME USER
- REVOKE
- REVOKE FRAGMENT
- TRUNCATE
- UPDATE STATISTICS
- SAVE EXTERNAL DIRECTIVES
- SET CONSTRAINTS
- SET Database Object Mode
- SET INDEXES
- SET TRIGGERS
- START VIOLATIONS TABLE
- STOP VIOLATIONS

Activity logged for databases with transaction logging

If a database uses transaction logging, the following SQL statements generate one or more log records. If these statements are rolled back, the rollback also generates log records.

- DELETE
- FLUSH
- INSERT
- LOAD
- MERGE
- PUT
- SELECT INTO TEMP
- UNLOAD
- UPDATE

The following SQL statements generate logs in special situations.

Table 1. SQL statements that generate logs in special situations.

SQL statement	Log record that the statement generates
BEGIN WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
COMMIT WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
ROLLBACK WORK	Returns an error unless the database uses transaction logging. A log record is produced if the transaction does some other logging work.
EXECUTE	Whether this statement generates a log record depends on the command being run.
EXECUTE FUNCTION	Whether this statement generates a log record depends on the function being executed.

SQL statement	Log record that the statement generates
EXECUTE IMMEDIATE	Whether this statement generates a log record depends on the command being run.
EXECUTE PROCEDURE	Whether this statement generates a log record depends on the procedure being executed.

Activity that is not logged

Some SQL statements are not logged.

The following SQL statements do not produce log records, regardless of the database logging mode.

- ALLOCATE COLLECTION
- ALLOCATE DESCRIPTOR
- ALLOCATE ROW
- CLOSE
- CONNECT
- DATABASE
- DEALLOCATE COLLECTION
- DEALLOCATE DESCRIPTOR
- DEALLOCATE ROW
- DECLARE
- DESCRIBE
- DISCONNECT
- FETCH
- FREE
- GET DESCRIPTOR
- GET DIAGNOSTICS
- INFO
- LOCK TABLE
- OPEN
- OUTPUT
- PREPARE
- RELEASE SAVEPOINT
- SAVEPOINT
- SELECT
- SET AUTOFREE
- SET COLLATION
- SET CONNECTION
- SET DATASKIP
- SET DEBUG FILE
- SET DEFERRED_PREPARE
- SET DESCRIPTOR
- SET ENCRYPTION PASSWORD
- SET ISOLATION
- SET LOCK MODE
- SET LOG
- SET OPTIMIZATION
- SET PDQPRIORITY
- SET ROLE
- SET SESSION AUTHORIZATION
- SET STATEMENT CACHE
- SET TRANSACTION
- SET Transaction Mode
- SET USER PASSWORD
- UNLOCK TABLE
- WHENEVER
- SET ENVIRONMENT
- SET EXPLAIN

For temporary tables in temporary dbspaces, nothing is logged, not even the SQL statements that are always logged for other types of tables. If you include temporary (nonlogging) dbspaces in the value of the DBSPACETEMP configuration parameter, the database server places nonlogging tables in these temporary dbspaces first.

Database-logging status

You must use transaction logging with a database to take advantage of any of the features listed in Database server processes that require logging.

Every database that the database server manages has a logging status. The logging status indicates whether the database uses transaction logging and, if so, which log-buffering mechanism the database employs. To find out the transaction-logging status of a database, use the database server utilities, as explained in Monitor the logging mode of a database. The database-logging status indicates any of the following types of logging:

- Unbuffered transaction logging

- Buffered transaction logging
- ANSI-compliant transaction logging
- No logging

All logical-log records pass through the logical-log buffer in shared memory before the database server writes them to the logical log on disk. However, the point at which the database server flushes the logical-log buffer is different for buffered transaction logging and unbuffered transaction logging. For more information, see Figure 1 and Flush the logical-log buffer.

Unbuffered transaction logging

If transactions are made against a database that uses unbuffered logging, the records in the logical-log buffer are guaranteed to be written to disk during commit processing. When control returns to the application after the COMMIT statement (and before the PREPARE statement for distributed transactions), the logical-log records are on the disk. The database server flushes the records as soon as any transaction in the buffer is committed (that is, a commit record is written to the logical-log buffer).

When the database server flushes the buffer, only the used pages are written to disk. Used pages include pages that are only partially full, however, so some space is wasted. For this reason, the logical-log files on disk fill up faster than if all the databases on the same database server use buffered logging.

Unbuffered logging is the best choice for most databases because it guarantees that all committed transactions can be recovered. In the event of a failure, only uncommitted transactions at the time of the failure are lost. However, with unbuffered logging, the database server flushes the logical-log buffer to disk more frequently, and the buffer contains many more partially full pages, so it fills the logical log faster than buffered logging does.

Buffered transaction logging

If transactions are made against a database that uses buffered logging, the records are held (*buffered*) in the logical-log buffer for as long as possible. They are not flushed from the logical-log buffer in shared memory to the logical log on disk until one of the following situations occurs:

- The buffer is full.
- A commit on a database with unbuffered logging flushes the buffer.
- A checkpoint occurs.
- The connection is closed.

If you use buffered logging and a failure occurs, you cannot expect the database server to recover the transactions that were in the logical-log buffer when the failure occurred. Thus, you might lose some committed transactions. In return for this risk, performance during alterations improves slightly. Buffered logging is best for databases that are updated frequently (when the speed of updating is important), as long as you can recreate the updates in the event of failure. You can tune the size of the logical-log buffer to find an acceptable balance for your system between performance and the risk of losing transactions to system failure.

ANSI-compliant transaction logging

The ANSI-compliant database logging status indicates that the database owner created this database using the MODE ANSI keywords. ANSI-compliant databases always use unbuffered transaction logging, enforcing the ANSI rules for transaction processing. You cannot change the buffering status of ANSI-compliant databases.

No database logging

If you turn off logging for a database, transactions are not logged, but other operations are logged. For more information, see Activity that is always logged. Usually, you would turn off logging for a database when you are loading data, or just running queries.

If you are satisfied with your recovery source, you can decide not to use transaction logging for a database to reduce the amount of database server processing. For example, if you are loading many rows into a database from a recoverable source such as tape or an ASCII file, you might not require transaction logging, and the loading would proceed faster without it. However, if other users are active in the database, you would not have logical-log records of their transactions until you reinitiate logging, which must wait for a level-0 backup.

Databases with different log-buffering status

All databases on a database server use the same logical log and the same logical-log buffers. Therefore, transactions against databases with different log-buffering statuses can write to the same logical-log buffer. In that case, if transactions exist against databases with buffered logging and against databases with unbuffered logging, the database server flushes the buffer either when it is full or when transactions against the databases with unbuffered logging complete.

Database logging in an X/Open DTP environment

Databases in the X/Open distributed transaction processing (DTP) environment must use *unbuffered logging*. Unbuffered logging ensures that the database server logical logs are always in a consistent state and can be synchronized with the transaction manager. If a database created with buffered logging is opened in an X/Open DTP environment, the database status automatically changes to unbuffered logging. The database server supports both ANSI-compliant and non-ANSI databases. For more information, see Transaction managers.

Settings or changes for logging status or mode

The user who creates a database with the CREATE DATABASE statement establishes the logging status or buffering mode for that database. For more information about the CREATE DATABASE statement, see the *IBM® Informix® Guide to SQL: Syntax*.

If the CREATE DATABASE statement does not specify a logging status, the database is created without logging.

Only the database server administrator can change logging status. Manage the database-logging mode, describes this topic. Ordinary users cannot change database-logging status.

If a database does not use logging, you are not required to consider whether buffered or unbuffered logging is more appropriate. If you specify logging but do not specify the buffering mode for a database, the default is unbuffered logging.

Users *can* switch from unbuffered to buffered (but not ANSI-compliant) logging and from buffered to unbuffered logging for the *duration of a session*. The SET LOG statement performs this change within an application. For more information about the SET LOG statement, see the *IBM Informix Guide to SQL: Syntax*.

Manage the database-logging mode

You can monitor and modify the database-logging mode.

The topics in this section provide information about:

- Understanding database-logging mode
- Modifying database-logging mode with **ondblog**
- Modifying database-logging mode with **ontape**
- Monitoring transaction logging

As a database server administrator, you can alter the logging mode of a database as follows:

- Change transaction logging from buffered to unbuffered.
- Change transaction logging from unbuffered to buffered.
- Make a database ANSI compliant.
- Add transaction logging (buffered or unbuffered) to a database.
- End transaction logging for a database.

For information about database-logging mode, when to use transaction logging, and when to buffer transaction logging, see Logging. To find out the current logging mode of a database, see Monitor the logging mode of a database.

For information about using SQL administration API commands instead of some **ondblog** and **ontape** commands, see Remote administration with the SQL administration API and the *IBM® Informix® Administrator's Reference*.

- Change the database-logging mode
- Modify the database-logging mode with ondblog
- Modify the database logging mode with ontape
- Modify the table-logging mode
- Monitor transactions
- Monitor the logging mode of a database

Change the database-logging mode

You can use **ondblog** or **ontape** to add or change logging. Then use ON-Bar, or **ontape** to back up the data. When you use ON-Bar or **ontape**, the database server must be in online, administration, or quiescent mode.

For information about ON-Bar and **ontape**, see the *IBM® Informix® Backup and Restore Guide*.

The following table shows how the database server administrator can change the database-logging mode. Certain logging mode changes take place immediately, while other changes require a level-0 backup.

Table 1. Logging mode transitions

Converting from:	Converting to no logging	Converting to unbuffered logging	Converting to buffered logging	Converting to ANSI compliant
No logging	Not applicable	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)	Level-0 backup (of affected storage spaces)
Unbuffered logging	Yes	Not applicable	Yes	Yes
Buffered logging	Yes	Yes	Not applicable	Yes
ANSI compliant	Illegal	Illegal	Illegal	Not applicable

Changing the database-logging mode has the following effects:

- While the logging status is being changed, the database server places an exclusive lock on the database to prevent other users from accessing the database, and frees the lock when the change is complete.
- If a failure occurs during a logging-mode change, check the logging mode in the flags in the **sysdatabases** table in the **sysmaster** database, after you restore the database server data. For more information, see Monitor the logging mode of a database. Then try the logging-mode change again.
- After you choose either buffered or unbuffered logging, an application can use the SQL statement SET LOG to change from one logging mode to the other. This change lasts for the duration of the session. For information about SET LOG, see the *IBM Informix Guide to SQL: Syntax*.
- If you add logging to a database, the change is not complete until the next level-0 backup of all the storage spaces for the database.

Modify the database-logging mode with ondblog

You can use the **ondblog** utility to change the logging mode for one or more databases. If you add logging to a database, you must create a level-0 backup of the dbspace(s) that contains the database before the change takes effect. For more information, see the topics on using **ondblog** in the *IBM® Informix® Administrator's Reference*.

- Change the buffering mode with ondblog
- Cancel a logging mode change with ondblog
- End logging with ondblog
- Make a database ANSI compliant with ondblog
- Changing the logging mode of an ANSI-compliant database

Change the buffering mode with ondblog

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, run the following command:

```
ondblog unbuf stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, run the following command:

```
ondblog buf stores_demo
```


Cancel a logging mode change with ondblog

To cancel the logging mode change request before the next level-0 backup occurs, run the following command:

```
ondblog cancel stores_demo
```

You cannot cancel the logging changes that are executed immediately.

End logging with ondblog

To end logging for two databases that are listed in a file called dbfile, run the following command:

```
ondblog nolog -f dbfile
```


Make a database ANSI compliant with ondblog

To make a database called **stores_demo** into an ANSI-compliant database with **ondblog**, run the following command:

```
ondblog ansi stores_demo
```


Changing the logging mode of an ANSI-compliant database

After you create or convert a database to ANSI mode, you cannot easily change it to any other logging mode. If you accidentally convert a database to ANSI mode, follow these steps to change the logging mode:

To change the logging mode:

1. To unload the data, use **dbexport** or any other migration utility. The **dbexport** utility creates the schema file.
   For information about how to load and unload data, see the *IBM® Informix® Migration Guide*.
2. To recreate a database with buffered logging and load the data, use the **dbimport -l buffered** command.
   To recreate a database with unbuffered logging and load the data, use the **dbimport -l** command.

Modify the database logging mode with ontape

If you use **ontape** as your backup tool, you can use **ontape** to change the logging mode of a database.

- Turn on transaction logging with ontape
- End logging with ontape
- Change buffering mode with ontape
- Make a database ANSI compliant with ontape

Turn on transaction logging with ontape

Before you modify the database-logging mode, read Change the database-logging mode.

You add logging to a database with **ontape** at the same time that you create a level-0 backup.

For example, to add buffered logging to a database called **stores_demo** with **ontape**, run the following command:

```
ontape -s -B stores_demo
```

To add unbuffered logging to a database called **stores_demo** with **ontape**, run the following command:

```
ontape -s -U stores_demo
```

In addition to turning on transaction logging, these commands create full-system storage-space backups. When **ontape** prompts you for a backup level, specify a level-0 backup.
Tip: With **ontape**, you must perform a level-0 backup of all storage spaces.

End logging with ontape

To end logging for a database called **stores_demo** with **ontape**, run the following command:

```
ontape -N stores_demo
```

Change buffering mode with ontape

To change the buffering mode from buffered to unbuffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, run the following command:

```
ontape -U stores_demo
```

To change the buffering mode from unbuffered to buffered logging on a database called **stores_demo**, using **ontape**, without creating a storage-space backup, run the following command:

```
ontape -B stores_demo
```


Make a database ANSI compliant with ontape

To make a database called **stores_demo**, which already uses transaction logging (either unbuffered or buffered), into an ANSI-compliant database with **ontape**, run the following command:

```
ontape -A stores_demo
```

To make a database called **stores_demo**, which does not already use transaction logging, into an ANSI-compliant database with **ontape**, run the following command:

```
ontape -s -A stores_demo
```

In addition to making a database ANSI compliant, this command also creates a storage-space backup at the same time. Specify a level-0 backup when you are prompted for a level.
Tip: After you change the logging mode to ANSI compliant, you cannot easily change it again. To change the logging mode of ANSI-compliant databases, unload the data, recreate the database with the new logging mode, and reload the data. For details, see Changing the logging mode of an ANSI-compliant database.

Modify the table-logging mode

The database server creates standard tables that use logging by default. To create a nonlogging table, use the CREATE TABLE statement with the WITH LOG clause. For information about the CREATE TABLE and ALTER TABLE statements, see the *IBM Informix Guide to SQL: Syntax*. For more information, see Table types for Informix.

- Alter a table to turn off logging
- Alter a table to turn on logging
- Disable logging on temporary tables

Alter a table to turn off logging

To switch a table from logging to nonlogging, use the SQL statement ALTER TABLE with the TYPE option of RAW. For example, the following statement changes table **tablog** to a RAW table:

```
ALTER TABLE tablog TYPE (RAW)
```


Alter a table to turn on logging

To switch from a nonlogging table to a logging table, use the SQL statement ALTER TABLE with the TYPE option of STANDARD. For example, the following statement changes table **tabnolog** to a STANDARD table:

```
ALTER TABLE tabnolog TYPE (STANDARD)
```

Important: When you alter a table to STANDARD, you turn logging on for that table. After you alter the table, perform a level-0 backup if you must be able to restore the table.

Disable logging on temporary tables

You can disable logging on temporary tables to improve performance and to prevent IBM® Informix® from transferring temporary tables when using a primary server in a data replication environment such as with HDR secondary, RS secondary, and SD secondary servers.

To disable logging on temporary tables, set the TEMPTAB_NOLOG configuration parameter to `1`.

For HDR, RSS, and SDS secondary servers in a high-availability cluster, logical logging on temporary tables must always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1 or 2.

You can use the **onmode -wf** command to change the value of TEMPTAB_NOLOG.

Related information:
TEMPTAB_NOLOG configuration parameter

Monitor transactions

This topic contains references for information about ways to monitor transactions.

Command	Description	Reference
onstat -x	Monitor transactions.	Monitor a global transaction
onstat -g sql	Monitor SQL statements, listed by session ID and database.	Performance monitoring in the *IBM® Informix® Performance Guide*
onstat -g spf	Display detailed information about SQL queries.	
onstat -g stm	Monitor memory usage of prepared SQL statements.	Memory utilization in the *IBM Informix Performance Guide*

Monitor the logging mode of a database

These topics explain ways to monitor the logging mode of your database and tables.

- Monitor the logging mode with SMI tables

Monitor the logging mode with SMI tables

Query the **sysdatabases** table in the **sysmaster** database to determine the logging mode. This table contains a row for each database that the database server manages. The **flags** field indicates the logging mode of the database. The **is_logging**, **is_buff_log**, and **is_ansi** fields indicate whether logging is active, and whether buffered logging or ANSI-compliant logging is used. For a description of the columns in this table, see the **sysdatabases** section in the chapter about the **sysmaster** database in the *IBM® Informix® Administrator's Reference*.

Logical log

The information in Logging, and these topics explains how the database server uses the logical log. For information about how to perform logical-log tasks, see Manage logical-log files, and Manage the database-logging mode.

- What is the logical log?
- Location of logical-log files
  When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace.
- Identification of logical-log files
- Status flags of logical-log files
- Size of the logical-log file
- Dynamic log allocation
- Freeing of logical-log files
- Log blobspaces and simple large objects
- Log sbspaces and smart large objects
- Logging process

Related reference:
Database server maintenance tasks

What is the logical log?

To keep a history of transactions and database server changes since the time of the last storage-space backup, the database server generates log records. The database server stores the log records in the *logical log*, a circular file that is composed of three or more logical-log files. The log is called *logical* because the log records represent logical operations of the database server, as opposed to physical operations. At any time, the combination of a storage-space backup plus logical-log backup contains a complete copy of your database server data.

As the database server administrator, you must configure and manage the logical log. For example, if you do not back up the log files regularly, the logical log fills and the database server suspends processing.

These responsibilities include the following tasks:

- Choosing an appropriate location for the logical log
  See Location of logical-log files.
- Monitoring the logical-log file status
  See Identification of logical-log files.
- Allocating an appropriate amount of disk space for the logical log
  See Size of the logical-log file.
- Allocating additional log files whenever necessary
  See Allocate logical log files.
- Backing up the logical-log files to media
  See Back up logical-log files and Freeing of logical-log files.
- Managing logging of blobspaces and sbspaces
  See Log blobspaces and simple large objects and Log sbspaces and smart large objects.

Location of logical-log files

When the database server initializes disk space, it places the logical-log files and the physical log in the root dbspace.

To improve performance by reducing the number of writes to the root dbspace and minimize contention, move the logical-log files out of the root dbspace to a dbspace on a disk that is not shared by active tables or the physical log.

To improve performance further, separate the logical-log files into two groups and store them on two separate disks (neither of which contains data). For example, if you have six logical-log files, you might locate files 1, 3, and 5 on disk 1, and files 2, 4, and 6 on disk 2. This arrangement improves performance because the same disk drive never is required to handle writes to the current logical-log file and backups at the same time.

The logical-log files contain critical information and must be mirrored for maximum data protection. If you move logical-log files to a different dbspace, plan to start mirroring on that dbspace.

Related concepts:
Move logical-log files

Identification of logical-log files

Each logical-log file, whether backed up to media or not, has a unique ID number. The sequence begins with `1` for the first logical-log file filled after you initialize the database server disk space. When the current logical-log file becomes full, the database server switches to the next logical-log file and increments the unique ID number for the new log file by one. Log files that are newly added or marked for deletion have unique ID numbers of `0`.

The actual disk space allocated for each logical-log file has an identification number known as the *log file number*. For example, if you configure six logical-log files, these files have log numbers one through six. The log numbers might be out of sequence. As logical-log files are backed up and freed, the database server reuses the disk space for the logical-log files.

The following table illustrates the relationship between the log numbers and the unique ID numbers. Log 7 is inserted after log 5 and used for the first time in the second rotation.

Table 1. Logical-log file-numbering sequence

Log file number	First rotation unique ID number	Second rotation unique ID number	Third rotation unique ID number
1	1	7	14
2	2	8	15
3	3	9	16
4	4	10	17
5	5	11	18
7	0	12	19

Log file number	First rotation unique ID number	Second rotation unique ID number	Third rotation unique ID number
6	6	13	20

Status flags of logical-log files

All logical-log files have one of the following status flags in the first position: Added (**A**), Deleted (**D**), Free (**F**), or Used (**U**). The following table shows the possible log-status flag combinations.

Table 1. Logical-log status flags

Status flag	Status of logical-log file
A------	Log file has been added, and is available, but has not yet been used.
D------	If you drop a log file with a status of U-B, it is marked as deleted. This log file is dropped and its space is freed for reuse when you take a level-0 backup of all storage spaces.
F------	Log file is free and available for use. A logical-log file is freed after it is backed up, all transactions within the logical-log file are closed, and the oldest update stored in this file is flushed to disk.
U	Log file has been used but not backed up.
U-B----	Log file is backed up but still required for recovery. (The log file is freed when it is no longer required for recovery.)
U-B---L	Log is backed up but still required for recovery. Contains the last checkpoint record.
U---C	The database server is currently filling the log file.
U---C-L	This current log file contains the last checkpoint record.

Use the **onstat -l** command to list the log files by number and monitor the status flags and percentage of log space used. For more details, see The onstat -l command.

Size of the logical-log file

The minimum size for a logical-log file is 200 KB.

The maximum size for a logical-log file is 524288 pages (equivalent to 0x7ffff + 1), with a 2 KB or 4 KB base-page size, depending on the operating system. To determine the database server's base-page size on your operating system, run **onstat -d** and then check the `pgsize` value for the root dbspace.

Determine the size and number of log files to use. If you allocate more disk space than necessary, space is wasted. If you do not allocate enough disk space, however, performance might be adversely affected. Use larger log files when many users are writing to the logs at the same time.

Note: Smaller log files mean that you can recover to a later time if the disk that contains the log files goes down. If continuous log backup is set, log files are automatically backed up as they fill. Smaller logs result in slightly longer logical recovery.

- Number of logical-log files
- Performance considerations

Number of logical-log files

When you think about the number of logical-log files, consider these points:

- You must always have at least three logical-log files and a maximum of 32,767 log files.
- The number of log files affects the frequency of logical-log backups.
- The number of logical-log files affects the rate at which blobspace blobpages can be reclaimed. See Back up log files to free blobpages.

Performance considerations

For a given level of system activity, the less logical-log disk space that you allocate, the sooner that logical-log space fills up, and the greater the likelihood that user activity is blocked due to backups and checkpoints. Tune the logical-log size to find the optimum value for your system.

- Logical-log backups
  When the logical-log files fill, you must back them up. The backup process can hinder transaction processing that involves data located on the same disk as the logical-log files. Put the physical log, logical logs, and user data on separate disks. (See the *IBM® Informix® Backup and Restore Guide*.)

- Size of the logical log
  A smaller logical log fills faster than a larger logical log. You can add a larger logical-log file, as explained in Adding logical-log files manually.
- Size of individual logical-log records
  The sizes of the logical-log records vary, depending on both the processing operation and the database server environment. In general, the longer the data rows, the larger the logical-log records. The logical log contains images of rows that have been inserted, updated, or deleted. Updates can use up to twice as much space as inserts and deletes because they might contain both before-images and after-images. (Inserts store only the after-image and deletes store only the before-image.)
- Number of logical-log records
  The more logical-log records written to the logical log, the faster it fills. Databases with transaction logging fill the logical log faster than transactions against databases without transaction logging.
- Type of log buffering
  Databases that use unbuffered transaction logging fill the logical log faster than databases that use buffered transaction logging.
- Enterprise Replication on a table
  Because Enterprise Replication generates before-images and after-images of the replicated tables, it might cause the logical log to fill.
- Frequency of rollbacks
  More rollbacks fill the logical log faster. The rollbacks themselves require logical-log file space although the rollback records are small.
- Number of smart large objects
  Smart large objects that have user data logging enabled and have a large volume of user data updates can fill logical logs at a prodigious rate. Use temporary smart large objects if you do not want to log the metadata.

Dynamic log allocation

Dynamic log allocation prevents log files from filling and hanging the system during long transaction rollbacks. The only time that this feature becomes active is when the next log file contains an open transaction. (A *transaction* is *long* if it is not committed or rolled back when it reaches the long-transaction high-watermark.)

The database server automatically (dynamically) allocates a log file after the current log file when the next log file contains an open transaction. You can use dynamic log allocation for the following actions:

- Add a log file while the system is active
- Insert a log file after the current log file
- Immediately access new log files even if the root dbspace is not backed up

The best way to test dynamic log allocation is to produce a transaction that spans all the log files and then use **onstat -l** to check for newly added log files. For more information, see Allocate logical log files.
Important: You still must back up log files to prevent them from filling. If the log files fill, the system hangs until you perform a backup.

Freeing of logical-log files

Each time the database server commits or rolls back a transaction, it attempts to free the logical-log file in which the transaction began. The following criteria must be satisfied before the database server frees a logical-log file for reuse:

- The log file is backed up.
- No records within the logical-log file are associated with open transactions.
- The logical-log file does not contain the oldest update not yet flushed to disk.

- Action if the next logical-log file is not free
- Action if the next log file contains the last checkpoint

Action if the next logical-log file is not free

If the database server attempts to switch to the next logical-log file but finds that the next log file in sequence is still in use, the database server immediately suspends all processing. Even if other logical-log files are free, the database server cannot skip a file in use and write to a free file out of sequence. Processing stops to protect the data within the logical-log file.

The logical-log file might be in use for any of the following reasons:

- The file contains the latest checkpoint or the oldest update not yet flushed to disk.
  Issue the **onmode -c** command to perform a checkpoint and free the logical-log file. For more information, see Force a checkpoint.
- The file contains an open transaction.
  The open transaction is the long transaction explained in Controlling long transactions.
- The file is not backed up.

If the logical-log file is not backed up, processing resumes when you use ON-Bar or **ontape** to back up the logical-log files.

Action if the next log file contains the last checkpoint

The database server does not suspend processing when the next log file contains the last checkpoint or the oldest update. The database server always forces a checkpoint when it switches to the last available log, if the previous checkpoint record or oldest update that is not yet flushed to disk is located in the log that follows the last available log. For example, if four logical-log files have the status shown in the following list, the database server forces a checkpoint when it switches to logical-log file 3.

Log file number
Logical-log file status
1
U-B----
2
U---C--
3
F
4
U-B---L

Log blobspaces and simple large objects

Simple-large-object data (TEXT and BYTE data types) is potentially too voluminous to include in a logical-log record. If simple large objects are always logged, they might be so large that they slow down the logical log.

The database server assumes that you designed your databases so that smaller simple large objects are stored in dbspaces and larger simple large objects are stored in blobspaces:

- The database server includes simple-large-object data in log records for simple large objects stored in dbspaces.
- The database server does not include simple-large-object data in log records for simple large objects stored in blobspaces. The logical log records blobspace data only when you back up the logical logs.

To obtain better overall performance for applications that perform frequent updates of simple large objects in blobspaces, reduce the size of the logical log. Smaller logs can improve access to simple large objects that must be reused. For more information, see the chapter on configuration effects on I/O utilization in your *IBM® Informix® Performance Guide*.

- Switch log files to activate blobspaces
- Back up log files to free blobpages
- Back up blobspaces after inserting or deleting TEXT and BYTE data

Switch log files to activate blobspaces

You must switch to the next logical-log file in these situations:

- After you create a blobspace, if you intend to insert simple large objects in the blobspace right away
- After you add a new chunk to an existing blobspace, if you intend to insert simple large objects in the blobspace that uses the new chunk

The database server requires that the statement that creates a blobspace, the statement that creates a chunk in the blobspace, and the statements that insert simple large objects into that blobspace are created in separate logical-log files. This requirement is independent of the database-logging status.

For instructions on switching to the next log file, see Switch to the next logical-log file.

Back up log files to free blobpages

When you delete data stored in blobspace pages, those pages are not necessarily freed for reuse. The blobspace pages are free only when both of the following actions have occurred:

- The TEXT or BYTE data has been deleted, either through an UPDATE to the column or by deleting the row
- The logical log that stores the INSERT of the row that has TEXT or BYTE data is backed up

Back up blobspaces after inserting or deleting TEXT and BYTE data

Be sure to back up all blobspaces and logical logs containing transactions on simple large objects stored in a blobspace. During log backup, the database server uses the data pointer in the logical log to copy the changed text and byte data from the blobspace into the logical log.

Log sbspaces and smart large objects

Sbspaces, described in Sbspaces, contain two components: metadata and user data. By default, sbspaces are not logged.

The metadata component of the sbspace describes critical characteristics of smart large objects stored in a particular sbspace. The metadata contains pointers to the smart large objects. If the metadata were to be damaged or become inaccessible, the sbspace would be corrupted and the smart large objects within that sbspace would be unrecoverable.

Metadata in a standard sbspace is always logged, even if logging is turned off for a database. Logging sbspace metadata ensures that the metadata can always be recovered to a consistent transaction state. However, metadata in a temporary sbspace is not logged.

- Sbspace logging
- Smart-large-object log records
- Prevent long transactions when logging smart-large-object data

Sbspace logging

When an sbspace is logged, the database server slows down, and the logical logs fill up quickly. If you use logging for sbspaces, you must ensure that the logical logs are large enough to hold the logging data. For more information, see Estimate the log size when logging smart large objects.

When you turn on logging for a database, the database server does not begin logging until you perform a level-0 backup. However, when you turn on logging for a smart large object, the database server begins logging changes to it immediately. To reduce the volume of log entries, load smart large objects with logging turned off and then turn logging back on to capture updates to the smart large objects.
Important: When you turn logging on for a smart large object, you must immediately perform a level-0 backup to be able to recover and restore the smart large object. For more information, see Back up sbspaces and the *IBM® Informix® Backup and Restore Guide*.

- Logging for smart large objects
- Logging for updated smart large objects
- Turn logging on or off for an sbspace
  You can control whether logging is on of off for an sbspace with several different methods.

Logging for smart large objects

Use logging for smart large objects if users are updating the data frequently or if the ability to recover any updated data is critical. The database server writes a record of the operation (insert, update, delete, read, or write) to the logical-log buffer. The modified portion of the CLOB or BLOB data is included in the log record.

To increase performance, turn off logging for smart large objects. Also turn off logging if users are primarily analyzing the data and updating it infrequently, or if the data is not critical to recover.

Logging for updated smart large objects

When you update a smart large object, the database server does not log the entire object. Assume that the user is writing X bytes of data at offset Y with logging enabled for smart large objects. The database server logs the following information:

- If Y is set to the end of the large object, the database server logs X bytes (the updated byte range).
- If Y is at the beginning or in the middle of the large object, the database server logs the smallest of these choices:
  - Difference between the old and new image
  - Before-image and after-image
  - Nothing is logged if the before- and after-images are the same

Turn logging on or off for an sbspace

You can control whether logging is on of off for an sbspace with several different methods.

If you want to use logging in an sbspace, specify the **-Df "LOGGING=ON"** option of the **onspaces** command when you create the sbspace. If logging is turned off in the sbspace, you can turn on logging for smart large objects in specific columns. One column that contains smart large objects can have logging turned on while another column has logging turned off.

To verify that smart large objects in an sbspace are logged, use the **oncheck -pS** *sbspace_name* **| grep "Create Flags"** command.

If you create smart large objects in the sbspace with the default logging option and you see the LO_NOLOG flag in the output, the smart large objects in this sbspace are not logged. If you see the LO_LOG flag in the output, all smart large objects in this sbspace are logged.

You can modify the logging status of an sbspace in any of the following ways.

Function or statement to specify	Logging action	References
onspaces -ch -Df "LOGGING=ON" **onspaces -ch -Df "LOGGING=OFF"**	Turns logging on or off for an existing sbspace	Alter storage characteristics of smart large objects onspaces -ch: Change sbspace default specifications
The SQL administration API **task()** or **admin()** function with the set sbspace logging on or set sbspace logging off argument	Turns logging on or off for an existing sbspace	set sbspace logging argument: Change the logging of an sbspace (SQL administration API)
LOG option in the PUT clause of the CREATE TABLE or alter table statement	Turns on logging for all smart large objects that you load into the column	Logging PUT Clause
mi_lo_create DataBlade API function	Turns off logging for a smart large object when it is initially loaded	*IBM® Informix DataBlade API Function Reference*
mi_lo_alter DataBlade API function	Turns on logging after the load is complete	*IBM Informix DataBlade API Function Reference*
ifx_lo_create Informix® ESQL/C function	Turns off logging for a smart large object when it is initially loaded	*IBM Informix ESQL/C Programmer's Manual*
ifx_lo_alter Informix ESQL/C function	Turns on logging after the load is complete	*IBM Informix ESQL/C Programmer's Manual*

Smart-large-object log records

When you create a smart large object with the LOG option, the logical log creates a *smart-blob log record*. Smart-blob log records track changes to user data or metadata. When smart large objects are updated, only the modified portion of the sbpage is in the log record. User-data log records are created in the logical log only when logging is enabled for the smart large object.
Warning: Be careful about enabling logging for smart large objects that are updated frequently. This logging overhead might significantly slow down the database server.
For information about the log records for smart large objects, see the chapter on interpreting logical-log records in the *IBM® Informix® Administrator's Reference*.

Prevent long transactions when logging smart-large-object data

You can use smart large objects in situations where the data collection process for a single smart large object lasts for long periods of time. Consider, for example, an application that records many hours of low-quality audio information. Although the amount of data collected might be modest, the recording session might be long, resulting in a long-transaction condition.
Tip: To prevent long transactions from occurring, periodically commit writes to smart large objects.

Logging process

These topics describe in detail the logging process for dbspaces, blobspaces, and sbspaces. This information is not required for performing normal database server administration tasks.

- Dbspace logging
- Blobspace logging

Dbspace logging

The database server uses the following logging process for operations that involve data stored in dbspaces:

1. Reads the data page from disk to the shared-memory page buffer
2. Copies the unchanged page to the physical-log buffer, if required
3. Writes the new data to the page buffer and creates a logical-log record of the transaction, if required
4. Flushes the physical-log buffer to the physical log on disk
5. Flushes the logical-log buffer to a logical-log file on disk
6. Flushes the page buffer and writes it back to disk

Blobspace logging

The database server logs blobspace data, but the data does not pass through either shared memory or the logical-log files on disk. The database server copies data stored in a blobspace directly from disk to tape. Records of modifications to the blobspace overhead pages (the free-map and bitmap pages) are the only blobspace data that reaches the logical log.

Manage logical-log files

You must manage logical-log files even if none of your databases uses transaction logging. See Logical log for background information about logical logs.

You must log-in as either **informix** or **root** on UNIX to make any of the changes described in this chapter. You must be a member of the **Informix-Admin** group on Windows.

You perform these tasks when setting up your logical log:

- Before you initialize or restart the database server, use the LOGFILES parameter to specify the number of logical-log files to create.
- After the database server is online, estimate the size and number of logical-log files that your system requires.
  See Estimate the size and number of log files.

- If you do not want to use the default values, change the LOGSIZE and LOGBUFF configuration parameters.
- Add the estimated number of logical-log files.
  See Allocate logical log files.

You perform the following tasks routinely:

- Backing up a logical-log file
- Switching to the next logical-log file
- Freeing a logical-log file
- Monitoring logging activity and log-backup status

You perform these tasks occasionally, if necessary:

- Adding a logical-log file
- Dropping a logical-log file
- Changing the size of a logical-log file
- Moving a logical-log file
- Changing the logical-log configuration parameters
- Monitoring event alarms for logical logs
- Setting high-watermarks for transactions

For information about using SQL administration API commands instead of some **oncheck**, **onmode**, **onparams** and **onspaces** commands, see Remote administration with the SQL administration API and the *IBM® Informix Administrator's Reference*.

- Estimate the size and number of log files
- Back up logical-log files
- Switch to the next logical-log file
- Free a logical-log file
- Monitor logging activity
- Allocate logical log files
  When you initialize or restart the database server, it creates the number of logical-log files that are specified by the LOGFILES configuration parameter. The size of the logical log files is specified by the LOGSIZE configuration parameter.
- Dropping logical-log files
  You can use an **onparams** command to drop logical-log files.
- Change the size of logical-log files
- Move logical-log files
  You might want to move logical-log files for performance reasons or to make more space in the dbspace.
- Display logical-log records
- Controlling long transactions

Estimate the size and number of log files

Use the LOGSIZE configuration parameter to set the size of the logical-log files.

The amount of log space that is optimal for your database server system depends on the following factors:

- Your application requirements and the amount of update activity your applications experience. Increased update activity requires increased log space.
- The recovery time objective (RTO) standards for the amount of time, in seconds, that the server is given to recover from a problem after you restart the server and bring it into online or quiescent mode.
  In the case of a catastrophic event, consider how much data loss you can tolerate. More frequent log backups, which reduce the risk of data and transaction loss, require increased log space.
- Whether you use Enterprise Replication or data replication configurations such as HDR secondary, SD secondary or RS secondary servers.
  These replication services can influence the number and size of log files. If your system uses any of these replication services, see guidelines in High-availability cluster configuration or in the .

Some guidelines for determining log size are:

- Generally, you can more easily manage a few large log files than you can manage many small log files.
- Having too much log space does not affect performance. However, not having enough log files and log space can affect performance, because the database server triggers frequent checkpoints.
- Smart large objects in blobspaces are not logged, but they are included in the log backup in which the object was created. This means that the objects are not freed until the server backs up the log in which they were created. Therefore, if smart large objects in a blobspace are frequently updated, you might require more frequent log backups to acquire additional free space within a blobspace.
- For applications that generate a small amount of log data, start with 10 log files of 10 megabytes each.
- For applications that generate a large amount of log data, start with 10 log files with 100 megabytes.

There are two ways to maintain an RTO policy, which determines the tolerance for loss of data in case of a catastrophic event such as the loss of the data server:

- One way to maintain an RTO policy is to use automatic log backups that trigger log backups whenever a log file fills up. This limits data loss to the transactions contained in the log file during the backup, plus any additional transactions that occur during the log backup.
- Another way to maintain an RTO policy is to use the Scheduler. You can create a task that automatically backs up any new log data at timed intervals since the last log backup. This limits data loss to the transactions not backed up between time intervals. For information about using the Scheduler, see The Scheduler.

If an RTO policy is required, you can use the Scheduler to insert a task that executes at an appropriate frequency to maintain the policy. This automatically backs up log files at certain times within the daily cycle. If the log space fills before the logs being backed up and recycled, you can back up the logs and add a new log file to allow transaction processing to continue, or you can use the Scheduler to add a new task to detect this situation and perform either operation automatically.

You can add log files at any time, and the database server automatically adds log files when required for transaction consistency, for example, for long transactions that might consume large amounts of log space.

The easiest way to increase the amount of space for the logical log is to add another logical-log file. See Adding logical-log files manually.

The following expression provides an example total-log-space configuration, in KB:

```
LOGSIZE = (((connections * maxrows) * rowsize) / 1024) / LOGFILES
```

Expression element	Explanation
LOGSIZE	Specifies the size of each logical-log file in KB.
connections	Specifies the maximum number of connections for all network types that you specify in the sqlhosts file or registry and in the NETTYPE parameter. If you configured more than one connection by setting multiple NETTYPE configuration parameters in your configuration file, add the users fields for each NETTYPE, and substitute this total for *connections* in the preceding formula.
maxrows	Specifies the largest number of rows to be updated in a single transaction.
rowsize	Specifies the average size of a table row in bytes. To calculate the *rowsize*, add the length (from the **syscolumns** system catalog table) of the columns in the row.
1024	Converts the LOGSIZE to the specified units of KB.
LOGFILES	Specifies the number of logical-log files.

- Estimate the log size when logging smart large objects
- Estimate the number of logical-log files

Estimate the log size when logging smart large objects

If you plan to log smart-large-object user data, you must ensure that the log size is considerably larger than the amount of data being written. If you store smart large objects in standard sbspaces, the metadata is always logged, even if the smart large objects are not logged. If you store smart large objects in temporary sbspaces, there is no logging at all.

Estimate the number of logical-log files

The LOGFILES parameter provides the number of logical-log files at system initialization or restart. If all your logical-log files are the same size, you can calculate the total space allocated to the logical-log files as follows:

```
total logical log space = LOGFILES * LOGSIZE
```

If the database server contains log files of different sizes, you cannot use the (LOGFILES * LOGSIZE) expression to calculate the size of the logical log. Instead, you must add the sizes for each individual log file on disk. Check the **size** field in the **onstat -l** output. For more information, see The onstat -l command.

For information about LOGSIZE, LOGFILES, and NETTYPE, see the topics about configuration parameters in the *IBM® Informix® Administrator's Reference*.

Back up logical-log files

The logical logs contain a history of the transactions that have been performed. The process of copying a logical-log file to media is called *backing up* a logical-log file. Backing up logical-log files achieves the following two objectives:

- It stores the logical-log records on media so that they can be rolled forward if a data restore is required.
- It makes logical-log-file space available for new logical-log records.
  If you neglect to back up the log files, you can run out of log space.

You can initiate a manual logical-log backup or set up continuous logical-log backups. After you restore the storage spaces, you must restore the logical logs to bring the data to a consistent state. For more information about log backups, see the *IBM® Informix® Backup and Restore Guide*.

- Backing up blobspaces
- Back up sbspaces

Backing up blobspaces

It does not matter whether you back up the logical logs or blobspaces first.

To back up blobspace data:

1. Close the current logical log if it contains transactions on simple large objects in a blobspace.
2. Perform a backup of the logical logs and blobspace as soon as possible after updating simple-large-object data.

Warning: If you do not back up these blobspaces and logical logs, you might not be able to restore the blobspace data. If you wait until a blobspace is down to perform the log backup, the database server cannot access the blobspace to copy the changed data into the logical log.

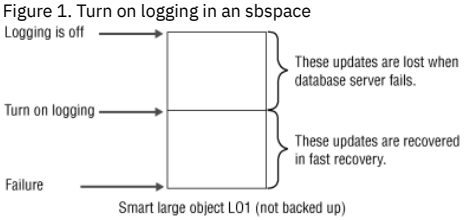
Back up sbspaces

When you turn on logging for smart large objects, you must perform a level-0 backup of the sbspace.

The following figure shows what happens if you turn on logging in an sbspace that is not backed up. The unlogged changes to smart large object **LO1** are lost during the failure, although the logged changes are recoverable. You cannot fully restore **LO1**.

During fast recovery, the database server rolls forward all committed transactions for **LO1**. If **LO1** is unlogged, the database server would be unable to roll back uncommitted transactions. Then the **LO1** contents would be incorrect. For more information, see Fast recovery.

Figure 1. Turn on logging in an sbspace



Switch to the next logical-log file

You might want to switch to the next logical-log file before the current log file becomes full for the following reasons:

- To back up the current log
- To activate new blobspaces and blobspace chunks

The database server can be in online mode to make this change. Run the following command to switch to the next available log file: **onmode -l**

The change takes effect immediately. (Be sure that you type a lowercase L on the command line, not a number 1.)

Free a logical-log file

If a log file is newly added (status **A**), it is immediately available for use. It also can be dropped immediately.

You might want to free a logical-log file for the following reasons:

- So that the database server does not stop processing
- To free the space used by deleted blobpages

The procedures for freeing log files vary, depending on the status of the log file. Each procedure is described in the following topics. To find out the status of logical-log files, see Status flags of logical-log files and Monitor logging activity.
Tip: For information using ON-Bar or **ontape** to back up storage spaces and logical logs, see the *IBM® Informix® Backup and Restore Guide*.

- Delete a log file with status D
- Free a log file with status U
- Freeing a log file with status U-B or F
  If a log file is backed up but still in use (status U-B), some transactions in the log file are still under way, or the log file contains the oldest update that is required for fast recovery. Because a log file with status F has been used in the past, it follows the same rules as for status U-B.
- Freeing a log file with status U-C or U-C-L
- Free a log file with status U-B-L
  If a log file is backed up and all transactions within it are closed but the file is not free (status U-B-L), this logical-log file contains the most-recent checkpoint record. You can free log files with a status U-B-L.

Delete a log file with status D

When you drop a used log file, it is marked as deleted (status D) and cannot be used again, and **onparams** prints this message:

```
Log file log_file_number has been pre-dropped. It will be
deleted from the log list and its space can be reused
once you take level 0 archives of all BLOBspaces,
Smart BLOBspaces and non-temporary DBspaces.
```

The level 0 archive is necessary to make sure that the log file itself and all of the associated information in the different dbspaces has been archived. The log file is deleted at the end of the level 0 archive; however, because the removal of the log file is itself a change in the root reserved pages structure on the disk, the next archive to be taken also must be a level 0 archive. The level 0 archive must occur before a level 1 or level 2 archive can be performed.

Free a log file with status U

If a log file contains records, but is not yet backed up (status U), back up the file using the backup tool that you usually use.

If backing up the log file does not change the status to free (F), its status changes to either U-B or U-B-L. See Freeing a log file with status U-B or F or Free a log file with status U-B-L.

Freeing a log file with status U-B or F

If a log file is backed up but still in use (status U-B), some transactions in the log file are still under way, or the log file contains the oldest update that is required for fast recovery. Because a log file with status F has been used in the past, it follows the same rules as for status U-B.

To free a backed up log file that is in use:

1. If you do not want to wait until the transactions complete, take the database server to quiescent mode. See Changing database server operating modes. Any active transactions are rolled back.
2. Use the **onmode -c** command to force a checkpoint. Do this because a log file with status U-B might contain the oldest update.

A log file that is backed up but not in use (status U-B) is not required to be freed. In the following example, log 34 is not required to be freed, but logs 35 and 36 do. Log 35 contains the last checkpoint, and log 36 is backed up but still in use.

```
34 U-B--   Log is used, backed up, and not in use
35 U-B-L   Log is used, backed up, contains last checkpoint
36 U-B--   Log is used, backed up, and not in use
37 U-C--   This is the current log file, not backed up
```

Tip: You can free a logical log with a status of U-B (and not L) only if it is not spanned by an active transaction and does not contain the oldest update.

Freeing a log file with status U-C or U-C-L

Follow these steps to free the current log file.

To free the current log file (status C):

1. Run the following command to switch the current log file to the next available log file: **onmode -l**
2. Back up the original log file with ON-Bar or **ontape**.
3. After all full log files are backed up, you are prompted to switch to the next available log file and back up the new current log file.
   You are not required to do the backup because you just switched to this log file.

After you free the current log file, if the log file has status U-B or U-B-L, see Freeing a log file with status U-B or F or Free a log file with status U-B-L.

Free a log file with status U-B-L

If a log file is backed up and all transactions within it are closed but the file is not free (status U-B-L), this logical-log file contains the most-recent checkpoint record. You can free log files with a status U-B-L.

To free log files with a status U-B-L, the database server must create a new checkpoint. You can run the following command to force a checkpoint: **onmode -c**

Monitor logging activity

Monitor the logical-log files to determine the total available space (in all the files), the space available in the current file, and the status of a file (for example, whether the log has been backed up yet). For information about monitoring the logical-log buffers, see Monitor physical and logical-logging activity.

- Monitor the logical log for fullness
- Monitor temporary logical logs
- SMI tables
- Monitor log-backup status

Monitor the logical log for fullness

You can use the following command-line utilities to monitor logical-log files.

- The onstat -l command
- The oncheck -pr command

The onstat -l command

The **onstat -l** command displays information about physical and logical logs.

The output section that contains information about each logical-log file includes the following information:

- The address of the logical-log file descriptor
- The log file number
- Status flags that indicate the status of each log (free, backed up, current, and so on)
- The unique ID of the log file
- The beginning page of the file
- The size of the file in pages, the number of pages used, and the percentage of pages used

The log file numbers in the **numbers** field can get out of sequence if you drop several logs in the middle of the list or if the database server dynamically adds log files.

For more information about and an example of **onstat -l** output, see the *IBM® Informix® Administrator's Reference*.

The oncheck -pr command

The database server stores logical-log file information in the reserved pages dedicated to checkpoint information. Because the database server updates this information only during a checkpoint, it is not as recent as the information that the **onstat -l** option displays. For more details on using these options to display reserved page information, see the *IBM® Informix® Administrator's Reference*.

You can view the checkpoint reserved pages with the **oncheck -pr** command. The following example shows sample output for one of the logical-log files.

```
...
Log file number             1
Unique identifier           7
Log contains last checkpoint Page 0, byte 272
Log file flags              0x3  Log file in use
                                 Current log file
Physical location           0x1004ef
Log size                    750 (p)
Number pages used           1
Date/Time file filled       01/29/2001 14:48:32
...
```


Monitor temporary logical logs

The database server uses *temporary logical logs* to roll forward transactions during a warm restore, because the permanent logs are not available then. When the rollforward completes, the database server frees the temporary log files. If you issue **onstat -l** during a warm restore, the output includes a fourth section on temporary log files in the same format as regular log files. Temporary log files use only the B, C, F, and U status flags.

SMI tables

Query the **syslogs** table to obtain information about logical-log files. This table contains a row for each logical-log file. The columns are as follows.

number
Identification number of the logical-log file
uniqid
Unique ID of the log file
size
Size of the file in pages
used
Number of pages used
is_used
Flag that indicates whether the log file is being used
is_current
Flag that indicates whether the log file is current
is_backed_up
Flag that indicates whether the log file has been backed up
is_new
Flag that indicates whether the log file has been added since the last storage space backup
is_archived
Flag that indicates whether the log file has been written to the archive tape
is_temp
Flag that indicates whether the log file is flagged as a temporary log file

Monitor log-backup status

To monitor the status of the logs and to see which logs have been backed up, use the **onstat -l** command. A status flag of B indicates that the log has been backed up.

Allocate logical log files

When you initialize or restart the database server, it creates the number of logical-log files that are specified by the LOGFILES configuration parameter. The size of the logical log files is specified by the LOGSIZE configuration parameter.

You can manually add logical log files or configure the database server to add logical log files as needed. The database server updates the value of the LOGFILES configuration parameter dynamically when logical log files are added.

The following configuration parameters also affect logical log files. You can update the value of these configuration parameters while the server is running, unless otherwise noted.

AUTO_LLOG
: Automatically adds logical logs to improve performance and limits the total size of logical log files.

DYNAMIC_LOGS
: Automatically adds logical logs to prevent transaction blocking.

LOGBUFF
: Sets the size of the three logical-log buffers in shared memory. You must restart the database server when you change the value of the LOGBUFF configuration parameter.

LTXEHWM
: Sets the percentage of available log space that, when filled, triggers the database server to give the long transaction currently that is being rolled back exclusive access to the logical log.

LTXHWM
: Sets the percentage of available log space that, when filled, triggers the database server to check for a long transaction.

- Dynamically add a logical-log file to prevent transaction blocking
  The DYNAMIC_LOGS configuration parameter determines when the database server dynamically adds a logical-log file to prevent transaction blocking.
- Dynamically add logical logs for performance
  You can set the AUTO_LLOG configuration parameter to enable the database server to dynamically add logical logs to improve performance.
- Adding logical-log files manually
  You can use an **onparams** command to add logical-log files.

Related information:
LOGFILES configuration parameter
LOGSIZE configuration parameter
AUTO_LLOG configuration parameter
DYNAMIC_LOGS configuration parameter
LOGBUFF configuration parameter
LTXEHWM configuration parameter
LTXHWM configuration parameter

Dynamically add a logical-log file to prevent transaction blocking

The DYNAMIC_LOGS configuration parameter determines when the database server dynamically adds a logical-log file to prevent transaction blocking.

When you use the default value of 2 for DYNAMIC_LOGS, the database server dynamically adds a new log file and sets off an alarm if the next active log file contains the beginning of the oldest open transaction.

The database server checks the logical-log space at these points:

- After switching to a new log file
- At the beginning of the transaction-cleanup phase of logical recovery

If the DYNAMIC_LOGS parameter is set to 1 and the next active log file contains records from an open transaction, the database server prompts you to add a log file manually and sets off an alarm. After you add the log file, the database server resumes processing the transaction.

If the DYNAMIC_LOGS parameter is set to 0 and the logical log runs out of space during a long transaction rollback, the database server can hang. (The long transaction prevents the first logical-log file from becoming free and available for reuse.) To fix the problem and complete the long transaction, set DYNAMIC_LOGS to 2 and restart the database server.

- Size and number of dynamically added log files
  The purpose of enabling dynamic logs with the DYNAMIC_LOGS configuration parameter is to add enough log space to allow transactions to roll back.
- Location of dynamically added logical log files
  If the DYNAMIC_LOGS configuration parameter is set to 2, the default location of dynamically added log files is the dbspace that contains the newest log file.
- Monitor events for dynamically added logs
  You can monitor the event alarms that are triggered when the database server dynamically adds logical log files to prevent transaction blocking. The DYNAMIC_LOGS configuration parameter value must be 1 or 2.

Related reference:
Monitor events for dynamically added logs
Controlling long transactions
Related information:
DYNAMIC_LOGS configuration parameter

Size and number of dynamically added log files

The purpose of enabling dynamic logs with the DYNAMIC_LOGS configuration parameter is to add enough log space to allow transactions to roll back.

When dynamically adding a log file, the database server uses the following factors to calculate the size of the log file:

- Average log size
- Amount of contiguous space available

If the logical log is low on space, the database server adds as many log files as necessary to allow the transaction to roll back. The number of log files is limited by:

- The maximum number of log files supported

- The amount of disk space for the log files
- The amount of free contiguous space in the root chunk

If the database server stops adding new log files because it is out of disk space, it writes an error message and sets off an alarm. Add a dbspace or chunk to an existing dbspace. Then the database server automatically resumes processing the transaction.

The reserve pages in the root chunk store information about each log file. The extents that contain this information expand as more log files are added. The root chunk requires two extents of 1.4 megabytes each to track 32,767 log files, the maximum number supported.

If during reversion, the chunk reserve page extent is allocated from a non-root chunk, the server attempts to put it back in the root chunk. If not enough space is available in the root chunk, the reversion fails. A message containing the required space displays in the online log. The required space must be freed from the root chunk before trying the reversion again.

Location of dynamically added logical log files

If the DYNAMIC_LOGS configuration parameter is set to 2, the default location of dynamically added log files is the dbspace that contains the newest log file.

The database server allocates log files in dbspaces, in the following search order. A dbspace becomes critical if it contains logical-log files or the physical log.

Pass	Allocate log file in
1	The dbspace that contains the newest log files (If this dbspace is full, the database server searches other dbspaces.)
2	Mirrored dbspace that contains log files (but excluding the root dbspace)
3	All dbspaces that already contain log files (excluding the root dbspace)
4	The dbspace that contains the physical log
5	The root dbspace
6	Any mirrored dbspace
7	Any dbspace

If you do not want to use this search order to allocate the new log file, you must set the DYNAMIC_LOGS parameter to 1 and run **onparams -a -i** with the location you want to use for the new log. For details, see Monitor events for dynamically added logs.

Monitor events for dynamically added logs

You can monitor the event alarms that are triggered when the database server dynamically adds logical log files to prevent transaction blocking. The DYNAMIC_LOGS configuration parameter value must be 1 or 2.

When each alarm is triggered, a message is written to the message log.

You can include the **onparams** command to add log files in your alarm script for event class ID 27, log file required. Your script can also run the **onstat -d** command to check for adequate space and run the **onparams a -i** command with the location that has enough space. You must use the **-i** option to add the new log right after the current log file.

Table 1. Event alarms for dynamically added log files

Class ID	Severity	Class message	Message
26	3	Dynamically added log file *log_number*	This message shows when the database server dynamically adds a log file. Dynamically added log file *log_number* to DBspace *dbspace_number*.
27	4	Log file required	This message shows when DYNAMIC_LOGS is set to 1 and the database server is waiting for you to add a log file. ALERT: The oldest logical log *log_number* contains records from an open transaction *transaction_address*. Logical logging remains blocked until a log file is added. Add the log file with the **onparams -a** command, using the **-i** (insert) option, as in: `onparams -a -d dbspace -s size -i` Then complete the transaction as soon as possible.
28	4	No space for log file	ALERT: Because the oldest logical log *log_number* contains records from an open transaction *transaction_address*, the server is attempting to dynamically add a log file. But there is no space available. Add a dbspace or chunk, then complete the transaction as soon as possible.

The following table shows the actions that the database server takes for each setting of the DYNAMIC_LOGS configuration parameter.

Table 2. The DYNAMIC_LOGS settings

DYNAMIC_ LOGS value	Meaning	Event alarm	Wait to add log?	Dynamic log added?
2 (default)	Allows automatic allocation of new log files to prevent open transactions from hanging the system.	Yes (26, 28)	No	Yes
1	Allows manual addition of new log files.	Yes (27)	Yes	No
0	Does not allocate log files but issues the following message about open transactions: Warning: The oldest logical-log file *log_number* contains records from an open transaction *transaction_address*, but the dynamic log feature is turned off.	No	No	No

Related reference:
Dynamically add a logical-log file to prevent transaction blocking
Related information:
Event Alarms
DYNAMIC_LOGS configuration parameter

Dynamically add logical logs for performance

You can set the AUTO_LLOG configuration parameter to enable the database server to dynamically add logical logs to improve performance.

When you set the AUTO_LLOG configuration parameter, you also specify a dbspace in which to create new logical log files and the size of all logical log files at which the server stops adding logs for performance.

The AUTO_LLOG and DYNAMIC_LOGS configuration parameters add logical logs under different conditions that do not directly interact. When the AUTO_LLOG configuration parameter is enabled, logical logs are added to improve performance. When the DYNAMIC_LOGS configuration parameter is enabled, logical logs are added under more urgent conditions, such as when a long transaction threatens to block the server by using all available log space. The settings of the two configuration parameters do not constrain each other. For example, the maximum size that is specified in the AUTO_LLOG configuration parameter does not affect the amount of log space that can be added by the DYNAMIC_LOGS configuration parameter. Similarly, the value of AUTO_LLOG configuration parameter does not affect the amount of log space that you can add manually.

Related information:
AUTO_LLOG configuration parameter

Adding logical-log files manually

You can use an **onparams** command to add logical-log files.

You might add logical-log files manually for the following reasons:

- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- To enable an open transaction to roll back
- As part of moving logical-log files to a different dbspace

Restriction: You cannot do the following actions:

- Add a log file to a blobspace or sbspace.
- Add logical or physical logs to dbspaces that have non-default page sizes.

Add logical-log files one at a time, up to a maximum of 32,767 files, to any dbspace. As soon as you add a log file to a dbspace, it becomes a critical dbspace. You can add a logical-log file during a storage space backup.

You can add a logical-log file in either of the following locations:

- At the end of the file list using the **onparams -a** command
- After the current logical-log file using the **onparams -a -i** command

To add a logical-log file using **onparams**

1. Log-in as user **informix** or **root** on UNIX or as a member of the **Informix-Admin** group on Windows.
2. Ensure that the database server is in online, administration, or quiescent, or cleanup phase of fast-recovery mode.
   The database server writes the following message to the log during the cleanup phase:

   ```
   Logical recovery has reached the transaction cleanup phase.
   ```

3. Decide whether you want to add the log file to the end of the log file list or after the current log file.
   You can insert a log file after the current log file regardless of the DYNAMIC_LOGS parameter value. Adding a log file of a new size does not change the value of LOGSIZE.

   - The following command adds a logical-log file to the end of the log file list in the **logspace** dbspace, using the log-file size specified by the LOGSIZE configuration parameter:

     ```
     onparams -a -d logspace
     ```

- The following command inserts a 1000 KB logical-log file after the current log file in the **logspace** dbspace:

  ```
  onparams -a -d logspace -s 1000 -i
  ```

- To add a logical-log file with a new size (in this case, 250 KB), run the following command:

  ```
  onparams -a -d logspace -s 250
  ```

4. Use **onstat -l** to check the status of the log files. The status of the new log file is **A** and is immediately available.
5. The next time you must back up data, perform a level-0 backup of the root dbspace and the dbspaces that contain the new log files.
   Although you are no longer required to back up immediately after adding a log file, your next backup must be level-0 because the data structures have changed. For information about backing up data, see the *IBM® Informix® Backup and Restore Guide*.

For more information about using **onparams** to add a logical-log file, see the *IBM Informix Administrator's Reference*.

Related concepts:
Move logical-log files
Related information:
onstat -g iov command: Print AIO VP statistics

Dropping logical-log files

You can use an **onparams** command to drop logical-log files.

To drop a logical-log file and increase the amount of the disk space available within a dbspace, you can use **onparams**. The database server requires a minimum of three logical-log files at all times. You cannot drop a log if your logical log is composed of only three log files.

The rules for dropping log files have changed:

- If you drop a log file that has never been written to (status **A**), the database server deletes it and frees the space immediately.
- If you drop a used log file (status **U-B**), the database server marks it as deleted (**D**). After you take a level-0 backup of the dbspaces that contain the log files and the root dbspace, the database server deletes the log file and frees the space.
- You cannot drop a log file that is currently in use or contains the last checkpoint record (status **C** or **L**).

To drop a logical-log file with **onparams**:

1. Ensure that the database server is in online, administration, or quiescent mode.
2. Run the following command to drop a logical-log file whose log file number is 21: **onparams -d -l 21**
   Drop log files one at a time. You must know the log file number of each logical log that you intend to drop.
3. If the log file has a status of newly Added (**A**), it is dropped immediately.
   If the log file has a status of Used (**U**), it is marked as Deleted (**D**).
4. To drop a used log file, take a level-0 backup of all the dbspaces.
   This backup prevents the database server from using the dropped log files during a restore and ensures that the reserved pages contain information about the current number of log files.

For information about using **onparams** to drop a logical-log file, see the *IBM® Informix® Administrator's Reference*.

For information about using **onlog** to display the logical-log files and unique ID numbers, see Display logical-log records.

Related concepts:
Move logical-log files
Related information:
onparams -d -l lognum: Drop a logical-log file
The onlog utility

Change the size of logical-log files

If you want to change the size of the log files, it is easier to add new log files of the appropriate size and then drop the old ones. You can change the size of logical-log files in the following ways:

- Use **onparams** with the **-s** option to add a new log file of a different size.
  See Adding logical-log files manually.
- Increase the LOGSIZE value in the onconfig file if you want the database server to create larger log files.

Move logical-log files

You might want to move logical-log files for performance reasons or to make more space in the dbspace.

To find the location of logical-log files, run the **onstat -l** command. Although moving the logical-log files is not difficult, it can be time-consuming.

Moving logical-log files is a combination of two simpler actions:

- Optionally dropping logical-log files from their current dbspace.
- Adding the logical-log files to their new dbspace

Restriction: You cannot move logical log files into dbspaces that have non-default page sizes.
The database server must be in online, administration, quiescent, or fast-recovery mode.

You can change the location for new logical logs by setting the AUTO_LLOG configuration parameter to 1 and the name of the dbspace. The AUTO_LLOG configuration parameter enables the database server to add logical logs as needed to improve performance.

Example

The following procedure provides an example of how to move six logical-log files from the root dbspace to another dbspace, **dbspace_1**:

1. Add six new logical-log files to **dbspace_1** by running the following command:

   ```
   onparams -a -d dbspace_1
   ```

2. Take a level-0 backup of all storage spaces to free all log files except the current log file. For example, you can run the following command to back up all log files, including the current log file:

   ```
   onbar -l -b -c
   ```

3. Run the **onmode -l** command to switch to a new current log file.
4. Drop all six logical-log files in the root dbspace by running the **onparams -d -l** command with the log file number for each log file. You cannot drop the current logical-log file.
5. Create a level-0 backup of the root dbspace and **dbspace_1**. For example, you can run the following command:

   ```
   onbar -b root dbspace_1
   ```

Related concepts:
Size of the root dbspace
Related tasks:
Dropping logical-log files
Adding logical-log files manually
Related reference:
Location of logical-log files
Related information:
AUTO_LLOG configuration parameter
onstat -l command: Print physical and logical log information
onbar -b syntax: Backing up

Display logical-log records

Use the **onlog** utility to display and interpret logical-log records. For information about using **onlog**, see the *IBM® Informix® Administrator's Reference*.

Controlling long transactions

There are multiple ways to control long transactions:

- Adjust the long-transaction high-watermark setting
- Adjust the exclusive access, long-transaction high-watermark setting
- Adjust the size of log files
- Set limits on logspace available to individual transactions
- Set limits on the amount of time a transaction can run

The database server uses the LTXHWM and LTXEHWM configuration parameters to set high-watermarks for long transactions. If DYNAMIC_LOGS is set to `1` or `2`, the default LTXHWM value is 80 percent and LTXEHWM is 90 percent. If DYNAMIC_LOGS is set to `0`, the default LTXHWM value is 50 percent and the default LTXHEWM value is 60 percent.

If you decrease your high-watermark values, you increase the likelihood of long transactions. To compensate, allocate additional log space.

Long-transaction high-watermark (LTXHWM)

The *long-transaction high-watermark* is the percentage of total log space that a transaction is allowed to span before it is rolled back.

If the database server finds an open transaction in the oldest used log file, it dynamically adds log files. Because the log space is increasing, the high-watermark expands outward. When the log space reaches the high-watermark, the database server rolls back the transaction. The transaction rollback and other processes also generate logical-log records. The database server continues adding log files until the rollback is complete to prevent the logical log from running out of space. More than one transaction can be rolled back if more than one long transaction exists.

For example, the database server has 10 logical logs and LTXHWM is set to 98. A transaction begins in log file 1 and update activity fills logs 1 - 9. The database server dynamically adds log file 11 after log file 10. If the transaction does not complete, this process continues until the database server adds 40 log files. When the database server adds the 50th log, the transaction is caught up to the high-watermark and the database server rolls it back.

Exclusive access, long-transaction high-watermark (LTXEHWM)

The *exclusive-access, long-transaction high-watermark* occurs when the long transaction currently being rolled back is given exclusive access to the logical log. The database server dramatically reduces log-record generation. Only threads that are currently rolling back transactions and threads that are currently writing COMMIT records are allowed access to the logical log. Restricting access to the logical log preserves as much space as possible for rollback records that are being written by the user threads that are rolling back transactions.
Important: If you set both LTXHWM and LTXEHWM to 100, long transactions are never stopped by the database server. Therefore, you must set LTXHWM to below 100 for normal database server operations. Set LTXHWM to 100 to run scheduled transactions of unknown length. Set LTXEHWM to 100 if you have enough disk space, and you never want to block other users while a long transaction is rolling back.

Adjust the size of log files to prevent long transactions

Use larger log files when many users are writing to the logs at the same time. If you use small logs and long transactions are likely to occur, reduce the high-watermark. Set the LTXHWM value to `50` and the LTXEHWM value to `60`.

If the log files are too small, the database server might run out of log space while rolling back a long transaction. In this case, the database server cannot block fast enough to add a log file before the last one fills. If the last log file fills, the system hangs and displays an error message. To fix the problem, shut down and restart the database server.

Set limits on logspace available to transactions

The SESSION_LIMIT_LOGSPACE configuration parameter limits how much log space a session can use for each transaction, which prevents transactions above a specific size from occurring, and prevents large transactions from a single session from monopolizing system resources.

The database server terminates a transaction that exceeds the log space limit, and produces an error in the database server message log.

The size limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Set limits on the amount of time a transaction can run

The SESSION_LIMIT_TXN_TIME configuration parameter limits how much time a transaction can run in a session, which prevents transactions that require a large amount of time from occurring, and prevents long transactions from a single session from monopolizing system resources.

The database server terminates a transaction that exceeds the time limit, and produces an error in the database server message log.

The time limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Related reference:
Dynamically add a logical-log file to prevent transaction blocking
Related information:
SESSION_LIMIT_LOGSPACE configuration parameter
SESSION_LIMIT_TXN_TIME configuration parameter

Physical logging, checkpoints, and fast recovery

These topics cover the three procedures that the database server uses to achieve data consistency:

- Physical logging
- Checkpoints
- Fast recovery

The *physical log* is a set of disk pages where the database server stores an unmodified copy of the page called a *before-image*. *Physical logging* is the process of storing a before-image of a page that the database server is going to change. A *checkpoint* is a point when the database server synchronizes the pages on disk with the pages in the shared-memory buffers. *Fast recovery* is an automatic procedure that restores the database server to a consistent state after it goes offline under uncontrolled conditions.

These procedures ensure that multiple, logically related writes are recorded as a unit, and that data in shared memory is periodically made consistent with data on disk.

For the tasks to manage and monitor the physical log and checkpoints, see Manage the physical log.

- Critical sections
- Physical logging
- Size and location of the physical log
  When the database server initializes disk space, it places the physical log in the root dbspace. The initial size of the physical log is set by the PHYSFILE configuration parameter.
- Checkpoints
- Fast recovery

Related reference:
Database server maintenance tasks

Critical sections

A *critical section* is a section of code (or machine instructions) that must be performed as a single unit. A critical section ensures the integrity of a thread by allowing it to run a series of instructions before it is swapped out.

Physical logging

Physical logging is the process of storing the pages that the database server is going to change before the changed pages are actually recorded on disk. Before the database server modifies certain pages in the shared-memory buffer pool, it stores before-images of the pages in the physical-log buffer in shared memory.

The database server maintains the before-image page in the physical-log buffer in shared memory for those pages until one or more page cleaners flush the pages to disk. The unmodified pages are available in case the database server fails or the backup procedure requires them to provide an accurate snapshot of the database server data. Fast recovery and database server backups use these snapshots.

The database server recycles the physical log at each checkpoint, except in the special circumstances. For more information about checkpoints, see Checkpoints.

- Fast recovery use of physically-logged pages
- Backup use of physically-logged pages
- Database server activity that is physically logged

Fast recovery use of physically-logged pages

After a failure, the database server uses the before-images of pages to restore these pages on the disk to their state at the last checkpoint. Then the database server uses the logical-log records to return all data to physical and logical consistency, up to the point of the most-recently completed transaction. Fast recovery explains this procedure in more detail.

Backup use of physically-logged pages

When you perform a backup, the database server performs a checkpoint and coordinates with the physical log to identify the correct version of pages that belong on the backup. In a level-0 backup, the database server backs up all disk pages. For more details, see the *IBM® Informix® Backup and Restore Guide*.

Database server activity that is physically logged

If multiple modifications were made to a page between checkpoints, typically only the first before-image is logged in the physical log.

The physical log is a cyclical log in which the pages within the physical log are used once per checkpoint. If the RTO_SERVER_RESTART configuration parameter is set, additional physical logging occurs to improve fast recovery performance.

- Physical recovery messages
- Physical logging and simple large objects
- Physical logging and smart large objects

Physical recovery messages

When fast recovery begins, the database server logs the following message with the name of the chunk and offset:

```
Physical recovery started at page chunk:offset.
```

When the fast recovery completes, the database server logs the following message with the number of pages examined and restored:

```
Physical recovery complete: number pages examined, number pages restored.
```

Physical logging and simple large objects

The database server pages in the physical log can be any database server page, including simple large objects in table spaces (tblspaces). Even overhead pages (such as chunk free-list pages, blobspace free-map pages, and blobspace bitmap pages) are copied to the physical log before data on the page is modified and flushed to disk.

Blobspace blobpages are not logged in the physical log. For more information about blobspace logging, see Log blobspaces and simple large objects.

Physical logging and smart large objects

The user-data portion of smart large objects is not physically logged. However, the metadata is physically logged. For information about smart large objects, see Sbspaces.

Size and location of the physical log

When the database server initializes disk space, it places the physical log in the root dbspace. The initial size of the physical log is set by the PHYSFILE configuration parameter.

After you initialize the database server for the first time, you can change the size or location of the physical log with the **onparams** utility.

To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize disk contention), you can move the physical log out of the root dbspace to another dbspace, preferably to a disk that does not contain active tables or the logical-log files. For best performance, create the plogspace to store the physical log and allow the database server to expand the size of the physical log as needed to improve performance.

Recommendation: Locate critical dbspaces on fault-tolerant storage devices. If the storage that the physical log is in is not fault-tolerant, use IBM® Informix® mirroring for the dbspace that contain the physical log. This protects the database if the storage device fails. However, if you mirror the plogspace, it cannot be expanded.

- Strategy for estimating the size of the physical log
  The size of the physical log depends on two factors: the rate at which transactions generate physical log activity and whether you set the RTO_SERVER_RESTART configuration parameter
- Physical-log overflow when transaction logging is turned off

Related concepts:
Plogspace
Related reference:
Change the physical-log location and size

Strategy for estimating the size of the physical log

The size of the physical log depends on two factors: the rate at which transactions generate physical log activity and whether you set the RTO_SERVER_RESTART configuration parameter

The rate at which transactions generate physical log activity can affect checkpoint performance. During checkpoint processing, if the physical log starts getting too full as transactions continue to generate physical log data, the database server blocks transactions to allow the checkpoint to complete and to avoid a physical log overflow.

To avoid transaction blocking, the database server must have enough physical log space to contain all of the transaction activity that occurs during checkpoint processing. Checkpoints are triggered whenever the physical log becomes 75 percent full. When the physical log becomes 75 percent full, checkpoint processing must complete before the remaining 25 percent of the physical log is used. Transaction blocking occurs as soon as the system detects a potential for a physical log overflow, because every active transaction might generate physical log activity.

For example, suppose you have a one gigabyte physical log and 1000 active transactions. 1000 active transactions have the potential to generate approximately 80 megabytes of physical log activity if every transaction is in a critical section simultaneously. When 750 megabytes of the physical log fills, the database server triggers a checkpoint. If the checkpoint has not completed by the time the 920 megabytes of the physical log are used, transaction blocking occurs until the checkpoint completes. If transaction blocking takes place, the server automatically triggers more frequent checkpoints to avoid transaction blocking. You can disable the generation of automatic checkpoints.

The server might also trigger checkpoints if many dirty partitions exist, even if the physical log is not 75 percent full, because flushing the modified partition data to disk requires physical log space. When the server checks if the Physical Log is 75 percent full, the server also checks if the following condition is true:

```
(Physical Log Pages Used + Number of Dirty Partitions) >=
(Physical Log Size * 9) /10)
```

For more information about checkpoint processing and automatic checkpoints, see Checkpoints.

The second factor to consider when estimating the size of the physical log depends on your use of the RTO_SERVER_RESTART configuration parameter to specify a target amount of time for fast recovery. If you are not required to consider fast recovery time, you are not requires to enable the RTO_SERVER_RESTART configuration parameter. If you specify a value for the RTO_SERVER_RESTART configuration parameter, transaction activity generates additional physical log activity.

Typically, this additional physical log activity has little or no affect on transaction performance. The extra logging is used to assist the buffer pool during fast recovery, so that log replay performs optimally. If the physical log is considerably larger than the combined sizes of all buffer pools, page flushing and page faulting occur during fast recovery. The page flushing and page faulting substantially reduce fast recovery performance, and the database server cannot maintain the RTO_SERVER_RESTART policy.

For systems with less that four gigabytes of buffer pool space, the physical log can be sized at 110 percent of the combined size of all the buffer pools. For larger buffer pools, start with four gigabytes of physical log space and then monitor checkpoint activity. If checkpoints occur too frequently and seem to affect performance, increase the physical log size.

A rare condition, called a physical-log overflow, can occur when the database server is configured with a small physical log and has many users. Following the previously described size guidelines helps avoid physical-log overflow. The database server generates performance warnings to the message log whenever it detects suboptimal configurations.

You can use the **onstat -g ckp** command to display configuration recommendations if a suboptimal configuration is detected.

Physical-log overflow when transaction logging is turned off

The physical log can overflow if you use simple large objects or smart large objects in a database with transaction logging turned off, as the following example shows.

When the database server processes simple large objects, each portion of the simple large object that the database server stores on disk can be logged separately, allowing the thread to exit the critical sections of code between each portion. However, if logging is turned off, the database server must carry out all operations on the simple large object in one critical section. If the simple large object is large and the physical log small, this scenario can cause the physical logs to become full. If this situation occurs, the database server sends the following message to the message log:

```
Physical log file overflow
```

The database server then initiates a shutdown. For the suggested corrective action, see your message log.

Checkpoints

Periodically, the database server flushes transactions and data within the buffer pool to disk. Until the transactions and data are flushed to disk, the data and transactions are in a state of flux. Instead of forcing every transaction to disk immediately after a transaction is completed, the database server writes transactions to the logical log. The database server logs the transactions as they occur. In the event of a system failure, the server:

- Replays the log to redo and restore the transactions.
- Returns the database to a state consistent with the state of the database system at the time of the failure.

To facilitate the restoration or logical recovery of a database system, the database server generates a consistency point, called a *checkpoint*. A checkpoint is a point in time in the log when a known and consistent state for the database system is established. Typically, a checkpoint involves recording a certain amount of information so that, if a failure occurs, the database server can restart at that established point.

The purpose of a checkpoint is to periodically move the restart point forward in the logical log. If checkpoints did not exist and a failure occurred, the database server would be required to process all the transactions that were recorded in the logical log since the system restarted.

A checkpoint can occur in one of these situations:

- When specific events occur. For example, a checkpoint occurs whenever a dbspace is added to the server or a database backup is performed.
  Typically, these types of events trigger checkpoints that block transaction processing. Therefore, these checkpoints are called *blocking* checkpoints.
- When resource limitations occur. For example, a checkpoint is required for each span of the logical log space to guarantee that the log has a checkpoint at which to begin fast recovery. The database server triggers a checkpoint when the physical log is 75 percent full to avoid physical log overflow.
  Checkpoints triggered by resource limitations usually do not block transactions. Therefore, these checkpoints are called *nonblocking checkpoints*.

  However, if the database server begins to run out of resources during checkpoint processing, transaction blocking occurs in the midst of checkpoint processing to make sure that the checkpoint completes before a resource is depleted. If transactions are blocked, the server attempts to trigger checkpoints more frequently to avoid transaction blocking during checkpoint processing. For more information, see Strategy for estimating the size of the physical log.

If failover occurs, and the secondary server becomes the primary server, checkpoint discrepancies between the two servers can affect re-connection attempts. If a checkpoint on the new secondary server does not exist on the new primary server, attempts to connect the secondary server to the primary server fail. The secondary server must be fully restored before it can connect to the primary server.

Automatic checkpoints cause the database server to trigger more frequent checkpoints to avoid transaction blocking. Automatic checkpoints attempt to monitor system activity and resource usage (physical and logical log usage along with how dirty the buffer pools are) to trigger checkpoints in a timely manner so that the processing of the checkpoint can complete before the physical or logical log is depleted. The database server generates at least one automatic checkpoint for each span of the logical-log space. This guarantees the existence of a checkpoint where fast recovery can begin. Use the AUTO_CKPTS configuration parameter to enable or disable automatic checkpoints when the database server starts. (You can dynamically enable or disable automatic checkpoints by using **onmode -wm** or **onmode -wf**.)
Tip:

- If automatic checkpoints are triggered too frequently because of physical log activity, you can increase the physical log size or use a plogspace to automatically tune the physical log resources.
- If automatic checkpoints are triggered too frequently because of logical log activity, you can set the AUTO_LLOG parameter in the onconfig file to allow the server to automatically increase the logical log space to reduce checkpoint frequency.

Manual checkpoints are event-based checkpoints that you can initiate. The database server provides two methods for determining how long fast recovery takes in the event of an unplanned outage.

- Use the CKPTINTVL configuration parameter to specify how frequently the server triggers checkpoints.
- Use the RTO_SERVER_RESTART configuration parameter to specify how much time fast recovery takes.
  When you use the RTO_SERVER_RESTART configuration parameter:
  - The database server ignores the CKPTINTVL configuration parameter.
  - The database server monitors the physical and logical log usage to estimate the duration of fast recovery. If the server estimates that fast recovery exceeds the time specified in the RTO_SERVER_RESTART configuration parameter, the server automatically triggers a checkpoint.

The RTO_SERVER_RESTART configuration parameter is intended to be a target amount of time, not a guaranteed amount of time. Several factors that can increase restart time can also influence fast recovery time. These factors include rolling back long transactions that were active at the time of an unplanned outage. For more information about the RTO_SERVER_RESTART and AUTO_CKPTS configuration parameters, see the topics on configuration parameters in the *IBM® Informix® Administrator's Reference*.

- LRU values for flushing a buffer pool between checkpoints
- Checkpoints during backup

LRU values for flushing a buffer pool between checkpoints

The LRU values for flushing a buffer pool between checkpoints are not critical for checkpoint performance. The **lru_max_dirty** and **lru_min_dirty** values, which are set in the BUFFERPOOL configuration parameter, are usually necessary only for maintaining enough clean pages for page replacement. Start by setting **lru_min_dirty** to `70` and **lru_max_dirty** to `80`.

If transactions are blocked during a checkpoint, the database server subsequently attempts to increase checkpoint frequency to eliminate the transaction being blocked. When the server searches for a free page to perform page replacement and a foreground write occurs, the server subsequently automatically increases the LRU flushing frequency to prevent this event from occurring again. When the database server completes page replacement and finds a frequently accessed page, the server automatically increases LRU flushing. Any automatic adjustments to LRU flushing do not persist to the onconfig file.

For more information about monitoring and tuning checkpoint parameters and information about LRU tuning and adjustments, see the *IBM® Informix® Performance Guide*.

Checkpoints during backup

If you perform a backup, the database server runs a checkpoint and flushes all changed pages to the disk. If you perform a restore, the database server reapplies all logical-log records.

For information about ON-Bar or **ontape**, see the *IBM® Informix® Backup and Restore Guide*.

Fast recovery

Fast recovery is an automatic, fault-tolerant feature that the database server executes every time that it moves from offline to quiescent, administration, or online mode. You are not required to take any administrative actions for fast recovery; it is an automatic feature.

The fast-recovery process checks if, the last time that the database server went offline, it did so in uncontrolled conditions. If so, fast recovery returns the database server to a state of physical and logical consistency.

If the fast-recovery process finds that the database server came offline in a controlled manner, the fast-recovery process terminates, and the database server moves to online mode.

See Fast recovery after a checkpoint.

- Need for fast recovery
- Situations when fast recovery is initiated
- Fast recovery after a checkpoint

Need for fast recovery

Fast recovery restores the database server to physical and logical consistency after any failure that results in the loss of the contents of memory for the database server. For example, the operating system fails without warning. System failures do not damage the database but instead affect transactions that are in progress at the time of the failure.

Situations when fast recovery is initiated

Every time that the administrator brings the database server to quiescent, administration, or online mode from offline mode, the database server checks to see if fast recovery is required.

As part of shared-memory initialization, the database server checks the contents of the physical log. The physical log is empty when the database server shuts down under control. The move from online mode to quiescent mode includes a checkpoint, which flushes the physical log. Therefore, if the database server finds pages in the physical log, the database server clearly went offline under uncontrolled conditions, and fast recovery begins.

Fast recovery and buffered logging

If a database uses buffered logging (as described in Buffered transaction logging), some logical-log records associated with committed transactions might not be written to the logical log at the time of the failure. If this occurs, fast recovery cannot restore those transactions. Fast recovery can restore only transactions with an associated COMMIT record stored in the logical log on disk. (For this reason, buffered logging represents a trade-off between performance and data vulnerability.)

Possible physical log overflow during fast recovery

During fast recovery, the physical log can overflow. If this occurs, the database server tries to extend the physical log space to a disk file named plog_extend.*servernum*. The default location of this file is $INFORMIXDIR/tmp.

Use the ONCONFIG parameter PLOG_OVERFLOW_PATH to define the location for creating this file.

The database server removes the plog_extend.*servernum* file when the first checkpoint is performed during a fast recovery.

Fast recovery and no logging

For databases or tables that do not use logging, fast recovery restores the database to its state at the time of the most recent checkpoint. All changes made to the database since the last checkpoint are lost.

Fast recovery after a checkpoint

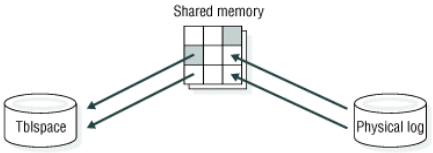
Fast recovery returns the database server to a consistent state as part of shared-memory initialization. All committed transactions are restored, and all uncommitted transactions are rolled back.

Fast recovery occurs in the following steps:

1. The database server uses the data in the physical log to return all disk pages to their condition at the time of the most recent checkpoint. This point is known as *physical consistency*.
2. The database server locates the most recent checkpoint record in the logical-log files.
3. The database server rolls forward all logical-log records written after the most recent checkpoint record.
4. The database server rolls back all uncommitted transactions. Some XA transactions might be unresolved until the XA resource manager is available.

The server returns to the last-checkpoint state

To return all disk pages to their condition at the time of the most recent checkpoint, the database server writes the before-images stored in the physical log to shared memory and then back to disk. Each before-image in the physical log contains the address of a page that was updated after the checkpoint. When the database server writes each before-image page in the physical log to shared memory and then back to disk, changes to the database server data since the time of the most recent checkpoint are undone. The database server is now physically consistent. The following figure illustrates this step.
Figure 1. Writing all remaining before-images in the physical log back to disk



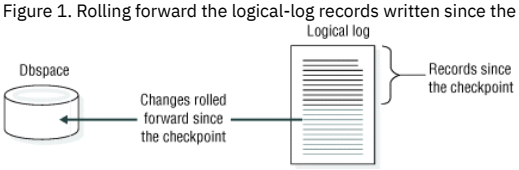
The server locates the checkpoint record in the logical log

After returning to the last checkpoint state, the database server locates the address of the most recent checkpoint record in the logical log. The most recent checkpoint record is guaranteed to be in the logical log on disk.

The server rolls forward logical-log records

After locating the checkpoint record in the logical log, the database server rolls forward the logical-log records that were written after the most recent checkpoint record. This action reproduces all changes to the databases since the time of the last checkpoint, up to the point at which the uncontrolled shutdown occurred. The following figure illustrates this step.

Figure 1. Rolling forward the logical-log records written since the most recent checkpoint



The server rolls back uncommitted transactions

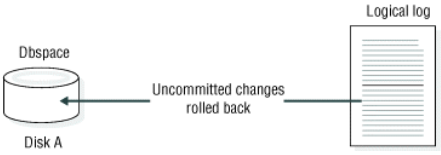
After rolling the logical-log records forward, the database server rolls back all logical-log records for transactions that were not committed at the time the system failed. All databases are logically consistent because all committed transactions are rolled forward and all uncommitted transactions are rolled back. Some XA transactions might be unresolved until the XA resource manager is available.

Transactions that have completed the first phase of a two-phase commit are exceptional cases. For more information, see How the two-phase commit protocol handles failures.

Because one or more transactions possibly spanned several checkpoints without being committed, this rollback procedure might read backward through the logical log, past the most recent checkpoint record. All logical-log files that contain records for open transactions are available to the database server because a log file is not freed until all transactions that it contains are closed.

The following figure illustrates the rollback procedure. Here, uncommitted changes are rolled back from the logical log to a dbspace on a particular disk. When fast recovery is complete, the database server returns to quiescent, administration, or online mode.

Figure 1. Rolling back all incomplete transactions



Manage the physical log

These topics describe the following procedures:

- Changing the location and size of the physical log
- Monitoring the physical log, physical-log buffers, and logical-log buffers
- Monitoring and forcing checkpoints

See Physical logging, checkpoints, and fast recovery for background information.

- Change the physical-log location and size
  You can use the **onparams** utility to change the location and size of the physical log.
- Monitor physical and logical-logging activity

- Monitor checkpoint information
  Monitor checkpoint activity to view information that includes the number of times that threads were required to wait for the checkpoint to complete. This information is useful for determining if the checkpoint interval is appropriate.
- Turn automatic LRU tuning on or off
  Use the AUTO_LRU_TUNING configuration parameter to enable or disable automatic LRU tuning when the database server starts.

Change the physical-log location and size

You can use the **onparams** utility to change the location and size of the physical log.

You can move the physical-log file to try to improve performance. When the database server initializes disk space, it places the disk pages that are allocated for the physical log in the root dbspace. You might improve performance by moving the physical log to another dbspace.

You can move the physical log to a dbspace or the plogspace. When the physical log is in the plogspace, the database server increases the size of the physical log as needed to improve performance. When the physical log is in a dbspace, you must manually increase the size of the physical log.

To move the physical log to the plogspace, create the plogspace by running the **onspaces -c -P** command or the SQL administration API **admin()** or **task()** function with the **create plogspace** argument. To change the location of the plogspace, create a new plogspace. The physical log is moved to the new plogspace and the old plogspace is dropped.

Prerequisites to moving the physical log to a dbspace:

- Log in as user **informix** or **root** on UNIX or as a member of the **Informix-Admin** group on Windows.
- Determine whether adequate contiguous space in the target chunk is available by running the **oncheck -pe** command.
  The space that is allocated for the physical log must be contiguous. When you change the physical log size or location, if the target dbspace does not contain adequate contiguous space, the server does not change the physical log. Additionally, if insufficient resources for the physical log exist when you initialize the database server, the initialization fails. The dbspace must use the default page size.

To move the physical log to a dbspace, run the **onparams -p -s** command or the SQL administration API **admin()** or **task()** function with the **create dbspace** argument.

The following example changes the size and location of the physical log. The new physical-log size is 400 KB, and the log is in the **dbspace6** dbspace:

```
onparams -p -s 400 -d dbspace6
```

Related concepts:
Size of the root dbspace
Related reference:
Monitor physical and logical-logging activity
Size and location of the physical log
The oncheck -pe command
Related information:
onparams -p: Change physical-log parameters
onspaces -c -P: Create a plogspace

Monitor physical and logical-logging activity

Monitor the physical log to determine the percentage of the physical-log file that gets used before a checkpoint occurs. You can use this information to find the optimal size of the physical-log file. It must be large enough that the database server is not required to force checkpoints too frequently and small enough to conserve disk space and guarantee fast recovery.

Monitor physical-log and logical-log buffers to determine if they are the optimal size for the current level of processing. The important statistic to monitor is the pages-per-disk-write statistic. For more information about tuning the physical-log and logical-log buffers, see your *IBM® Informix® Performance Guide*.

To monitor the physical-log file, physical-log buffers, and logical-log buffers, use the following commands.

Utility	Command	Additional information

Utility	Command	Additional information
Command line	**onstat -l**	The first line displays the following information for each physical-log buffer: • The number of buffer pages used (**bufused**) • The size of each physical log buffer in pages (**bufsize**) • The number of pages written to the buffer (**numpages**) • The number of writes from the buffer to disk (**numwrits**) • The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**) The second line displays the following information about the physical log: • The page number of the first page in the physical-log file (**phybegin**) • The size of the physical-log file in pages (**physize**) • The current position in the log where the next write occurs, specified as a page number (**physpos**) • The number of pages in the log that have been used (**phyused**) • The percentage of the total physical-log pages that have been used (**%used**) The third line displays the following information about each logical-log buffer: • The number of buffer pages used (**bufused**) • The size of each logical-log buffer in pages (**bufsize**) • The number of records written to the buffer (**numrecs**) • The number of pages written to the buffer (**numpages**) • The number of writes from the buffer to disk (**numwrits**) • The ratio of records to pages in the buffer (**recs/pages**) • The ratio of pages written to the buffer to the number of writes to disk (**pages/IO**)
Command line	**onparams -p**	Moves or resizes the physical log.
Command line	**onmode -l**	Advances to the next logical-log file.

For more information about and an example of **onstat -l** output, see the *IBM Informix Administrator's Reference*.

For information about using SQL administration API commands instead of some **onparams** and **onmode** commands, see Remote administration with the SQL administration API and the *IBM Informix Guide to SQL: Syntax*.

Related reference:
Change the physical-log location and size

Monitor checkpoint information

Monitor checkpoint activity to view information that includes the number of times that threads were required to wait for the checkpoint to complete. This information is useful for determining if the checkpoint interval is appropriate.

To monitor checkpoints, use the following commands.

Utility	Command	Additional Information
The **onstat** utility	**onstat -m**	View the last 20 lines in the message log. If a checkpoint message is not in the last 20 lines, read the message log directly with a text editor. The database server writes individual checkpoint messages to the log when the checkpoint ends. If a checkpoint occurs, but the database server has no pages to write to disk, the database server does not write any messages to the message log.
The **onstat** utility	**onstat -p**	Obtains these checkpoint statistics: • **numckpts**: Number of checkpoints that occurred since the database server was brought online. • **ckptwaits**: Number of times that a user thread waits for a checkpoint to finish. The database server prevents a user thread from entering a critical section during a checkpoint.

For information about tuning the checkpoint interval, see your *IBM® Informix® Performance Guide*.

- Turn checkpoint tuning on or off
- Force a checkpoint
  When necessary, you can force a checkpoint with an **onmode** or SQL administration API command.
- Server-provided checkpoint statistics
- SMI tables

Turn checkpoint tuning on or off

To turn automatic checkpoint tuning on, issue an **onmode –wf AUTO_CKPTS=1** command. To turn automatic checkpoint tuning off, issue an **onmode –wf AUTO_CKPTS=0** command.

Force a checkpoint

When necessary, you can force a checkpoint with an **onmode** or SQL administration API command.

Force a checkpoint in any of the following situations:

- To free a logical-log file that contains the most recent checkpoint record and that is backed up but not yet released (**onstat -l** status of `U-B-L` or `U-B`)
- Before you issue **onmode -sy** to place the database server in quiescent mode
- After building a large index, if the database server terminates before the next checkpoint. The index build restarts the next time that you restart the database server.
- If a checkpoint has not occurred for a long time and you are about to attempt a system operation that might interrupt the database server
- If foreground writes are taking more resources than you want (force a checkpoint to bring this down to zero temporarily)
- Before you run **dbexport** or unload a table, to ensure physical consistency of all data before you export or unload it
- After you perform a large load of tables using PUT or INSERT statements (Because table loads use the buffer cache, forcing a checkpoint cleans the buffer cache.)

To force a checkpoint, run **onmode -c**.

For information about using SQL administration API commands instead of some **onmode** commands, see Remote administration with the SQL administration API and the *IBM® Informix® Guide to SQL: Syntax*.

Server-provided checkpoint statistics

The database server provides history information about the previous twenty checkpoints. You can access this information through the SMI **sysckptinfo** table.

Related information:
sysckptinfo

SMI tables

Query the **sysprofile** table to obtain statistics on the physical-log and logical-log buffers. The **sysprofile** table also provides the same checkpoint statistics that are available from the **onstat -p** option. These rows contain the following statistics.

plgpagewrites
: Number of pages written to the physical-log buffer

plgwrites
: Number of writes from the physical-log buffer to the physical log file

llgrecs
: Number of records written to the logical-log buffer

llgpagewrites
: Number of pages written to the logical-log buffer

llgwrites
: Number of writes from the logical-log buffer to the logical-log files

numckpts
: Number of checkpoints that have occurred since the database server was brought online)

ckptwaits
: Number of times that threads waited for a checkpoint to finish entering a critical section during a checkpoint

value
: Values for **numckpts** and **ckptwaits**

Turn automatic LRU tuning on or off

Use the AUTO_LRU_TUNING configuration parameter to enable or disable automatic LRU tuning when the database server starts.

If the RTO_SERVER_RESTART configuration parameter is set, the database server automatically triggers checkpoints so that it can bring the server online within the specified time. The database server prints warning messages in the message log if the server cannot meet the RTO_SERVER_RESTART policy.

To turn off automatic LRU tuning for a particular session, issue an **onmode –wm AUTO_LRU_TUNING=***0* command.

To turn on automatic LRU tuning after turning it off during a session, issue an **onmode –wm AUTO_LRU_TUNING=***1* command

Automatic LRU tuning changes affect all buffer pools and adjust **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

For more information about LRU tuning, see the *IBM® Informix® Performance Guide*.

Fault tolerance

- Mirroring
- Using mirroring
- Consistency checking

Mirroring

These topics describe the database server mirroring feature. For instructions on how to perform mirroring tasks, see Using mirroring.

- Mirroring
- Mirroring process

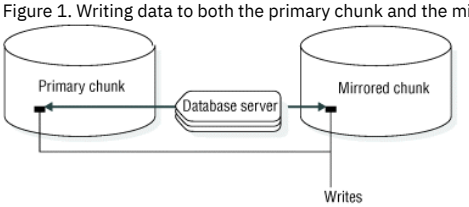
Related concepts:
Feature configuration
Chunks

Mirroring

Mirroring is a strategy that pairs a *primary* chunk of one defined dbspace, blobspace, or sbspace with an equal-sized *mirror chunk*.

Every write to the primary chunk is automatically accompanied by an identical write to the mirror chunk. This concept is illustrated in the following figure. If a failure occurs on the primary chunk, mirroring enables you to read from and write to the mirror chunk until you can recover the primary chunk, all without interrupting user access to data.

Figure 1. Writing data to both the primary chunk and the mirror chunk



Mirroring is not supported on disks that are managed over a network. The same database server instance must manage all the chunks of a mirrored set.

- Benefits of mirroring
- Costs of mirroring
- Consequences of not mirroring
- Data to mirror
- Alternatives to mirroring

Benefits of mirroring

If media failure occurs, mirroring provides the database server administrator with a means of recovering data without taking the database server offline. This feature results in greater reliability and less system downtime. Furthermore, applications can continue to read from and write to a database whose primary chunks are on the affected media, provided that the chunks that mirror this data are located on separate media.

Any critical database must be located in a mirrored dbspace. The root dbspace, which contains the database server reserved pages, must be mirrored.

Costs of mirroring

Disk-space costs and performance costs are associated with mirroring. The disk-space cost is due to the additional space required for storing the mirror data. The performance cost results from performing writes to both the primary and mirror chunks. The use of multiple virtual processors for disk writes reduces this performance

cost. The use of *split reads*, whereby the database server reads data from either the primary chunk or the mirror chunk, depending on the location of the data within the chunk, actually causes performance to improve for read-only data. For more information about how the database server performs reads and writes for mirror chunks, see Actions during processing.

Consequences of not mirroring

If you do not mirror your dbspaces, the frequency with which you must restore from a storage-space backup after media failure increases.

When a mirror chunk suffers media failure, the database server reads exclusively from the chunk that is still online until you bring the down chunk back online. When the second chunk of a mirrored pair goes down, the database server cannot access the data stored on that chunk. If the chunk contains logical-log files, the physical log, or the root dbspace, the database server goes offline immediately. If the chunk does not contain logical-log files, the physical log, or the root dbspace, the database server can continue to operate, but threads cannot read from or write to the down chunk. If an unmirrored chunk goes down, you must restore it by recovering the dbspace from a backup.

Data to mirror

Ideally, you must mirror all of your data. If disk space is an issue, however, you might not be able to do so. In this case, select certain critical chunks to mirror.

Critical chunks always include the chunks that are part of the root dbspace, the chunk that stores the logical-log files, and the chunk that stores the physical logs. If any one of these critical chunks fail, the database server goes offline immediately.

If some chunks hold data that is critical to your business, give these chunks high priority for mirroring.

Also give priority for mirroring to chunks that store frequently used data. This action ensures that the activities of many users would not be halted if one widely used chunk goes down.

Alternatives to mirroring

Mirroring, as explained in this manual, is a database server feature. Your operating system or hardware might provide alternative mirroring solutions.

If you are considering a mirroring feature provided by your operating system instead of database server mirroring, compare the implementation of both features before you decide which to use. The slowest step in the mirroring process is the actual writing of data to disk. The database server strategy of performing writes to mirror chunks in parallel helps to reduce the time required for this step. (See Disk writes to mirror chunks.) In addition, database server mirroring uses split reads to improve read performance. (See Disk reads from mirror chunks.) Operating-system mirroring features that do not use parallel mirror writes and split reads might provide inferior performance.

Nothing prevents you from running database server mirroring and operating-system mirroring at the same time. They run independently of each other. In some cases, you might decide to use both the database server mirroring and the mirroring feature provided by your operating system. For example, you might have both database server data and other data on a single disk drive. You can use the operating-system mirroring to mirror the other data and database server mirroring to mirror the database server data.

- Logical volume managers
- Hardware mirroring
- External backup and restore

Logical volume managers

Logical volume managers are an alternative mirroring solution. Some operating-system vendors provide this type of utility to have multiple disks seem to be one file system. Saving data to more than two disks gives you added protection from media failure, but the additional writes have a performance cost.

Hardware mirroring

Another solution is to use hardware mirroring such as redundant array of inexpensive disks (RAID). An advantage of this type of hardware mirroring is that it requires less disk space than database server mirroring does to store the same amount of data to prevent media failure.

Some hardware mirroring systems support *hot swapping*. You can swap a bad disk while keeping the database server online. Reducing I/O activity before performing a hot swap is recommended.

Important: If problems occur with the database server while using hardware mirroring, see the operating-system or disk documentation or technical support for assistance.

External backup and restore

If you use hardware disk mirroring, you can get your system online faster with external backup and restore than with conventional ON-Bar commands. For more information about external backup and restore, see the *IBM® Informix® Backup and Restore Guide*.

Mirroring process

This section describes the mirroring process in greater detail. For instructions on how to perform mirroring operations such as creating mirror chunks, starting mirroring, changing the status of mirror chunks, and so forth, see Using mirroring.

- Creation of a mirror chunk
- Mirror status flags
- Recovery
- Actions during processing
- Result of stopping mirroring
- Structure of a mirror chunk

Creation of a mirror chunk

When you specify a mirror chunk, the database server copies all the data from the primary chunk to the mirror chunk. This copy process is known as *recovery*. Mirroring begins as soon as recovery is complete.

The recovery procedure that marks the beginning of mirroring is delayed if you start to mirror chunks within a dbspace that contains a logical-log file. Mirroring for dbspaces that contain a logical-log file does not begin until you create a level-0 backup of the root dbspace. The delay ensures that the database server can use the mirrored logical-log files if the primary chunk that contains these logical-log files becomes unavailable during a dbspace restore.

The level-0 backup copies the updated database server configuration information, including information about the new mirror chunk, from the root dbspace reserved pages to the backup. If you perform a data restore, the updated configuration information at the beginning of the backup directs the database server to look for the mirrored copies of the logical-log files if the primary chunk becomes unavailable. If this new storage-space backup information does not exist, the database server is unable to take advantage of the mirrored log files.

For similar reasons, you cannot mirror a dbspace that contains a logical-log file while a dbspace backup is being created. The new information that must be in the first block of the dbspace backup tape cannot be copied there after the backup has begun.

For more information about creating mirror chunks, see Using mirroring.

Mirror status flags

Dbspaces, blobspaces, and sbspaces have status flags that indicate whether they are mirrored or unmirrored.

You must perform a level-0 backup of the root dbspace before mirroring starts.

Chunks have status flags that indicate the following information:

- Whether the chunk is a primary or mirror chunk
- Whether the chunk is currently online, down, a new mirror chunk that requires a level-0 backup of the root dbspace, or in the process of being recovered

For descriptions of these chunk status flags, see the description of the **onstat -d** option in the *IBM® Informix® Administrator's Reference*. For information about how to display these status flags, see Monitor disk usage.

Recovery

When the database server recovers a mirror chunk, it performs the same recovery procedure that it uses when mirroring begins. The mirror-recovery process consists of copying the data from the existing online chunk onto the new, repaired chunk until the two are identical.

When you initiate recovery, the database server puts the down chunk in recovery mode and copies the information from the online chunk to the recovery chunk. When the recovery is complete, the chunk automatically receives online status. You perform the same steps whether you are recovering the primary chunk of a mirrored pair or recovering the mirror chunk.
Tip: You can still use the online chunk during the recovery process. If data is written to a page that has already been copied to the recovery chunk, the database server updates the corresponding page on the recovery chunk before it continues with the recovery process.
For information about how to recover a down chunk, see the information about recovering a mirror chunk on page Recover a mirror chunk.

Actions during processing

These topics explain some of the details of disk I/O for mirror chunks and how the database server handles media failure for these chunks.

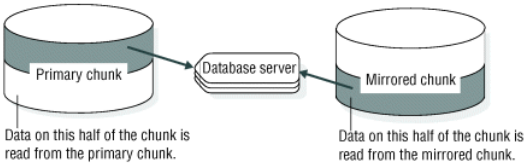
- Disk writes to mirror chunks
- Disk reads from mirror chunks
- Detection of media failures
- Chunk recovery

Disk writes to mirror chunks

During database server processing, the database server performs mirroring by executing two parallel writes for each modification: one to the primary chunk and one to the mirror chunk.

Disk reads from mirror chunks

The database server uses mirroring to improve read performance because two versions of the data are located on separate disks. A data page is read from either the primary chunk or the mirror chunk, depending on which half of the chunk includes the address of the data page. This feature is called a *split read*. Split reads improve performance by reducing the disk-seek time. Disk-seek time is reduced because the maximum distance over which the disk head must travel is reduced by half. The following figure illustrates a split read.
Figure 1. Split read reducing the maximum distance over which the disk head must travel



Detection of media failures

The database server checks the return code when it first opens a chunk and after any read or write. Whenever the database server detects that a primary (or mirror) chunk device has failed, it sets the chunk-status flag to down (D). For information about chunk-status flags, see Mirror status flags.

If the database server detects that a primary (or mirror) chunk device has failed, reads and writes continue for the one chunk that remains online. This statement is true even if the administrator intentionally brings down one of the chunks.

After the administrator recovers the down chunk and returns it to online status, reads are again split between the primary and mirror chunks, and writes are made to both chunks.

Chunk recovery

The database server uses asynchronous I/O to minimize the time required for recovering a chunk. The read from the chunk that is online can overlap with the write to the down chunk, instead of the two processes occurring serially. That is, the thread that performs the read is not required to wait until the thread that performs the write has finished before it reads more data.

Result of stopping mirroring

When you end mirroring, the database server immediately frees the mirror chunks and makes the space available for reallocation. The action of ending mirroring takes only a few seconds.

Create a level-0 backup of the root dbspace after you end mirroring to ensure that the reserved pages with the updated mirror-chunk information are copied to the backup. This action prevents the restore procedure from assuming that mirrored data is still available.

Structure of a mirror chunk

The mirror chunk contains the same control structures as the primary chunk, as follows:

- Mirrors of blobspace chunks contain blobspace overhead pages.
- Mirrors of dbspace chunks contain dbspace overhead pages.
- Mirrors of sbspaces contain metadata pages.

For information about these structures, see the section on the structure of a mirror chunk in the disk structures and storage chapter of the *IBM® Informix® Administrator's Reference*.

A display of disk-space use, provided by one of the methods explained under Monitor chunks, always indicates that the mirror chunk is *full*, even if the primary chunk has free space. The full mirror chunk indicates that none of the space in the chunk is available for use other than as a mirror of the primary chunk. The status remains full for as long as both primary chunk and mirror chunk are online.

If the primary chunk goes down and the mirror chunk becomes the primary chunk, disk-space allocation reports then accurately describe the fullness of the new primary chunk.

Using mirroring

These topics describe the various mirroring tasks that are required to use the database server mirroring feature. It provides an overview of the steps required for mirroring data.

- Preparing to mirror data
- Enable the MIRROR configuration parameter
  You can set the MIRROR configuration parameter to enable (or disable) mirroring.
- Allocate disk space for mirrored data
- Using mirroring
- Manage mirroring
  You can use the **onspaces** utility to manage mirroring.

Preparing to mirror data

This section describes how to start mirroring data on a database server that is not running with the mirroring function enabled.

To prepare to mirror data:

1. Take the database server offline and enable mirroring.
   See Enable the MIRROR configuration parameter.
2. Bring the database server back online.
3. Allocate disk space for the mirror chunks.
   You can allocate this disk space at any time, as long as the disk space is available when you specify mirror chunks in the next step. The mirror chunks must be on a different disk than the corresponding primary chunks. See Allocate disk space for mirrored data.
4. Choose the dbspace, blobspace, or sbspace that you want to mirror, and specify a mirror-chunk path name and offset for each primary chunk in that storage space. The mirroring process starts after you perform this step. Repeat this step for all the storage spaces that you want to mirror. See Using mirroring.

Enable the MIRROR configuration parameter

You can set the MIRROR configuration parameter to enable (or disable) mirroring.

Enabling mirroring starts the database server functions required for mirroring tasks. However, when you enable mirroring, you do not initiate the mirroring process. Mirroring does not actually start until you create mirror chunks for a dbspace, blobspace, or sbspace. See Using mirroring.

Enable mirroring when you initialize the database server if you plan to create a mirror for the root dbspace as part of initialization; otherwise, leave mirroring disabled. If you later decide to mirror a storage space, you can change the value of the MIRROR configuration parameter.

To enable mirroring for the database server, you must set the MIRROR parameter in onconfig to `1`. The default value of MIRROR is `0`, indicating that mirroring is disabled.

Do not set the MIRROR parameter to `1` if you are not using mirroring.

To change the value of MIRROR, you can edit the onconfig file with a text editor while the database server is in online mode. After you change the onconfig file, take the database server offline and then to quiescent for the change to take effect.

Allocate disk space for mirrored data

Before you can create a mirror chunk, you must allocate disk space for this purpose. You can allocate either raw disk space or cooked file space for mirror chunks. For an explanation of allocating disk space, see Allocate disk space.

Always allocate disk space for a mirror chunk on a different disk than the corresponding primary chunk with, ideally, a different controller. You can use this setup to access the mirror chunk if the disk on which the primary chunk is located goes down, or vice versa.

- Link chunks (UNIX)
- Relink a chunk to a device after a disk failure

Link chunks (UNIX)

Use the UNIX link (**ln**) command to link the actual files or raw devices of the mirror chunks to mirror path names. If a disk failure occurs, you can link a new file or raw device to the path name, eliminating the necessity to physically replace the disk that failed before the chunk is brought back online.

Relink a chunk to a device after a disk failure

On UNIX, if the disk on which the actual mirror file or raw device is located goes down, you can relink the chunk to a file or raw device on a different disk. If you do this, you can recover the mirror chunk before the disk that failed is brought back online. Typical UNIX commands that you can use for relinking are shown in the following examples.

The original setup consists of a primary root chunk and a mirror root chunk, which are linked to the actual raw disk devices, as follows:

```
ln -l
lrwxrwxrwx 1 informix 10 May 3 13:38 /dev/root@->/dev/rxy0h
lrwxrwxrwx 1 informix 10 May 3 13:40 /dev/mirror_root@->/dev/rsd2b
```

Assume that the disk on which the raw device /dev/rsd2b is located has gone down. You can use the **rm** command to remove the corresponding symbolic link, as follows:

```
rm /dev/mirror_root
```

Now you can relink the mirror chunk path name to a raw disk device, on a disk that is running, and proceed to recover the chunk, as follows:

```
ln -s /dev/rab0a /dev/mirror_root
```


Using mirroring

Mirroring starts when you create a mirror chunk for each primary chunk in a dbspace, blobspace, or sbspace.

When you create a mirror chunk, the database server copies data from the primary chunk to the mirror chunk. When this process is complete, the database server begins mirroring data. If the primary chunk contains logical-log files, the database server does not copy the data immediately after you create the mirror chunk but waits until you perform a level-0 backup. For an explanation of this behavior see Creation of a mirror chunk.
Important: You must always start mirroring for an entire dbspace, blobspace, or sbspace. The database server does not permit you to select particular chunks in a dbspace, blobspace, or sbspace to mirror. You must create mirror chunks for every chunk in the space.
You start mirroring a storage space when you perform the following operations:

- Create a mirrored root dbspace during system initialization
- Change the status of a dbspace from unmirrored to mirrored
- Create a mirrored dbspace, blobspace, or sbspace

Each of these operations requires you to create mirror chunks for the existing chunks in the storage space.

- Mirroring the root dbspace during initialization
- Change the mirror status

Mirroring the root dbspace during initialization

If you enable mirroring when you initialize the database server, you can also specify a mirror path name and offset for the root chunk. The database server creates the mirror chunk when the server is initialized. However, because the root chunk contains logical-log files, mirroring does not actually start until you perform a level-0 backup.

To specify the root mirror path name and offset, set the values of MIRRORPATH and MIRROROFFSET in the onconfig file before you start the database server.

If you do not provide a mirror path name and offset, but you do want to start mirroring the root dbspace, you must change the mirroring status of the root dbspace after the database server is initialized.

Change the mirror status

You can make the following two changes to the status of a mirror chunk:

- Change a mirror chunk from online to down
- Change a mirror chunk from down to recovery

You can take down or restore a chunk only if it is part of a mirrored pair. You can take down either the primary chunk or the mirror chunk, as long as the other chunk in the pair is online.

For information about how to determine the status of a chunk, see Monitor disk usage.

Manage mirroring

You can use the **onspaces** utility to manage mirroring.

For a full description of the **onspaces** syntax, see The onspaces utility in the *IBM® Informix® Administrator's Reference*.

- Start mirroring for unmirrored storage spaces
  You can prepare mirroring for a dbspace, blobspace, or sbspace at any time. However, the mirroring does not start until you perform a level-0 backup.
- Start mirroring for new storage spaces
  You can also start mirroring when you create a new dbspace, blobspace, or sbspace.
- Add mirror chunks
- Swap mirror chunk
- Take down a mirror chunk
- Recover a mirror chunk
- End mirroring

Start mirroring for unmirrored storage spaces

You can prepare mirroring for a dbspace, blobspace, or sbspace at any time. However, the mirroring does not start until you perform a level-0 backup.

- Start mirroring for unmirrored dbspaces using onspaces

Start mirroring for unmirrored dbspaces using onspaces

You can use the **onspaces** utility to start mirroring a dbspace, blobspace, or sbspace. For example, the following **onspaces** command starts mirroring for the dbspace **db_project**, which contains two chunks, **data1** and **data2**:

```
onspaces -m db_project\
-p /dev/data1 -o 0 -m /dev/mirror_data1 0\
-p /dev/data2 -o 5000 -m /dev/mirror_data2 5000
```

The following example shows how to turn on mirroring for a dbspace called **sp1**. You can either specify the primary path, primary offset, mirror path, and mirror offset in the command or in a file.

```
onspaces -m sp1 -f mirfile
```

The mirfile file contains the following line:

```
/ix/9.3/sp1 0 /ix/9.2/sp1mir 0
```

In this line, /ix/9.3/sp1 is the primary path, 0 is the primary offset, /ix/9.3/sp1mir is the mirror path, and 0 is the mirror offset.

Start mirroring for new storage spaces

You can also start mirroring when you create a new dbspace, blobspace, or sbspace.

- Start mirroring for new spaces using onspaces

Start mirroring for new spaces using onspaces

You can use the **onspaces** utility to create a mirrored dbspace. For example, the following command creates the dbspace **db_acct** with an initial chunk **/dev/chunk1** and a mirror chunk /dev/mirror_chk1:

```
onspaces -c -d db_acct -p /dev/chunk1 -o 0 -s 2500 -m /dev/mirror_chk1 0
```

Another way to start mirroring is to select Index by Utility > onspaces -m.

Add mirror chunks

If you add a chunk to a dbspace, blobspace, or sbspace that is mirrored, you must also add a corresponding mirror chunk.

- Add mirror chunks using onspaces

Add mirror chunks using onspaces

You can use the **onspaces** utility to add a primary chunk and its mirror chunk to a dbspace, blobspace, or sbspace. The following example adds a chunk, **chunk2**, to the **db_acct** dbspace. Because the dbspace is mirrored, a mirror chunk, **mirror_chk2**, is also added.

```
onspaces -a db_acct -p /dev/chunk2 -o 5000 -s 2500 -m /dev/mirror_chk2 5000
```


Swap mirror chunk

You can use the **swap_mirror** command to swap a mirror chunk and a primary chunk, making the original primary chunk the mirror, and the original mirror chunk the new primary. This operation is especially useful as an on-line method of migrating chunks to a newer, faster set of disk drives.

To swap a single mirrored chunk, use the following command:

execute function task("modify chunk swap_mirror",<chunk number>).

To swap all the chunks in a mirrored space, use the following command:

execute function task("modify space swap_mirrors","<space name>").

Note: You cannot swap a chunk if either its primary or mirror is down.

Take down a mirror chunk

When a mirror chunk is down, the database server cannot write to it or read from it. You might take down a mirror chunk to relink the chunk to a different device. (See Relink a chunk to a device after a disk failure.)

Taking down a chunk is not the same as ending mirroring. You end mirroring for a complete dbspace, which causes the database server to drop all the mirror chunks for that dbspace.

- Take down mirror chunks using onspaces

Take down mirror chunks using onspaces

You can use the **onspaces** utility to take down a chunk. The following example takes down a chunk that is part of the dbspace **db_acct**:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -D
```


Recover a mirror chunk

To begin mirroring the data in the chunk that is online, you must recover the down chunk.

- Recover a mirror chunk using onspaces

Recover a mirror chunk using onspaces

You can use the **onspaces -s** utility to recover a down chunk. For example, to recover a chunk that has the path name /dev/mirror_chk1 and an offset of 0 KB, issue the following command:

```
onspaces -s db_acct -p /dev/mirror_chk1 -o 0 -O
```


End mirroring

When you end mirroring for a dbspace, blobspace, or sbspace, the database server immediately releases the mirror chunks of that space. These chunks are immediately available for reassignment to other storage spaces.

Only users **informix** and **root** on UNIX or members of the **Informix-Admin** group on Windows can end mirroring.

You cannot end mirroring if any of the primary chunks in the dbspace are down. The system can be in online mode when you end mirroring.

- End mirroring using onspaces

End mirroring using onspaces

You can end mirroring with the **onspaces** utility. For example, to end mirroring for the root dbspace, enter the following command:

```
onspaces -r rootdbs
```

Another way to end mirroring is to select Index by Utility > onspaces -r.

Consistency checking

IBM® Informix® database servers are designed to detect database server malfunctions or problems caused by hardware or operating-system errors. It detects problems by performing *assertions* in many of its critical functions. An assertion is a consistency check that verifies that the contents of a page, structure, or other entity match what would otherwise be assumed.

When one of these checks finds that the contents are incorrect, the database server reports an assertion failure and writes text that describes the check that failed in the database server message log. The database server also collects further diagnostics information in a separate file that might be useful to IBM Informix Technical Support staff.

These topics provide an overview of consistency-checking measures and ways of handling inconsistencies.

- Perform periodic consistency checking
- Deal with corruption
- Collect diagnostic information
- Disable I/O errors
- Monitor the database server for disabling I/O errors

Related reference:
Database server maintenance tasks

Perform periodic consistency checking

To gain the maximum benefit from consistency checking and to ensure the integrity of dbspace backups, you must periodically take the following actions:

- Verify that all data and the database server overhead information is consistent.
- Check the message log for assertion failures while you verify consistency.
- Create a level-0 dbspace backup after you verify consistency.

The following topics describe each of these actions.

- Verify consistency
- Monitor for data inconsistency
- Retain consistent level-0 backups

Verify consistency

Because of the time required for this check and the possible contention that the check can cause, schedule this check for times when activity is at its lowest. You must perform this check just before you create a level-0 backup.

Run the commands shown in the following table as part of the consistency check.

Table 1. Checking data consistency

Type of validation	Command
System catalog tables	**oncheck -cc**
Data	**oncheck -cD *dbname***
Extents	**oncheck -ce**
Indexes	**oncheck -cI *dbname***
Reserved pages	**oncheck -cr**
Logical logs and reserved pages	**oncheck -cR**
Metadata and smart large objects	**oncheck -cs**

You can run each of these commands while the database server is in online mode. For information about how each command locks objects as it checks them and which users can perform validations, see **oncheck** in the *IBM® Informix® Administrator's Reference*.

In most cases, if one or more of these validation procedures detects an error, the solution is to restore the database from a dbspace backup. However, the source of the error might also be your hardware or operating system.

- Validate system catalog tables
- Validate data pages
- Validate extents
- Validate indexes
  If an index is corrupted, the database server cannot use it in queries.
- Validate logical logs
- Validate reserved pages
- Validate metadata

Validate system catalog tables

To validate system catalog tables, use the **oncheck -cc** command.

Each database contains its own system catalog, which contains information about the database tables, columns, indexes, views, constraints, stored procedures, and privileges.

If a warning is displayed when validation completes, its only purpose is to alert you that no records of a specific type were found. These warnings do not indicate any problem with your data, your system catalog, or even your database design. This warning indicates only that no synonym exists for any table; that is, the system catalog contains no records in the table **syssyntable**. For example, the following warning might be displayed if you validate system catalog tables for a database that has no synonyms defined for any table:

```
WARNING: No syssyntable records found.
```

However, if you receive an error message when you validate system catalog tables, the situation is quite different. Contact IBM® Informix® Technical Support immediately.

Validate data pages

To validate data pages, use the **oncheck -cD** command.

If data-page validation detects errors, try to unload the data from the specified table, drop the table, recreate the table, and reload the data. For information about loading and unloading data, see the *IBM® Informix® Migration Guide*. If this procedure does not succeed, perform a data restore from a storage-space backup.

Validate extents

To validate extents in every database, use the **oncheck -ce** command.

Extents must not overlap. If this command detects errors, perform a data restore from a storage-space backup.

Validate indexes

If an index is corrupted, the database server cannot use it in queries.

You can validate indexes on each of the tables in the database by using the **oncheck -cI** command.

In addition, the Scheduler task **bad_index_alert** looks for indexes that have been marked as corrupted by the server. This task runs nightly. An entry is made into the **sysadmin:ph_alert** table for each corrupted index found by the task.

If an index is corrupted, drop and recreate it.

Validate logical logs

To validate logical logs and the reserved pages, use the **oncheck -cR** command.

Validate reserved pages

To validate reserved pages, use the **oncheck -cr** command.

Reserved pages are pages that are located at the beginning of the initial chunk of the root dbspace. These pages contain the primary database server overhead information. If this command detects errors, perform a data restore from storage-space backup.

This command might provide warnings. In most cases, these warnings call your attention to situations of which you are already aware.

Validate metadata

Run **oncheck -cs** for each database to validate metadata for all smart large objects in a database. If necessary, restore the data from a dbspace backup.

Monitor for data inconsistency

If the consistency-checking code detects an inconsistency during database server operation, an assertion failure is reported to the database server message log. (See the message-log topics in the *IBM® Informix® Administrator's Reference*.)

- Read assertion failures in the message log and dump files
- Validate table and tblspace data

Read assertion failures in the message log and dump files

The following example shows the form that assertion failures take in the message log.

```
Assert Failed: Short description of what failed
   Who: Description of user/session/thread running at the time
   Result: State of the affected database server entity
   Action: What action the database server administrator should take
   See Also: file(s) containing additional diagnostics
```

The `See Also:` line contains one or more of the following file names:

- af.xxx
- shmem.xxx
- gcore.xxx
- gcore.xxx
- */path name*/core

In all cases, xxx is a hexadecimal number common to all files associated with the assertion failures of a single thread. The files af.xxx, shmem.xxx, and gcore.xxx are in the directory that the ONCONFIG parameter DUMPDIR specifies.

The file af.xxx contains a copy of the assertion-failure message that was sent to the message log, and the contents of the current, relevant structures and data buffers.

The file shmem.xxx contains a complete copy of the database server shared memory at the time of the assertion failure, but only if the ONCONFIG parameter DUMPSHMEM is set to `1` or to `2`.

UNIX only: On UNIX, gcore.xxx contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPGCORE is set to `1` and your operating system supports the **gcore** utility. The core file contains a core dump of the database server virtual process on which the thread was running at the time, but only if the ONCONFIG parameter DUMPCORE is set to `1`. The path name for the core file is the directory from which the database server was last invoked.

Validate table and tblspace data

To validate table and tblspace data, use the **oncheck -cD** command on the database or table.

Most of the general assertion-failure messages are followed by additional information that usually includes the tblspace where the error was detected. If this check verifies the inconsistency, unload the data from the table, drop the table, recreate the table, and reload the data. Otherwise, no other action is required.

In many cases, the database server stops immediately when an assertion fails. However, when failures seem to be specific to a table or smaller entity, the database server continues to run.

When an assertion fails because of inconsistencies on a data page that the database server accesses on behalf of a user, an error is also sent to the application process. The SQL error depends on the operation in progress. However, the ISAM error is almost always either -105 or -172, as follows:

```
-105 ISAM error: bad isam file format
-172 ISAM error: Unexpected internal error
```

For additional details about the objectives and contents of messages, see the topics about message-log messages in the *IBM® Informix® Administrator's Reference*.

Retain consistent level-0 backups

After you perform the checks described in Verify consistency without errors, create a level-0 backup. Retain this storage-space backup and all subsequent logical-log backup tapes until you complete the next consistency check. Perform the consistency checks before every level-0 backup. If you do not, then at minimum keep all the tapes necessary to recover from the storage-space backup that was created immediately after the database server was verified to be consistent.

Deal with corruption

This section describes some of the symptoms of database server system corruption and actions that the database server or you, as administrator, can take to resolve the problems. Corruption in a database can occur as a consequence of hardware or operating-system problems, or from some unknown database server problems. Corruption can affect either data or database server overhead information.

- Find symptoms of corruption
  You can find information about corruption several different ways.
- Fix index corruption
- Fix I/O errors on a chunk

Find symptoms of corruption

You can find information about corruption several different ways.

The database server alerts the user and administrator to possible corruption in the following ways:

- Error messages reported to the application state that pages, tables, or databases cannot be found. One of the following errors is always returned to the application if an operation has failed because of an inconsistency in the underlying data or overhead information:

  ```
  -105 ISAM error: bad isam file format
  -172 ISAM error: Unexpected internal error
  ```

- Assertion-failure reports are written to the database server message log. They always indicate files that contain additional diagnostic information that can help you determine the source of the problem. See Verify consistency.
- The **oncheck** utility returns errors
- The **ph_alert** table shows information about corrupted indexes.

Fix index corruption

At the first indication of corruption, run the **oncheck -cI** command to determine if corruption exists in the index.

If you check indexes while the database server is in online mode, **oncheck** detects the corruption but does not prompt you for repairs. If corruption exists, you can drop and recreate the indexes using SQL statements while you are in online mode (the database server locks the table and index). If you run **oncheck -cI** in quiescent mode and corruption is detected, you are prompted to confirm whether the utility attempts to repair the corruption.

Fix I/O errors on a chunk

If an I/O error occurs during the database server operation, the status of the chunk on which the error occurred changes to down.

If a chunk is down, the **onstat -d** display shows the chunk status as `PD-` for a primary chunk and `MD-` for a mirror chunk. For an example of **onstat -d** output, see the *IBM® Informix® Administrator's Reference*.

In addition, the message log lists a message with the location of the error and a suggested solution. The listed solution is a possible fix, but does not always correct the problem.

If the down chunk is mirrored, the database server continues to operate using the mirror chunk. Use operating-system utilities to determine what is wrong with the down chunk and correct the problem. You must then direct the database server to restore mirror chunk data.

For information about recovering a mirror chunk, see Recover a mirror chunk.

If the down chunk is not mirrored and contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates a stop action. Otherwise, the database server can continue to operate but cannot write to or read from the down chunk or any other chunks in the dbspace of that chunk. You must take steps to determine why the I/O error occurred, correct the problem, and restore the dbspace from a backup.

If you take the database server to offline mode when a chunk is marked as down (`D`), you can restart the database server, provided that the chunk marked as down does not contain critical data (logical-log files, the physical log, or the root dbspace).

Collect diagnostic information

Several ONCONFIG parameters affect the way in which the database server collects diagnostic information. Because an assertion failure is generally an indication of an unforeseen problem, notify IBM® Informix® Technical Support whenever one occurs. The diagnostic information collected is intended for the use of IBM Informix technical staff. The contents and use of af.*xxx* files and shared core are not further documented.

To determine the cause of the problem that triggered the assertion failure, it is critically important that you not delete diagnostic information until IBM Informix Technical Support indicates that you can do so. The af.xxx file often contains information that they require to resolve the problem.

Several ONCONFIG parameters direct the database server to preserve diagnostic information whenever an assertion failure is detected or whenever the database server enters into an end sequence:

- DUMPDIR
- DUMPSHMEM
- DUMPCNT
- DUMPCORE
- DUMPGCORE

For more information about the configuration parameters, see the *IBM Informix Administrator's Reference*.

You decide whether to set these parameters. Diagnostic output can use a large amount of disk space. (The exact content depends on the environment variables set and your operating system.) The elements of the output can include a copy of shared memory and a core dump.
Tip: A *core dump* is an image of a process in memory at the time that the assertion failed. On some systems, core dumps include a copy of shared memory. Core dumps are useful only if this is the case.
Database server administrators with disk-space constraints might prefer to write a script that detects the presence of diagnostic output in a specified directory and sends the output to tape. This approach preserves the diagnostic information and minimizes the amount of disk space used.

Disable I/O errors

IBM® Informix® divides disabling I/O errors into two general types: destructive and nondestructive. A disabling I/O error is destructive when the disk that contains a database becomes damaged in some way. This type of event threatens the integrity of data, and the database server marks the chunk and dbspace as down. The database server prohibits access to the damaged disk until you repair or replace the disk and perform a physical and logical restore.

A disabling I/O error is nondestructive when the error does not threaten the integrity of your data. Nondestructive errors occur when someone accidentally disconnects a cable, you somehow erase the symbolic link that you set up to point to a chunk, or a disk controller becomes damaged.

Before the database server considers an I/O error to be disabling, the error must meet two criteria. First, the error must occur when the database server attempts to perform an operation on a chunk that has at least one of the following characteristics:

- The chunk has no mirror.
- The primary or mirror companion of the chunk under question is offline.

Second, the error must occur when the database server attempts unsuccessfully to perform one of the following operations:

- Seek, read, or write on a chunk
- Open a chunk
- Verify that chunk information about the first used page is valid
  The database server performs this verification as a sanity check immediately after it opens a chunk.

You can prevent the database server from marking a dbspace as down while you investigate disabling I/O errors. If you find that the problem is trivial, such as a loose cable, you can bring the database server offline and then online again without restoring the affected dbspace from backup. If you find that the problem is more serious, such as a damaged disk, you can use **onmode -O** to mark the affected dbspace as down and continue processing.

Monitor the database server for disabling I/O errors

The database server notifies you about disabling I/O errors in two ways:

- Message log
- Event alarms

- The message log to monitor disabling I/O errors
  The database server sends a message to the message log when a disabling I/O error occurs.
- Event alarms to monitor disabling I/O errors
  When a dbspace incurs a disabling I/O error, the database server passes the specified values as parameters to your event-alarm executable file.
- No bad-sector mapping

The message log to monitor disabling I/O errors

The database server sends a message to the message log when a disabling I/O error occurs.

The message is:

```
Assert Failed: Chunk {chunk-number} is being taken OFFLINE.
Who: Description of user/session/thread running at the time
Result: State of the affected database server entity
Action: What action the database server administrator should take
See Also: DUMPDIR/af.uniqid containing more diagnostics
```

The result and action depend on the current setting of ONDBSPACEDOWN, as described in the following table.

ONDBSPACEDOWN setting	Result	Action
0	Dbspace {*space_name*} is disabled.	Restore dbspace {*space_name*}.
	Blobspace {*space_name*} is disabled.	Restore blobspace {*space_name*}.
1	The database server must stop.	Shut down and restart the database server.
2	The database server blocks at next checkpoint.	Use **onmode -k** to shut down, or use **onmode -O** to override.

The value of ONDBSPACEDOWN has no affect on temporary dbspaces. For temporary dbspaces, the database server continues processing regardless of the ONDBSPACEDOWN setting. If a temporary dbspace requires fixing, you can drop and recreate it.

For more information about interpreting messages that the database server sends to the message log, see the topics about message-log messages in the *IBM® Informix® Administrator's Reference*.

Event alarms to monitor disabling I/O errors

When a dbspace incurs a disabling I/O error, the database server passes the specified values as parameters to your event-alarm executable file.

Event alarm values:

Severity
: 4 (Emergency)

Class
: 5

Class message
: Dbspace is disabled: 'dbspace-name'

Specific message
: Chunk {chunk-number} is being taken OFFLINE.

Event ID
: 5001

If you want the database server to use event alarms to notify you about disabling I/O errors, write a script that the database server executes when it detects a disabling I/O error. For information about how to set up this executable file that you write, see the appendix on event alarms and the topics on configuration parameters in the *IBM® Informix® Administrator's Reference*.

No bad-sector mapping

IBM® Informix® relies on the operating system of your host computer for bad-sector mapping. The database server learns of a bad sector or a bad track when it receives a failure return code from a system call. When this situation occurs, the database server retries the access several times to ensure that the condition is not spurious. If the condition is confirmed, the database server marks as down the chunk where the read or write was attempted.

The database server cannot take any action to identify the bad cylinder, track, or sector location because the only information available is the byte displacement within the chunk where the I/O operation was attempted.

If the database server detects an I/O error on a chunk that is not mirrored, it marks the chunk as down. If the down chunk contains logical-log files, the physical log, or the root dbspace, the database server immediately initiates a stop action. Otherwise, the database server can continue to operate, but applications cannot access the down chunk until its dbspace is restored.

High availability and scalability

A successful production environment requires database systems that are always available, with minimal if any planned outages, and that can be scaled quickly and easily as business requirements change.

Businesses must provide continuous access to database resources during planned and unplanned outages. Planned outages include scheduled maintenance of software or hardware. Unplanned outages are unexpected system failures such as power interruptions, network outages, hardware failures, operating system or other software errors. In the event of a disaster, such as an earthquake or a tsunami, there is the possibility of extensive system failure.

Businesses want to avoid overloading a server in the system to ensure data availability and to prevent, for example, denial of service attacks.

Businesses also want to quickly and easily expand their systems as their business grows, during seasonal business peak periods, and for end-of-month or end-of-year processing.

Systems with one or more of the following abilities can be resilient to outages and can improve the availability of data:

Redundancy
: The ability of a system to maintain secondary servers that are copies of the primary server and that can take over from the primary server if a failure occurs.

Failover
: The ability of a system to transfer all of the workload from a failed server to another server.

Workload balancing
: The ability of a system to automatically direct client requests to the server with the most workload capacity.

Scalability
: The ability of a system to take advantage of additional resources, such as database servers, processors, memory, or disk space.

Minimized affect on maintenance
: The ability to maintain all servers quickly and easily so that user applications are affected a little as possible.

- Strategies for high availability and scalability
  IBM Informix database software can be customized to create the appropriate high availability and scalability solution to match your business goals and environment.
- High-availability cluster configuration
- Cluster administration
- Connection management through the Connection Manager
  Connection Managers can control automatic failover for high-availability clusters, monitor client connections and direct requests to appropriate database servers, act as proxy servers and handle client/server communication, and prioritize connections between application servers and the primary server of a high-availability cluster. Connection Managers support high-availability clusters, replicate sets, server sets, and grids.
- Cluster failover, redirection, and restoration
  To maintain availability, you must plan for the failover of primary servers, redirecting client connections from unavailable servers, and restoring the cluster to its original configuration after a failure.

Related concepts:
Feature configuration
XA in high-availability clusters

Strategies for high availability and scalability

IBM® Informix® database software can be customized to create the appropriate high availability and scalability solution to match your business goals and environment.

To determine the best way to customize your database system for high availability and scalability, you must identify the strategies that help you achieve your business goals. You can use the appropriate technologies and components to support those strategies.

- Components supporting high availability and scalability
  IBM Informix database software can be customized to create systems that provide uninterrupted services, minimize maintenance and downtime, automatically redirect client connection requests to the most appropriate database servers, and distribute both processing and storage across hardware.
- Transparent scaling and workload balancing strategies
  IBM Informix servers scale easily and they dynamically balance workloads to ensure optimal use of resources.
- High availability strategies
  IBM Informix can be configured to maximize availability in various business situations.

Components supporting high availability and scalability

IBM® Informix® database software can be customized to create systems that provide uninterrupted services, minimize maintenance and downtime, automatically redirect client connection requests to the most appropriate database servers, and distribute both processing and storage across hardware.

High-availability clusters

A high-availability cluster consists of a primary server that contains the master copy of data, and is securely networked to at least one secondary server that is synchronized with the primary server or has access to the primary server's data. Transactions are sent to secondary servers after they are committed on the primary server, so database data is reliable. If the primary server fails, a secondary server can become the primary server, and client connections can be redirected to the new primary server.

High-availability cluster servers are configured with identical hardware and software. High-availability cluster servers can be in close proximity or geographically remote from each other, and applications can securely connect to any of the cluster servers.

There are three types of secondary servers:

- Shared-disk (SD) secondary servers, which share disk space with the primary server.
- High-availability data replication (HDR) secondary servers, which maintain synchronously or asynchronously updated copies of the entire primary server and can be accessed quickly if the primary server fails.
- Remote stand-alone (RS) secondary servers, which maintain asynchronously updated copies of the entire primary server, and can serve as remote-backup servers in disaster-recovery scenarios.

Enterprise Replication

Using Enterprise Replication, you can maintain complete or partial copies of your data across multiple servers by replicating transactions. Data is replicated asynchronously through transactions captured from the logical log. If a remote database server or network failure occurs, a local database server can service local users, and store transactions to be replicated until remote servers become available. At each database server, Enterprise Replication reads the logical logs to capture locally originating transactions, and then replicates those transactions to other database servers in the Enterprise Replication domain.

Shard clusters

IBM Informix can horizontally partition (*shard*) a table or collection across multiple database servers. The set of database servers that data is sharded across is called a *shard cluster*, and each of the database servers in the set is a *shard server*. Distributing rows or documents across a shard cluster reduces the size of the associated index on each shard server, and distributes performance across your hardware. As your database grows in size, you can scale up by adding more database servers. Horizontal partitioning is also known as sharding.

Rows or documents that are inserted on one shard server can be replicated or sent to other shard servers, depending on the sharding rules you specify. Queries that are performed on a shard server can select data from other shard servers in a shard cluster. When data is sharded based on a replication key that specifies certain segmentation characteristics, queries can skip shard servers that do not contain relevant data. This query optimization is another benefit that comes from data sharding.

Connection management

The Connection Manager is a utility that can monitor the workload and status of database servers in high-availability clusters, Enterprise Replication domains, grids, and server sets, and then use a redirection policy to send client connection requests to the most appropriate database server. Connection Managers can also act as proxy servers to handle client/server communication and circumvent connection issues that are related to firewalls.
Connection Managers can control failover for high-availability clusters, automatically promoting secondary servers to the role of the primary server if the original primary server fails.

If a partial network failure occurs, Connection Managers can prioritize connections between application servers and the primary server of a high-availability cluster, to better define failover.

Grids

A grid is a set of interconnected replication servers, where SQL commands can be propagated from one server to all the others. Grids provide an easier way to administer a large group of servers, update database schemas, run stored procedures and user-defined routines, and administer replication.

- Advantages of data replication

Related concepts:
Connection management through the Connection Manager
Related information:
JSON data sharding
Shard cluster setup
Grid setup and management
Setting up and managing Enterprise Replication

Advantages of data replication

The advantages of data replication do not come without a cost. Data replication obviously requires more storage, and updating replicated data can take more processing time than updating a single object.

You can implement data replication in the logic of client applications by explicitly specifying where data must be updated. However, this method of achieving data replication is costly, prone to error, and difficult to maintain. Instead, the concept of data replication is often coupled with *replication transparency*. Replication transparency is built into a database server (instead of into client applications) to handle automatically the details of locating and maintaining data replicas.

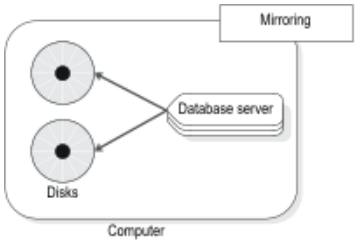
- Clustering versus mirroring
- Clustering versus two-phase commit
- Type of data replicated in clusters
- Primary and secondary database servers

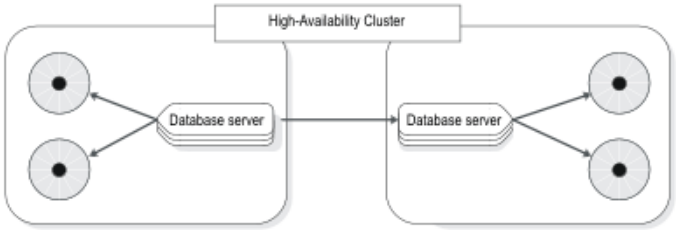
Clustering versus mirroring

Clustering and mirroring are transparent methods for increasing fault tolerant.

Mirroring, described Mirroring, is the mechanism by which a single database server maintains a copy of a specific dbspace on a separate disk. This mechanism protects the data in mirrored dbspaces against disk failure because the database server automatically updates data on both disks and automatically uses the other disk if one of the dbspaces fails.

Alternatively, a cluster duplicates on an entirely separate database server all the data that a database server manages, not just the specified dbspaces. Because clustering involves two separate database servers, it protects the data that these database servers manage, not just against disk failures, but against all types of database server failures, including a computer failure or the catastrophic failure of an entire site.
Figure 1. A comparison of mirroring and clustering





Clustering versus two-phase commit

The two-phase commit protocol, described in detail in Multiphase commit protocols, ensures that transactions are uniformly committed or rolled back across multiple database servers.

In theory, you can take advantage of two-phase commit to replicate data by configuring two database servers with identical data and then defining triggers on one of the database servers that replicate updates to the other database server. However, this sort of implementation has numerous synchronization problems in different failure scenarios. Also, the performance of distributed transactions is inferior to clustering.

Type of data replicated in clusters

A high-availability cluster replicates data in dbspaces and sbspaces, but does not replicate data in blobspaces.

All built-in and extended data types are replicated to the secondary server. User-defined types (UDTs) must be logged and are located in a single database server. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server. For data stored in an sbspace to be replicated, the sbspace must be logged.

Data stored in operating system files or persistent external files or memory objects associated with user-defined routines are not replicated.

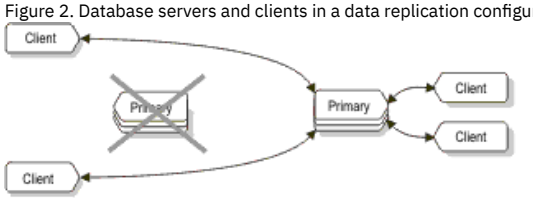
User-defined types, user-defined routines, and DataBlade modules have special installation and registration requirements. For instructions, see How data initially replicates.

Primary and secondary database servers

When you configure a set of database servers to use data replication, one database server is called the *primary* database server, and the others are called *secondary* database servers. (In this context, a database server that does not use data replication is called a *standard* database server.) The secondary server can include any combination of the SD secondary, RS secondary, and HDR secondary servers.

As the following figure illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.
Figure 1. A primary and secondary database server in a data replication configuration

If one of the database servers fails, as the following figure shows, you can redirect the clients that use that database server to the other database server in the pair, which becomes the primary server.
Figure 2. Database servers and clients in a data replication configuration after a failure



Transparent scaling and workload balancing strategies

IBM® Informix® servers scale easily and they dynamically balance workloads to ensure optimal use of resources.

IBM Informix can address these objectives:

- Periodically increase capacity
- Geographically disperse processing and increase capacity
- Balance workload to optimize resource use

Periodically increase capacity

If your business environment experiences peak periods, you might be required to periodically increase capacity. You can increase capacity by adding a remote stand-alone secondary server. That type of secondary server maintains a complete copy of the data, with updates transmitted asynchronously from the primary server over secure network connections. If the amount of data is large and making multiple copies of it is difficult, use shared-disk secondary servers instead of remote stand-alone secondary servers. You can use high-availability data replication (HDR) secondary servers if you want to increase capacity only for reporting (read-only) workloads.

Table 1. Scalability with shared-disk secondary servers

Advantages	Potential disadvantages
• High availability. This secondary server shares disks with the primary server.	• No failover. The secondary server might be configured to run on the same computer hardware as the primary server. • No data redundancy. This secondary server does not maintain a copy of the data. (Use SAN devices for disk storage.) • The primary and secondary servers require the same hardware, operating system, and version of the database server product.

Use an SD secondary server for these reasons:

- Increased reporting capacity
  Multiple secondary servers can offload reporting function without affecting the primary server.
- Server failure backup
  If a failure of the primary server, an SD secondary can be promoted quickly and easily to a primary server. For example, if you are using SAN (storage area network) devices that provide ample and reliable disk storage but you are concerned with server failure, SD secondary servers can provide a reliable backup.

Table 2. Scalability with remote stand-alone secondary servers

Advantages	Potential disadvantages
• Data redundancy. This secondary server maintains a copy of the data. • Failover. The secondary server can be geographically remote from the primary server, such as in another building, another town, or another country. • No requirement to change applications. Client connections to primary or secondary server can be automatically switched in the event of server failure.	• The primary and secondary servers require the same hardware, operating system, and version of the database server product.

Geographically disperse processing and increase capacity

Businesses with offices in various locations might want to use local servers for processing local requests instead of relying on a single, centralized server. In that case, you can set up a network of Enterprise Replication servers. Remote database server outages are tolerated. If database server or network failure occurs, the local database server continues to service local users. The local database server stores replicated transactions in persistent storage until the remote server becomes available. Enterprise Replication on the local server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.

You can also use Enterprise Replication to set up a shard cluster, where your data is horizontally partitioned (*sharded*) across multiple servers. As your capacity requirements grow, you can add additional database servers to the shard cluster, increasing your overall capacity.

Table 3. Scalability with Enterprise Replication

Advantages	Potential disadvantages
• The servers can be in another building, another town, or another country. • The servers can be on different hardware. • The servers can run on different operating systems. • The servers can run different versions of the database server product. • A subset of the data can be replicated (asynchronous, log-based replication). • It is possible to add shared-disk secondary servers to assist the replication servers, using multiple Connection Managers for automatic client redirection. • You can add additional shard servers to an established shard cluster as your capacity needs increase.	• Conflicts are possible. • Transaction failures are possible. If one occurs, you must repair inconsistent data.

Use an RS secondary server in your environment for the following reasons:

- Increased server availability
  One or more RS secondary servers provide added assurance by maintaining multiple servers that can be used to increase availability.
- Geographically distant backup support

It is often desirable to have a secondary server located at some distance from the site for worst-case disaster recovery scenarios. An RS secondary server is an ideal remote backup solution. The high level of coordination between a primary and secondary HDR pair can cause performance issues if the secondary server is located on a WAN (Wide-Area Network). Keeping the primary and secondary servers relatively close together eases maintenance and minimizes the affect on performance.

- Improved reporting performance
  Multiple secondary servers can offload reporting function without affecting the primary server. Also, an RS secondary server configuration makes it easier to isolate reporting requirements from the HA requirements, resulting in better solutions for both environments.

- Availability over unstable networks
  A slow or unstable network environment can cause delays on both the primary and secondary server if checkpoints are achieved synchronously. RS secondary server configurations use fully duplexed networking and require no such coordination. An RS secondary server is an attractive solution if network performance between the primary server and RS secondary server is less than optimal.

Balance workload to optimize resource use

You can configure workload balancing when you create or modify a service level agreement SLA. Informix gathers information from each server in a cluster and automatically connects the client application to the server that has the least amount of activity.

You can create groups within a cluster that are specific to certain types of applications, such as those for online transaction processing (OLTP) or (warehouse). Applications can choose to connect to the specific group for optimized performance of each type of query.

Related information:
JSON data sharding
Shard cluster setup
Setting up and managing Enterprise Replication

High availability strategies

IBM® Informix® can be configured to maximize availability in various business situations.

Goal	Strategy	Advantages	Potential disadvantages
Protect system from server failure	Use a secondary server that shares disk space with the primary server. (shared-disk secondary server)	• Very high availability. This secondary server has access to the same data as the primary server. If the primary server fails, the secondary server can take over quickly. • The database is always in sync because this secondary server has access to the same data as the primary server. • No requirement to change applications. Client connections to primary or secondary server are automatically switched in the event of server failure.	• This secondary server on the same computer as the primary server. • No data redundancy. This secondary server does not maintain a copy of the data. (Use SAN devices for disk storage.) • Primary and secondary servers require the same hardware, operating system, and version of the database server product. • Secondary server hardware must be able to handle the same load as the primary server. If the secondary server is too small, it might affect the performance of the primary.
Protection from site failure	• Use a secondary server that maintains a copy of the database server and the data. (high availability data replication server) • (Can also use RSS and ER)	• Very high availability. Applications can access this server quickly if they cannot connect to a primary server. • Data is replicated synchronously. • Increased scalability • No requirement to change applications	• Local to the primary • Requires an exact replica of the data (including table and database schemas). • Primary and secondary servers require the same hardware, operating system, and version of the database server product.
Multilevel site failure protection	• Use a secondary server that is geographically distant from the primary server and that is updated asynchronously from the primary server. (remote stand-alone secondary server) • (Can also use ER)	• Very high availability. Applications can access this server quickly if they cannot connect to a primary server. • Data is replicated asynchronously. • Increased scalability • No requirement to change applications	
Geographically dispersed processing with site failure protection	ER and HDR	ER and backup for ER	Multiple connection managers required

High-availability cluster configuration

These topics describe how to plan, configure, start, and monitor high-availability clusters for IBM® Informix®, and how to restore data after a media failure. If you plan to use a high-availability cluster, read all the topics within this section first. If you plan to use IBM Informix Enterprise Replication, see the .

High availability and scalability, explains what a high-availability cluster is, how it works, and how to design client applications for a cluster environment.

- Plan for a high-availability cluster
- Configuring clusters
  Configure clusters by securing confirming the hardware, operating-system, and database requirements. You also set up a security protocol and the secure connection.
- Starting HDR for the First Time
- Remote standalone secondary servers
- Shared disk secondary servers

Plan for a high-availability cluster

Before you start setting up computers and database servers to use a high-availability cluster, you might want to do some initial planning. The following list contains planning tasks to perform:

- Choose and acquire appropriate hardware.
- If you are using more than one database server to store data that you want to replicate, migrate and redistribute data, so that it can be managed by a single database server.
- Ensure that all databases you want to replicate use transaction logging. To turn on transaction logging, see Manage the database-logging mode.
- Develop client applications to make use of both database servers in the replication pair. For an explanation of design considerations, see Redirection and connectivity for data-replication clients and Design data replication group clients.
- Create a schedule for starting HDR for the first time.
- Design a storage-space and logical-log backup schedule for the primary database server.
- Produce a plan for how to handle failures of either database server and how to restart HDR after a failure. Read Redirection and connectivity for data-replication clients.

Configuring clusters

Configure clusters by securing confirming the hardware, operating-system, and database requirements. You also set up a security protocol and the secure connection.

To configure your system as a high-availability cluster, you must take the following actions:

- Meet hardware and operating-system requirements.
- Meet database and data requirements.
- Meet database server configuration requirements.
- Configure connectivity.

Each of these topics are explained in this section.

You can configure your system to use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures the privacy and integrity of data transmitted over the network, for HDR communications. You can use the SSL protocol for connections between primary and secondary servers and for connections with remote standalone (RS) and shared disk (SD) secondary servers in a high-availability configuration. For information about using the SSL protocol, see Configuring server-to-server SSL connections.

The Connection Manager also supports Distributed Relational Database Architecture™ (DRDA) connections. For more information, see Distributed Relational Database Architecture (DRDA) communications.

- Hardware and operating-system requirements for clusters
  For a high-availability cluster to function, your hardware must meet certain requirements.
- Database and data requirements for clusters
  For a high-availability cluster to function, your database and data must meet certain requirements.
- Database server configuration requirements for clusters
- Configuring secure connections for high-availability clusters
  For a high-availability cluster to function, the database servers must establish trusted connection with each other. Secure connections between cluster servers by using a trusted-host file on each cluster server and including the connection security option in sqlhosts file entries.

Hardware and operating-system requirements for clusters

For a high-availability cluster to function, your hardware must meet certain requirements.

Your hardware must meet the following requirements:

- The primary and secondary servers must be able to run the same IBM® Informix® executable image, even if they do not have identical hardware or operating systems. For example, you can use servers with different Linux 32-bit operating systems because those operating systems can run the same executable image. In this situation, you cannot add a server on a Linux 64-bit operating system because that operating system requires a different executable image. Check the machine notes file: you can use any combination of hardware and operating systems listed as supported in the same machine notes file.
- The hardware that runs the primary and secondary database servers must support network capabilities.
- The amount of disk space allocated to dbspaces for the primary and secondary database servers must be equal. The type of disk space is irrelevant; you can use any mixture of raw or cooked spaces on the two database servers.
- The chunks on each computer must have the same path names. Symbolic links are allowed for UNIX platforms, but not for Windows platforms.

Database and data requirements for clusters

For a high-availability cluster to function, your database and data must meet certain requirements.

Your database and data must meet the following requirements:

- All data must be logged.
  All databases that you want to replicate must have transaction logging turned on.

  This requirement is important because the secondary database server uses logical-log records from the primary database server to update the data that it manages. If databases managed by the primary database server do not use logging, updates to those databases do not generate log records, so the secondary database server has no means of updating the replicated data. Logging can be buffered or unbuffered.

  If you must turn on transaction logging before you start HDR, see Turn on transaction logging with ontape.

- The data must be located in dbspaces or sbspaces.
  If your primary database server has simple large objects stored in blobspaces, modifications to the data within those blobspaces is not replicated as part of normal HDR processing. However, simple-large-object data within dbspaces is replicated.

  Smart large objects, which are stored in sbspaces, are replicated. The sbspaces must be logged. User-defined types (UDTs) are replicated, unless they have out-of-row data stored in operating system files. Data types with out-of-row data are replicated if the data is stored in an sbspace or in a different table on the same database server.

  You can encrypt storage spaces on high-availability servers. The encryption state of corresponding storage spaces on HDR and RS primary and secondary servers can be different. The encryption state of corresponding storage spaces on SD primary and secondary servers must be the same.

- The secondary servers must not use disk compression.
  If you use the IBM® Informix® disk compression feature, data that is compressed in the source table is compressed in the target table. You cannot perform compression operations on an HDR secondary, RS secondary, or SD secondary server, because the HDR target server must have the same data and physical layout as the source server.

Database server configuration requirements for clusters

For a high-availability cluster server pair to function, you must fully configure each of the database servers. For information about configuring a database server, see Overview of database server configuration and administration. You can then use the relevant aspects of that configuration to configure the other database server in the pair. For more information about the configuration parameters, see the *IBM Informix Administrator's Reference*.

These topics describe the following configuration considerations for cluster database server pairs:

- Database server version
- Storage space and chunk configuration
- Non-default page sizes in an HDR environment
- Mirroring
- Physical-log configuration
- Dbspace and logical-log tape backup devices
- Logical-log configuration
- High-availability cluster configuration parameters
- Cluster transaction coordination
  You can configure your high-availability cluster so that when a client session issues a commit, the server blocks the session until the transaction is applied in that session, on a secondary server, or across the cluster. Set the CLUSTER_TXN_SCOPE configuration parameter or run the `SET ENVIRONMENT CLUSTER_TXN_SCOPE` statement to configure this behavior.

Database server version

The versions of the database server on the primary and secondary database servers must be identical.

Storage space and chunk configuration

The number of dbspaces, the number of chunks, their sizes, their path names, and their offsets must be identical on the primary and secondary database servers. In addition, the configuration must contain at least one temporary dbspace if the HDR secondary server is used for creating activity reports. See Use of temporary dbspaces for sorting and temporary tables.

UNIX Only:

You must use symbolic links for the chunk path names, as explained in Allocating raw disk space on UNIX.

Important: If you do not use symbolic links for chunk path names, you cannot easily change the path name of a chunk. For more information, see Renaming chunks.

The following ONCONFIG parameters must have the same value on each database server:

- ROOTNAME
- ROOTOFFSET
- ROOTPATH
- ROOTSIZE

Non-default page sizes in an HDR environment

The page size of a dbspace and the buffer pool specifications are automatically propagated from the primary to the secondary database server. While both the primary and the secondary database servers must have the same buffer pools, the number of buffers in the buffer pools are not required to match.

Mirroring

You are not required to set the MIRROR parameter to the same value on the two database servers; you can enable mirroring on one database server and disable mirroring on the other. However, if you specify a mirror chunk for the root chunk of the primary database server, you must also specify a mirror chunk for the root chunk on the secondary database server. Therefore, the following ONCONFIG parameters must be set to the same value on both database servers:

- MIRROROFFSET
- MIRRORPATH

Physical-log configuration

The physical log must be identical on both database servers. The following ONCONFIG parameters must have the same value on each database server:

- PHYSBUFF
- PHYSFILE

Dbspace and logical-log tape backup devices

You can specify different tape devices for the primary and secondary database servers.

If you use ON-Bar, set the ON-Bar configuration parameters to the same value on both database servers. For information about the ON-Bar parameters, see the *IBM® Informix® Backup and Restore Guide*.

If you use **ontape**, the tape size and tape block size for the storage-space and logical-log backup devices must be identical. The following ONCONFIG parameters must have the same value on each database server:

- LTAPEBLK
- LTAPESIZE
- TAPEBLK
- TAPESIZE

To use a tape to its full physical capacity, set LTAPESIZE and TAPESIZE to `0`.

Logical-log configuration

All log records are replicated to the secondary server. You must configure the same number of logical-log files and the same logical-log size for both database servers. The following ONCONFIG parameters must have the same value on each database server:

- LOGBUFF
- LOGFILES
- LOGSIZE
- DYNAMIC_LOGS

The database server logs the addition of logical-log files. Logical-log files added dynamically on the primary server are automatically replicated on the secondary server. Although the DYNAMIC_LOGS value on the secondary server has no effect, keep DYNAMIC_LOGS in sync with the value on the primary server, in case their roles switch.

High-availability cluster configuration parameters

Set the HA_ALIAS configuration parameter on each database server that participates in a high-availability cluster.

Set TEMPTAB_NOLOG configuration parameter to 1 or 2 on each secondary server in a high-availability cluster. Secondary servers in a high-availability cluster must disable logical logging on temporary tables.

Set the following configuration parameters to the same value on both database servers in a HDR pair:

- DRAUTO
- DRINTERVAL
- DRTIMEOUT

Related information:
TEMPTAB_NOLOG configuration parameter

Cluster transaction coordination

You can configure your high-availability cluster so that when a client session issues a commit, the server blocks the session until the transaction is applied in that session, on a secondary server, or across the cluster. Set the CLUSTER_TXN_SCOPE configuration parameter or run the `SET ENVIRONMENT CLUSTER_TXN_SCOPE` statement to configure this behavior.

Multistep client operations that are performed on different high-availability cluster servers or in different sessions with high-availability cluster servers can fail because of asynchronous log processing. If a client application loads data onto a cluster server, and then attempts to process the same data on a second cluster server before the data is replicated to the second server, the operation fails. The client transaction must be applied on the second server before its data can be further processed.

Cluster transaction coordination causes client applications to wait for either cluster-wide or secondary-server application of transactions before the transaction commits are returned. This process prevents operation failures and ensures that the steps of multistep processes occur in serial order.

The different scopes for cluster transaction are:

- SESSION: When a client session issues a commit, the database server blocks the session until the effects of the transaction commit are returned to that session. After control is returned to the session, other sessions at the same database server or on other database servers in the cluster might be unaware of the transaction commit and the transaction's effects.
- SERVER: When a client session issues a commit, the database server blocks the session until the transaction is applied at the database server from which the client session issued the commit. Other sessions at that database server are aware of the transaction commit and the transaction's effects. Sessions at other database servers in the cluster might be unaware of the transaction's commit and its effects. This behavior is default for high-availability cluster servers.
- CLUSTER: When a client session issues a commit, the database server blocks the session until the transaction is applied at all database servers in the high-availability cluster, excluding RS secondary servers that are using DELAY_APPLY or STOP_APPLY. Other sessions at any database server in the high-availability cluster, excluding RS secondary servers that are using DELAY_APPLY or STOP_APPLY, are aware of the transaction commit and the transaction's effects.

Cluster transaction coordination was introduced in IBM® Informix® version 11.70.xC6. Before IBM Informix version 11.70.xC6, high-availability cluster servers had the following default behaviors:

- Primary servers had a cluster transaction scope of SERVER.
- Read-only secondary servers were in the dirty-read isolation level, and could read uncommitted data.
- Updatable secondary servers had a cluster transaction scope of SESSION.

Setting a CLUSTER_TXN_SCOPE value to CLUSTER does not change the behavior that is specified by the DRINTERVAL configuration parameter value. When a client application commits a transaction on a primary server, the primary server sends the HDR secondary server logical log buffers at maximum intervals that are specified by the DRINTERVAL configuration parameter. After the primary server sends logical log buffers to the HDR secondary server, it returns control to a session, but the session still does not receive a commit until the transaction is applied on all cluster servers.

Related information:
CLUSTER_TXN_SCOPE configuration parameter
DELAY_APPLY Configuration Parameter
STOP_APPLY configuration parameter
SET ENVIRONMENT statement

Configuring secure connections for high-availability clusters

For a high-availability cluster to function, the database servers must establish trusted connection with each other. Secure connections between cluster servers by using a trusted-host file on each cluster server and including the connection security option in sqlhosts file entries.

The secure ports that are specified in sqlhosts files are used only for communication between database servers. Client applications cannot connect to secure ports.
To configure a trusted environment for replication, complete the following steps for each cluster server:

1. Edit the sqlhosts file on each host that contains a cluster server:
   a. Add an entry for each cluster server that is running on that host, and include the **s=6** option.
   b. Add an entry for each other cluster server that participates in the cluster, and do not include the **s=6** option.
2. Set the `nettype` field of the sqlhosts file or registry and the NETTYPE configuration parameter to a network protocol such as ontlitcp or onsoctcp so that the database servers on two different computers can communicate with each other. Do not specify a non-network protocol such as onipcshm, onipcstr, or onipcnmp.
3. Specify trusted-host information. Trusted-host information can be specified in the following ways:
   - Create a hosts.equiv file in the $INFORMIXDIR/etc directory, and then manually add entries to the file.
   - Create a trusted-host file in the $INFORMIXDIR/etc directory, and then manually add entries to the file. You must set the REMOTE_SERVER_CFG configuration parameter to the trusted-host file's name and set the S6_USE_REMOTE_SERVER_CFG configuration parameter to `1`.
   - Run the **admin()** or **task()** function with the cdr add trustedhost argument, and specify trusted-host information. Trusted-host information that is specified by the cdr add trustedhost argument propagates to all servers in the high-availability cluster. Do not run this function if you have manually entered trusted-host information on any of the database servers in a high-availability cluster or Enterprise Replication domain.
4. Create a server alias for running utilities and client applications. For example, set the INFORMIXSERVER environment variable to the alias to run utilities such as **onstat** and **ontape** and client applications such as DB-Access.

Related concepts:
Trusted-host information
Related tasks:
Changing client connectivity information
Related reference:
sqlhosts connectivity information
sqlhosts file and SQLHOSTS registry key options
Related information:
S6_USE_REMOTE_SERVER_CFG configuration parameter
REMOTE_SERVER_CFG configuration parameter

Starting HDR for the First Time

After you complete the HDR configuration, you are ready to start HDR. This topic describes the necessary steps for starting HDR.

You want to start HDR on two database servers, **ServerA** and **ServerB**. The procedure for starting HDR, using **ServerA** as the primary database server and **ServerB** as the secondary database server, is described in the following steps. The following table lists the commands required to perform each step and the messages sent to the message log. You can use **ontape** or ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure.
Important: Even if you use ON-Bar to perform the backup and restore, the **ontape** utility is still required on both database servers to perform back ups and to apply logical logs. Do not remove the **ontape** utility from database servers that participate in an HDR cluster environment.
You can also set up HDR using external backup and restore. See the *IBM® Informix® Backup and Restore Guide* for information about how to perform an external backup and restore. See Decrease setup time using the ontape STDIO feature for the quickest way to set up your HDR secondary directly from the HDR primary.
To start HDR:

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.
2. Create a level-0 backup of **ServerA**.
3. Use the **onmode -d** command to set the type of **ServerA** to primary and to indicate the name of the associated secondary database server (in this case **ServerB**). When you issue an **onmode -d** command, the database server attempts to establish a connection with the other database server in the HDR pair and to start HDR operation. The attempt to establish a connection succeeds only if the other database server in the pair is already set to the correct type.

   At this point, **ServerB** is not online and is not set to type secondary, so the HDR connection is not established.

4. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 1. Do not perform a logical restore.
   If you are using:
   - ON-Bar, use the **onbar -r -p** command to perform a physical restore.
   - ON-Bar and performing an external restore, use the **onbar -r -p -e** command to perform the physical restore.
   - **ontape**, use the **ontape -p** option. You cannot use the **ontape -r** option because it performs both a physical and a logical restore.
     Note: You must place the physical restore of your primary server on the secondary server if they are on two different machines. The location of the physical restore is defined by the onconfig parameter TAPE. You must set your *IFX_ONTAPE_FILE_PREFIX* variable on your secondary server before you can run **ontape -p**.
   - **ontape** and performing an external restore, use the **ontape -p -e** command to perform the physical restore.
5. Use the **onmode -d** command to set the type of **ServerB** to secondary and indicate the associated primary database server.
   ServerB tries to establish an HDR connection with the primary database server (**ServerA**) and start operation. The connection must be successfully established.

   Before HDR begins, the secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 2. If all these logical-log records still are located on the primary database server disk, the primary database server sends these records directly to the secondary database server over the network and logical recovery occurs automatically.

   If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 6.

Important: You must complete steps 4 to 5 during the same session. If you must shut down and restart the secondary database server after step 4, you must redo step 4.

6. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.
   If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

   After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

The following table illustrates the preceding steps so that you can clearly determine which steps are performed on the primary server and which are performed on the secondary server. The table also shows information written to the log file after each step is performed.

Table 1. Steps to start HDR for the first time

Step	On the primary (ServerA)	On the secondary (ServerB)
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**ontape** command: Run **ontape -s -L 0** ON-Bar command :Run **onbar -b -L 0** Messages to message log: `Level 0 archive started on rootdbs` `Archive on rootdbs completed`	
3.	**onmode** command **onmode -d primary** *sec_name* Messages to message log: `DR: new type = primary` `server name = sec_name` `DR: Trying to connect to secondary server` `DR: Cannot connect to secondary server`	
4.		**ontape** command Run **ontape -p** or **ontape -p -e** Answer `no` when you are prompted to back up the logs. ON-Bar command Run **onbar -r -p** or **onbar -r -p -e** Messages to message log: `IBM Informix Database Server Initialized --` `Shared Memory Initialized` `Recovery Mode` `Physical restore of rootdbs started` `Physical restore of rootdbs completed`
5.		Run **onmode -d secondary** *prim_name* Messages to message log: `DR: new type = secondary, primary server` `name = prim_name` If all the logical-log records written to the primary database server since step 1 still are located on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 6 must be performed).
	Messages to message log: `DR: Primary server connected` `DR: Primary server operational`	Messages to message log: `DR: Trying to connect to primary server` `DR: Secondary server connected` `DR: Failure recovery from disk in progress` `n recovery worker threads will be started` `Logical Recovery Started` `Start Logical Recovery - Start Log n, End Log ?` `Starting Log Position - n 0xnnnnn` `DR: Secondary server operational`
6.		**ontape** command **ontape -l** ON-Bar command **onbar -r -l**

Step	On the primary (ServerA)	On the secondary (ServerB)
	Messages to message log: `DR: Primary server connected` `DR: Primary server operational`	Messages to message log: `DR: Secondary server connected` `DR: Failure recovery from disk in progress` `n recovery worker threads will be started` `Logical Recovery Started` `Start Logical Recovery - Start Log n, End Log ?` `Starting Log Position - n 0xnnnnnn` `DR: Secondary server operational`

- Decrease setup time using the ontape STDIO feature

Related concepts:
Backup and restore with high-availability clusters
Related reference:
Recovering a cluster after critical data is damaged

Decrease setup time using the ontape STDIO feature

You can dramatically improve the speed of setting up HDR by using the **ontape** `STDIO` feature. Using this feature, **ontape** writes the data to the shell's standard output during a backup, and then read it from standard input during a restore. Combining a `STDIO` backup with a simultaneous `STDIO` restore in a pipe using a remote command interpreter (such as rsh or ssh), allows performing the initial setup of an HDR (or RSS) secondary server using a single command line. This saves storage space by not writing to or reading from tape or disk, and does not require waiting for the backup to finish before the restore can start.

See the *IBM® Informix Backup and Restore Guide* for details about using the `STDIO` value.

This method for setting up HDR using **ontape** can be used regardless of which backup utility is used (**ontape** or ON-Bar).

Important: When you use STDIO in this way, no persistent backup is saved anywhere that can be used to perform a restore. The use of the `-F` (fake) option on the source (backup) side does not record the backup in the database server's reserved pages. Also, any interactive dialog is suppressed and no prompts or questions are displayed. You must also ensure that the remote part of the pipe picks the appropriate environment for the remote Informix® instance. The script must not produce any output other than the backup data since this would be read by the restore process (for example, do not enable tracing).
The steps in the following table must be performed by user **informix**, the scripts must be executable, and, if called without a complete path, must be located in your home directory. You can use **ssh** instead of **rsh** if you require secure data transmission across the network.

1. On the secondary server, install UDRs, UDTs, and DataBlade modules.
2. On the primary server, install UDRs, UDTs, and DataBlade modules.
3. On the primary server, register UDRs, UDTs, and modules.
4. On the primary server, run the following command:

   ```
   onmode -d primary secondary_server
   ```

5. On the primary server, run the following command:

   ```
   ontape -s -L 0 -t STDIO -F | rsh secondary_host ontape_HDR_restore.ksh
   ```

6. On the secondary server, run the following command:

   ```
   onmode -d secondary primary_server
   ```

In the previous table, the script ontape_HDR_restore.ksh on the secondary server must contain the following commands:

```
#!/bin/ksh
# first get the proper Informix environment set
. hdr_sec.env
# redirecting stdout and stderr required since otherwise command might never return
ontape -p -t STDIO > /dev/null 2>&1
```

The following steps show how to set up HDR from the secondary server:

1. On the secondary server, install UDRs, UDTs, and DataBlade modules.
2. On the primary server, install UDRs, UDTs, and DataBlade modules.
3. On the primary server, register UDRs, UDTs, and DataBlade modules.
4. On the primary server, run the following command:

   ```
   onmode -d primary secondary_server
   ```

5. On the secondary server, run the following command:

   ```
   rsh primary_host ontape_HDR_backup.ksh | ontape -p -t STDIO
   ```

6. On the secondary server, run the following command:

   ```
   onmode -d secondary primary_server
   ```

In the previous table, the script ontape_HDR_backup.ksh on the primary server must contain the following commands:

```
#!/bin/ksh
# first get the proper Informix environment set
. hdr_pri.env
ontape -s -L 0 -F -t STDIO
```

Remote standalone secondary servers

These topics provide an overview of setting up and configuring remote standalone (RS) secondary servers in a high availability environment.

- Comparison of RS secondary servers and HDR secondary servers
- Index page logging
- Starting an RS secondary server for the first time
  After you complete the hardware configuration of the RS secondary server, you are ready to start the RS secondary server and connect it to the primary server.
- Converting an offline primary server to an RS secondary server
  After a planned or unplanned failover of the primary server to an RS secondary server, you can convert the old primary server to an RS secondary server.
- Delayed application of log records
  To aid in disaster recovery scenarios, you can configure RS secondary servers to wait for a specified period of time before applying logs received from the primary server.
- Flow control for remote standalone secondary servers
  Flow control provides a way to limit log activity on the primary server so that remote standalone (RS) secondary servers in the cluster do not fall too far behind on processing transactions. Enabling flow control ensures that logs on RS secondary servers remain current if the servers are on a busy or intermittent network.

Comparison of RS secondary servers and HDR secondary servers

An RS secondary server is similar in many ways to an HDR secondary server. Logs are sent to the RS secondary server in much the same way as a primary server sends logs to an HDR secondary server. However, the RS secondary server is designed to function entirely within an asynchronous communication framework so that its effect on the primary server is minimized. Neither transaction commits nor checkpoints are synchronized between the primary server and RS secondary servers. Any transaction committed on the primary server is not guaranteed to be committed at the same time on the RS secondary server.

In a high-availability cluster, the log of the HDR secondary server must be ahead of the logs of any RS secondary servers. If the HDR secondary server becomes offline, the primary server continues to send logs to the RS secondary servers. However, when the HDR secondary comes back online, IBM® Informix® stops sending logs to RS secondary servers and prioritizes sending logs to the HDR secondary server until its log replay is ahead of the RS secondary server. This prioritization of the HDR secondary server logs is required because the HDR secondary server is the first failover choice in the cluster. If the RS secondary server logs are ahead of the HDR secondary server logs when a failover occurs, then the RS secondary server cannot synchronize with the new primary server.

While an RS secondary server is similar to an HDR secondary server, there are several things that an HDR secondary server supports that an RS secondary server does not support:

- SYNC mode
- DRAUTO parameter
- Synchronized checkpoints

For HDR, RSS, and SDS secondary servers in a high-availability cluster, logical logging on temporary tables must always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1 or 2.

Related information:
TEMPTAB_NOLOG configuration parameter

Index page logging

You must enable index page logging to use an RS secondary server.

- How index page logging works
- Enable or disable index page logging
- View index page logging statistics

Related concepts:
Replication of primary-server data to secondary servers

How index page logging works

When an index is created, index page logging writes the pages to the logical log for the purpose of synchronizing index creation between servers in high-availability environments.

Index page logging writes the full index to the log file, which is then transmitted asynchronously to the secondary server. The secondary server can be either an RS secondary or an HDR secondary server. The log file transactions are then read into the database on the secondary server. The secondary server is not required to rebuild the index during recovery. For RS secondary servers, the primary server does not wait for an acknowledgment from the secondary server, which allows immediate access to the index on the primary server.

Control index page logging with the ONCONFIG parameter LOG_INDEX_BUILDS. Set the LOG_INDEX_BUILDS parameter to `1` (enabled), to build indexes on the primary server and send them to the secondary server.

Enable or disable index page logging

Use the LOG_INDEX_BUILDS configuration parameter to enable or disable index page logging when the database server starts. You can change the value of LOG_INDEX_BUILDS in the onconfig file by running **onmode -wf LOG_INDEX_BUILDS=1** (enable) or **0** (disable).

Index page logging must be enabled when an RS secondary server exists in a high-availability environment.

View index page logging statistics

You can use the **onstat** utility or system-monitoring interface (SMI) tables to view whether index page logging is enabled or disabled. The statistics also display the date and time index page logging was enabled or disabled.

To view index page logging statistics, use the **onstat -g ipl** command, or query the **sysipl** table.

For an example of **onstat -g ipl** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

Starting an RS secondary server for the first time

After you complete the hardware configuration of the RS secondary server, you are ready to start the RS secondary server and connect it to the primary server.

Suppose you want to start a primary server and an RS secondary server, **ServerA** and **ServerB**. The procedure for starting the servers, using **ServerA** as the primary database server and **ServerB** as the RS secondary database server, is described in the following steps. The table ahead lists the commands required to perform each step.

The procedure requires that the primary server be backed up and then restored onto the secondary server. You can use **ontape** or ON-Bar to perform the backup and restore. You must use the same utility throughout the procedure.
Important: Even if you use ON-Bar to perform the backup and restore, the **ontape** utility is still required on both database servers to perform back ups and to apply logical logs. Do not remove the **ontape** utility from database servers that participate in an HDR cluster environment.
You can also set up an RS secondary server using standard ON-Bar or **ontape** commands for external backup and restore.
To start a primary server with an RS secondary server:

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers, and then register them on **ServerA** only.
   For information about how to install user-defined types or user-defined routines see the *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.
   For information about how to install DataBlade modules, see the *IBM Informix DataBlade Module Installation and Registration Guide*.
2. Activate index page logging on the primary server.
3. Record the identity of the RS secondary server on the primary server. The optional password provides authentication between the primary and RS secondary server when the connection between the primary and secondary servers is established for the first time.
4. Create a level-0 backup of **ServerA**.
5. Perform a physical restore of **ServerB** from the level-0 backup that you created in step 4. Do not perform a logical restore.
   Use the appropriate command:
   - Use the **onbar -r -p** command to perform a physical restore.
   - Use the **onbar -r -p -e** command to perform a physical external restore.
   - Use the **ontape -p** option. (Do not use the **ontape -r** option because it performs both a physical and a logical restore.)
   - Use the **ontape -p -e** command to perform the physical external restore.
6. Use the **onmode -d RSS ServerA** *password* command to set the type of **ServerB** to an RS secondary server and indicate the associated primary database server.
   For this example, **ServerA**'s DBSERVERNAME and HA_ALIAS configuration parameters are both set to `ServerA`, and **ServerB**'s DBSERVERNAME and HA_ALIAS configuration parameters are both set to `ServerB`.
   Using the database server's HA_ALIAS configuration parameter value, **ServerB** tries to establish a connection with the primary database server (**ServerA**) and start operation. The connection must be successfully established.

   The secondary database server performs a logical recovery using the logical-log records written to the primary database server since step 4. If all these logical-log records are still located on the primary database server disk, the primary database server sends these records directly to the RS secondary server over the network and logical recovery occurs automatically.

   If you have backed up and freed logical-log files on the primary database server, the records in these files are no longer on disk. The secondary database server prompts you to recover these files from tape. In this case, you must perform step 7.
   Important: You must complete steps 5 through 6 during the same session. If you must shut down and restart the secondary database server after step 5, you must redo step 5.
7. If logical-log records that were written to the primary database server are no longer on the primary disk, the secondary database server prompts you to recover these files from tape backups.
   If the secondary database server must read the backed-up logical-log files over the network, set the tape device parameters on the secondary database server to a device on the computer that is running the primary database server or to a device at the same location as the primary database server.

   After you recover all the logical-log files on tape, the logical restore completes using the logical-log files on the primary database server disk.

Table 1. Steps to start a primary with an RS secondary server for the first time

Step	On the primary	On the RS secondary
1.	Install UDRs, UDTs, and DataBlade modules. Register UDRs, UDTs, and DataBlade modules.	Install UDRs, UDTs, and DataBlade modules.
2.	**onmode** command **onmode -wf LOG_INDEX_BUILDS=1**	
3.	**onmode** command **onmode -d add RSS** *rss_ha_alias password*	
4.	**ontape** command **ontape -s -L 0** ON-Bar command **onbar -b -L 0**	
5.		**ontape** command **ontape -p** or **ontape -p -e** Answer `no` when you are prompted to back up the logs. ON-Bar command **onbar -r -p** or **onbar -r -p -e**
6.		**onmode** command **onmode -d RSS** *primary_ha_alias password* If all the logical-log records written to the primary database server since step 1 still are located on the primary database server disk, the secondary database server reads these records to perform logical recovery. (Otherwise, step 7 must be performed).
7.		**ontape** command **ontape -l** ON-Bar command **onbar -r -l** This step is required only when the secondary database server prompts you to recover the logical-log files from the tape backup.

- Decrease setup time through an alternative backup method

Related concepts:
Backup and restore with high-availability clusters

Decrease setup time through an alternative backup method

You can dramatically improve the speed of setting up a secondary server by using the **ontape** `STDIO` feature. See Decrease setup time using the ontape STDIO feature for more information.

See the *IBM® Informix® Backup and Restore Guide* for details about using the `STDIO` value.

Converting an offline primary server to an RS secondary server

After a planned or unplanned failover of the primary server to an RS secondary server, you can convert the old primary server to an RS secondary server.

For example, assume you have a primary server named srv1 that has failed over to an RS secondary server named srv2. The following steps show how to convert the old primary server to an RS secondary server.

1. On the new primary server (srv2) register the old primary server (srv1 as the RS secondary server.

   ```
   onmode -d add RSS srv1
   ```

2. If you are converting the old primary server to an RS secondary server and the server is offline, then initialize the server using backup and restore commands shown here: Starting an RS secondary server for the first time. Alternatively you can initialize the old primary server by running the following command:

   ```
   oninit -PHY
   ```

   See The oninit utility for more information.

3. Convert the server to an RS secondary server using the following commands:

   ```
   onmode -d RSS srv2
   ```

Delayed application of log records

To aid in disaster recovery scenarios, you can configure RS secondary servers to wait for a specified period of time before applying logs received from the primary server.

By delaying the application of log files you can recover quickly from erroneous database modifications by restoring the database from the RS secondary server. You can also stop the application of logs on an RS secondary server at a specified time.

For example, suppose a database administrator wants to delete certain rows from a table based on the age of the row. Each row in the table contains a timestamp that indicates when the row was created. If the database administrator inadvertently sets the filter to the wrong date, more rows than intended might be deleted. By delaying the application of log files, the rows would still exist on the RS secondary server. The database administrator can then extract the rows from the secondary server and insert them on the primary server.

Now suppose a database administrator is required to perform changes to the schema by renaming a table, but types the wrong command and drops the table **orders** instead of changing the table name to **store_orders**. If an RS secondary server is configured to delay application of logs, the database administrator can recover the **orders** table from the secondary server.

When delayed application of log files configured, transactions sent from the primary server are not applied until after a specified period of time has elapsed. Log files received from the primary server are staged in a specified secure directory on the RS secondary server, and then applied after the specified period of time. There are two ways to delay the application of log files:

- Apply the staged log files after a specified time interval
- Stop applying log files at a specified time

You enable the delayed application of log files by setting configuration parameters in the onconfig file of the RS secondary server. You must specify the directory in which log files are staged by setting the LOG_STAGING_DIR configuration parameter before enabling the delayed application of log files. After specifying the LOG_STAGING_DIR configuration parameter, you configure the DELAY_APPLY or STOP_APPLY configuration parameters either by editing the onconfig file or dynamically using **onmode -wf** commands.

Where log records are stored

The server creates additional directories named ifmxlog_## in the directory specified by LOG_STAGING_DIR, where ## is the instance specified by SERVERNUM. The directories are used to store the logical logs and are also used during the recovery of the RS secondary server. If recovery of the RS secondary server becomes necessary, and the logs have wrapped on the primary server, then the logs in ifmxlog_## can be used to recover the server. The files within ifmxlog_## are purged when no longer required.

Conditions that trigger delays

The time values in the BEGIN WORK, COMMIT WORK, and ROLLBACK WORK log records are used to calculate how to delay or stop the application of log files. The time values are calculated before passing the log pages to the recovery process.

When a BEGIN WORK statement is issued, the BEGIN WORK log record is not written until the first update activity is performed by the transaction; therefore, there can be a delay between the time that the BEGIN WORK statement is issued and when the BEGIN WORK log is written.

Interaction with secondary server updates

You must consider the interaction between secondary server updates and delayed application of log files. If updates are enabled, and the secondary server is updated, the updates are not applied until after the amount of time specified by DELAY_APPLY. Disabling secondary server updates, however, also disables Committed Read, which guarantees that every retrieved row is committed in the table at the time that the row is retrieved.

To retain the Committed Read isolation level, consider enabling secondary server updates using the UPDATABLE_SECONDARY configuration parameter, but removing the RS secondary server used for delayed application of log files from the Connection Manager service-level agreement list. Alternatively, consider moving the RS secondary server to a new SLA.

See Database updates on secondary servers and *IBM® Informix® Administrator's Reference* for more information.

- Specifying the log staging directory
  You configure the log staging directory to specify where log files on RS secondary servers are staged before being applied to the database.
- Delay application of log records on an RS secondary server
  You can delay the application of log records on an RS secondary server to prepare for disaster recovery scenarios.
- Stop the application of log records
  You can halt the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

Specifying the log staging directory

You configure the log staging directory to specify where log files on RS secondary servers are staged before being applied to the database.

You must specify a staging directory for log files sent from the primary server before enabling delayed application of log files. No default staging directory is defined. The server creates additional directories in the directory specified by LOG_STAGING_DIR named ifmxlog_##, where ## is the instance specified by SERVERNUM. The directories are used to store the logical logs and are also used during the recovery of the RS secondary server. The staged log files are automatically removed when they are no longer required. If the files within LOG_STAGING_DIR are lost, and the primary server has overwritten the logs, then the RS secondary server must be rebuilt.

You must ensure that the directory specified by LOG_STAGING_DIR exists and is secure. The directory must be owned by user **informix**, must belong to group **informix**, and must not have public read, write, or execute permission. If role separation is enabled, the directory specified by LOG_STAGING_DIR must be owned by the user or group that owns $INFORMIXDIR/etc. If the directory specified by LOG_STAGING_DIR is not secure, then the server cannot be initialized. The following message is written to the online message log if the directory is not secure:

```
The log staging directory (directory_name) is not secure.
```

You must also ensure that the disk contains sufficient space to hold all of the logs from the primary server, and that the directory does not contain staged logs from previous instances that are no longer being used.
To see information about the data being sent to the log-staging directory set for a RS secondary server, run the **onstat -g rss verbose** command on the RS secondary server.

If the write to the staging file fails, the RS secondary server raises event alarm 40007.

See *IBM® Informix® Administrator's Reference* for more information.

To set LOG_STAGING_DIR:

1. Ensure that the directory in which logs are to be stored exists and is secure.
2. Edit the RS secondary server onconfig file.
3. Specify the staging directory as follows: `LOG_STAGING_DIR directory_name` where *directory_name* is the name of the directory in which to store the logs.
4. Restart the server.

You can also set the LOG_STAGING_DIR configuration parameter without restarting the server by using the **onmode -wf** command; however, the delayed application of log files must not be active when the command is run.

Delay application of log records on an RS secondary server

You can delay the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

You enable the delayed application of log files by setting the DELAY_APPLY configuration parameter. You can manually edit the onconfig file and restart the server, or you can change the value dynamically using the **onmode -wf** command. When setting the value of DELAY_APPLY you must also set LOG_STAGING_DIR. If DELAY_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

Set DELAY_APPLY using both a *number* and a *modifier*. Number can contain up to three digits and indicates the number of modifier units. Modifier is one of:

- D (or d) for days
- H (or h) for hours
- M (or m) for minutes
- S (or s) for seconds

See *IBM® Informix® Administrator's Reference* for more information.

To delay the application of log files on the RS secondary for four hours:

```
onmode -wf DELAY_APPLY=4H
```

To delay the application of log files for one day:

```
onmode -wf DELAY_APPLY=1D
```

To disable delayed application of log files:

```
onmode -wf DELAY_APPLY=0
```


Stop the application of log records

You can halt the application of log records on an RS secondary server to prepare for disaster recovery scenarios.

You stop the application of log files on the RS secondary server by setting the STOP_APPLY configuration parameter. You can manually edit the onconfig file and restart the server, or you can change the value dynamically using the **onmode -wf** command. When setting the value of STOP_APPLY you must also set LOG_STAGING_DIR. If STOP_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

See *IBM® Informix® Administrator's Reference* for more information.

To stop the application of log files on the RS secondary server immediately, run the following command:

```
onmode -wf STOP_APPLY=1
```

To stop the application of log files at 11:00 p.m. on April 15th, 2009:

```
onmode -wf STOP_APPLY=2009:04:15-23:00:00
```

To resume the normal application of log files

```
onmode -wf STOP_APPLY=0
```

Flow control for remote standalone secondary servers

Flow control provides a way to limit log activity on the primary server so that remote standalone (RS) secondary servers in the cluster do not fall too far behind on processing transactions. Enabling flow control ensures that logs on RS secondary servers remain current if the servers are on a busy or intermittent network.

Set the RSS_FLOW_CONTROL configuration parameter on the primary server to enable flow control. All RS secondary servers in the cluster are affected by the primary server's RSS_FLOW_CONTROL configuration parameter setting. When flow control is active, users connected to the primary server may experience slower response time.

Logs are always sent to the RS secondary server in the order in which they were received.

To check if flow control is active for a RS secondary server, use the **onstat -g rss verbose** command, and compare the `RSS flow control` value to the `Approximate Log Page Backlog` value. If the `Approximate Log Page Backlog` is higher than the first value of `RSS flow control`, flow control is active. If the `Approximate Log Page Backlog` is lower than the second value of `RSS flow control`, flow control is disabled.

Related information:
RSS_FLOW_CONTROL configuration parameter
SDS_FLOW_CONTROL configuration parameter

Shared disk secondary servers

These topics provide an overview of setting up and configuring SD (shared disk) secondary servers in a high-availability environment. SD secondary server options are available with the standard version of IBM® Informix®.

- SD secondary server
  A shared-disk (SD) secondary server participates in high-availability cluster configurations. In such configurations, the primary server and the SD secondary server share the same disk or disk array.
- Disk requirements for SD secondary servers
- Setting up a shared disk secondary server
  You set up a shared disk system by configuring the primary server, configuring the SD secondary server, and starting the SD secondary.
- Obtain SD secondary server statistics
- Promote an SD secondary server to a primary server
- Convert a primary server to a standard server
- SD secondary server security
- Flow control for shared-disk secondary servers
  Flow control provides a way to limit log activity on the primary server so that shared-disk (SD) secondary servers in the cluster do not fall too far behind on processing transactions.

SD secondary server

A shared-disk (SD) secondary server participates in high-availability cluster configurations. In such configurations, the primary server and the SD secondary server share the same disk or disk array.

An SD secondary server does not maintain a copy of the physical database on its own disk space. Rather, it shares disks with the primary server.

SD secondary servers must be configured to access shared disk devices that allow concurrent access. Do not configure an SD secondary server that uses operating system buffering, such as NFS cross-mounted file systems. If the SD secondary server instance and the primary server instance both are located on a single machine, then both servers can access local disks. If the SD secondary server and the primary server are on separate physical machines, then they must be configured to access shared disk devices that appear locally attached, such as Veritas or GPFS™.

SD secondary servers can be used in conjunction with HDR secondary servers, with RS secondary servers, and with Enterprise Replication.

SD secondary servers can be added to a high availability environment very quickly, because they do not require a separate copy of the disk. Because the SD server shares the disk storage resources of the primary server, it is recommended that you provide some other means of disk backup, such as disk mirroring, or the use of an RS secondary server or an HDR secondary server.

The following restrictions affect the promotion of database server instances that are shared-disk secondary servers:

- An SD secondary server cannot be promoted to an RS secondary server.
- An SD secondary server cannot be promoted to a standard server that would exist outside the primary high availability environment.

Disk requirements for SD secondary servers

Except for disk requirements (which are shared with the primary server), hardware and software requirements are generally the same as for HDR secondary servers (See the Machine Notes for specific supported platforms). In addition, the primary disk system must be shared across the computers that are hosting the database servers. This means that the path to the dbspaces from the SD secondary is the same dbspace path as the primary server. see Configuring clusters.

Setting up a shared disk secondary server

You set up a shared disk system by configuring the primary server, configuring the SD secondary server, and starting the SD secondary.

Related information:
Database configuration parameters

Setting up the SD primary server

To set up the primary server:

1. Set the SDS_TIMEOUT configuration parameter to specify the amount of time in seconds that the primary server waits for a log position acknowledgment to be sent from the SD secondary server.
2. Configure the alias name of the SD primary server by running the following command:

   ```
   onmode -d set SDS primary ha_alias
   ```

   The server name that is specified by *ha_alias* becomes the primary server of the shared disk environment and the source of logs for the SD secondary server.

Setting up the SD secondary server

To set up the SD secondary server:

1. Set the following configuration parameters in the configuration file.
   - HA_ALIAS: set to define an alias for server-to-server communication in a high-availability cluster.
   - SDS_ENABLE: set to 1 (enable) on the secondary server to enable the shared disk environment.
   - SDS_PAGING: set to the path to two files that are used to hold pages that might be required to be flushed between checkpoints. Each file acts as temporary disk storage for chunks of any page size.
   - SDS_TEMPDBS: set to the temporary dbspace for the SD secondary server that is dynamically created when the server is first started.
   - SDS_LOGCHECK: set to the number of seconds to delay a failover if network communications between the primary and secondary servers is temporarily unavailable.
   - TEMPTAB_NOLOG: set to 1 or 2 to prevent logical logging on temporary tables.
   - UPDATABLE_SECONDARY: set to a positive integer if you want to enable client applications to perform update, insert, and delete operations on the secondary server.
2. Set the following configuration parameters to match those on the primary server:
   - ROOTNAME
   - ROOTPATH
   - ROOTOFFSET
   - ROOTSIZE
   - PHYSFILE
   - LOGFILES
   - LOGSIZE
   - DISK_ENCRYPTION

   You can set other configuration parameters to match those of the primary server except for DBSERVERALIASES, DBSERVERNAME, and SERVERNUM.
3. Add an entry to the sqlhosts file to identify the primary server:

   ```
   #dbservername   nettype   host   servicename   options
    name           protocol  name   port
   ```

4. Start the SD secondary server using the **oninit** command.
   The SD secondary server processes any open transactions as a fast recovery in quiescent mode. There is increased memory usage in the LGR memory pool during fast recovery.
5. Examine the online.log file on the secondary server to verify that it completed processing open transactions and is in online mode.
6. Allow client applications to connect to the SD secondary server.

Obtain SD secondary server statistics

Use the **onstat** utility or system-monitoring interface (SMI) tables to view SD secondary server statistics.

Use **onstat -g sds** to view SD secondary server statistics. The output of the **onstat** utility depends on whether the utility is run on the primary or secondary server.

Query the **syssrcsds** table to obtain information about shared disk statistics on the primary server.

Query the **systrgsds** table to obtain information about shared disk statistics on the secondary server.

For information about **onstat** and SMI tables see the *IBM® Informix® Administrator's Reference*.

Promote an SD secondary server to a primary server

Convert an SD secondary server to a primary server by issuing the following command on the SD secondary server:

```
onmode -d set SDS primary <alias>
```

An SD secondary server cannot be converted to a standard server.

Convert a primary server to a standard server

You can convert a primary server to a standard server and disconnect it from the shared disk environment using the following command on the primary server:

```
onmode -d clear SDS primary <alias>
```


SD secondary server security

SD secondary servers support similar encryption rules as HDR. See Database server configuration requirements for clusters for details.

Encryption can be enabled or disabled between any primary and secondary server pair. That is, you can encrypt traffic between the primary server and one SD secondary server and not encrypt traffic between the primary server and another SD secondary server.

See the Configure SMX connections topic for additional information about setting up and configuring encryption between primary servers and SD secondary servers.

Flow control for shared-disk secondary servers

Flow control provides a way to limit log activity on the primary server so that shared-disk (SD) secondary servers in the cluster do not fall too far behind on processing transactions.

Set the SDS_FLOW_CONTROL configuration parameter on the primary server to enable flow control. All SD secondary servers in the cluster are affected by the primary server's SDS_FLOW_CONTROL configuration parameter setting. When flow control is active, users connected to the primary server may experience slower response time.

Logs are always sent to the SD secondary server in the order in which they were received.

Related information:
SDS_FLOW_CONTROL configuration parameter
RSS_FLOW_CONTROL configuration parameter

Cluster administration

This chapter describes various administrative tasks, some optional, for monitoring and maintaining a cluster. For example, load-balancing to optimize performance, ensuring security.

- How data replication works
- Performing basic administration tasks
- Obtain RS secondary server statistics
- Remove an RS secondary server
- RS secondary server security
- Transaction completion during cluster failover
  You can configure servers in a high-availability cluster environment to continue processing transactions after failover of the primary server.

How data replication works

These topics describe the mechanisms that the database server uses to perform replication of data to secondary servers. For instructions on how to set up, start, and administer the various types of secondary servers, see the table

Table 1. Secondary server setup information

Secondary server type	See
HDR secondary	See High-availability cluster configuration, and information about starting an HDR pair using external backup and restore in the *IBM® Informix® Backup and Restore Guide*.
RS secondary	See Remote standalone secondary servers.
SD secondary	See Shared disk secondary servers

- How data initially replicates
- Replication of primary-server data to secondary servers
  All secondary server types use logs to replicate primary-server replicate data. The primary server sends its entire logical log to HDR and RS secondary servers, but only the log page's position to SD secondary servers.
- Data replication configuration examples
- Troubleshooting high-availability cluster environments
  A high-availability cluster environment requires little or no additional troubleshooting when compared with a stand-alone server environment. This topic explains the terminology used to describe high-availability cluster environments and provides some common troubleshooting procedures.
- Design data replication group clients

How data initially replicates

HDR secondary and RS secondary servers use storage-space backups and logical-log backups (both those backed up to tape and those on disk) to perform an initial replication of the data on the primary database server to the secondary database server.

SD secondary servers do not require a backup and restore from the primary server because SD secondary servers share the same disks as the primary.

To replicate data:

1. Install user-defined types, user-defined routines, and DataBlade modules on both database servers.
2. Register user-defined types, user-defined routines, and DataBlade modules on the primary database server only.
3. To synchronize the data managed by the two database servers, create a level-0 backup of all the storage spaces on the primary database server.
4. Restore all the storage spaces from the backup on the secondary database server in the data-replication pair.
   The secondary database server that you restored from a storage-space backup in the previous step then reads all the logical-log records generated since that backup from the primary database server.

   The database server reads the logical-log records first from any backed-up logical-log files that are no longer on disk and then from any logical-log files on disk.

For detailed instructions about replicating data, see Starting HDR for the First Time. The *IBM® Informix® Backup and Restore Guide* explains how to start replication using ON-Bar.

You must perform the initial backup with a storage-space backup. You cannot use data-migration utilities such as **onload** and **onunload** to replicate data because the physical page layout of tables on each database server must be identical in order for data replication to work.

Replication of primary-server data to secondary servers

All secondary server types use logs to replicate primary-server replicate data. The primary server sends its entire logical log to HDR and RS secondary servers, but only the log page's position to SD secondary servers.

Index page logging can be used by all secondary servers, but is required for replication to RS secondary servers.

Databases must use transaction logging to be replicated.

Warning: If the primary server and secondary server disconnect from each other, and are allowed to independently run as standard servers or primary servers, then high-availability data replication might have to be reestablished.

Replication to HDR secondary servers

There are three synchronization modes that the primary database server can use to replicate data to an HDR secondary server:

- *Fully synchronous mode*, where transactions require acknowledgement of completion on the HDR secondary server before they can complete.
  Data integrity is highest when you use fully synchronous mode, but system performance can be negatively affected if client applications use unbuffered logging and have many small transactions.
- *Asynchronous mode*, where transactions do not require acknowledgement of being received or completed on the HDR secondary server before they can complete.
  System performance is best when you use asynchronous mode, but if there is a server failure, data can be lost.
- *Nearly synchronous mode*, where transactions require acknowledgement of being received on the HDR secondary server before they can complete.
  Nearly synchronous mode can have better performance than fully synchronous mode and better data integrity than asynchronous mode. If used with unbuffered logging, SYNC mode, which is turned on when DRINTERVAL is set to -1, is the same as nearly synchronous mode.

The synchronization mode is controlled by the combination of DRINTERVAL configuration parameter value, HDR_TXN_SCOPE configuration parameter value, and database logging type.
The following two figures illustrate replication from a primary server to an HDR secondary server.
Figure 1. How data replicates from a primary to HDR secondary server

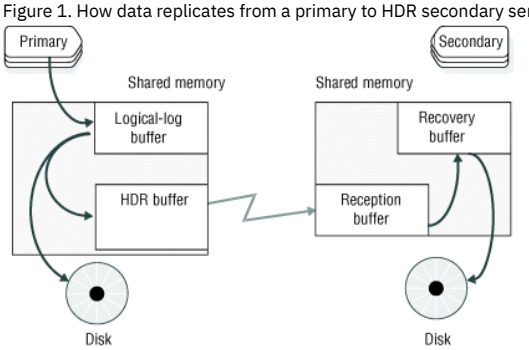
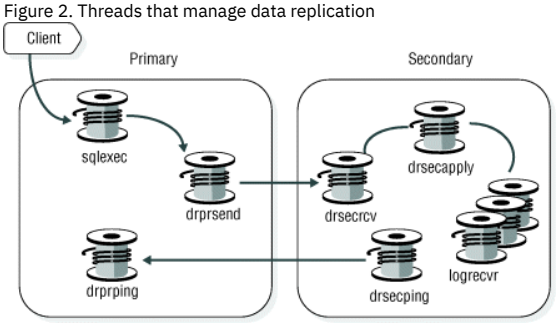


Figure 2. Threads that manage data replication



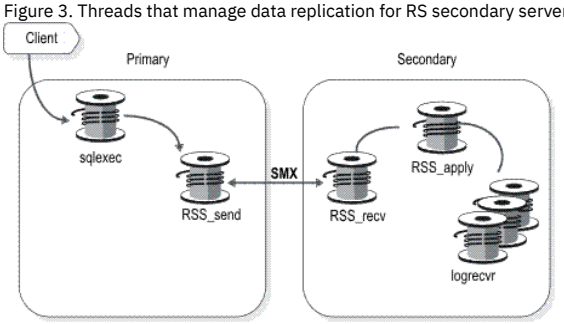
The contents of the primary server's logical-log buffer are copied to the shared-memory *data-replication buffer* and flushed to disk. If the primary server is using fully synchronous or nearly synchronous mode, it must receive an acknowledgement from the HDR secondary server before it can complete the logical-log flush. The primary server starts a **drprsend** thread to transmit the data-replication buffer across the network to the secondary server's **drsecrcv** thread, which then writes the data into the shared-memory *reception buffer*. The **drsecapply** thread copies the reception buffer to the recovery buffer. Both HDR and RS secondary servers use **logrecvr** threads to apply logical-log records their dbspaces. You can adjust the number of **logrecvr** threads by changing the value of the OFF_RECVRY_THREADS configuration parameter.

The **drprping** and **drsecping** threads send and receive messages to monitor the connection between two servers.

Replication to RSS secondary servers

Because checkpoints between a primary server and an RS secondary server are asynchronous, RS secondary servers require index page logging.

The following figure illustrated replication from a primary server to an RS secondary server.
Figure 3. Threads that manage data replication for RS secondary servers



If the primary server can verify that it is connected to an RS secondary server, the **RSS_send** thread copies a page from either the disk or the logical-log buffer to the data-replication buffer. The **RSS_Send** thread uses a Server Multiplexer Group (SMX) connection to send the data-replication buffer to the RS secondary server's **RSS_recv** thread. The **RSS_recv** thread then writes the data into the *reception buffer*. The **RSS_apply** thread copies the reception buffer to the recovery buffer.

Unlike with HDR fully synchronous mode or nearly synchronous mode, the primary server does not require acknowledgment from the secondary server before sending the next buffer. The primary server sends up to 32 unacknowledged data-replication buffers before the **RSS_send** thread waits for the **RSS_Recv** thread to receive an acknowledgment from the RS secondary server.

Replication to SD secondary servers

SD secondary servers read logical log pages from disk and then apply the data to their memory data buffers.

- Fully synchronous mode for HDR replication
  HDR fully synchronous mode ensures that any transaction committed on a primary server was also committed on the HDR secondary server, which can protects transactional consistency if a failure occurs.
- Nearly synchronous mode for HDR replication
  When you use nearly synchronous mode for HDR replication, the primary server flushes the logical-log buffer to disk after receiving acknowledgement that the HDR secondary server received a transmitted transaction. The primary server does not wait for acknowledgement that the transaction was committed on the HDR secondary server.
- Asynchronous mode for HDR replication
  Asynchronous HDR replication means that the primary server does not wait for a response from the HDR secondary server before flushes the logical log to disk. Asynchronous HDR replication can increase replication speed, but transactions can be lost.

- Lost-and-found transactions

Related concepts:
Cluster failures
Index page logging
Related information:
DRINTERVAL configuration parameter
HDR_TXN_SCOPE configuration parameter
HDR_TXN_SCOPE session environment option

Fully synchronous mode for HDR replication

HDR fully synchronous mode ensures that any transaction committed on a primary server was also committed on the HDR secondary server, which can protects transactional consistency if a failure occurs.

After the primary database server writes the logical-log buffer contents to the HDR buffer, it sends the records from the buffer to the HDR secondary database server. The logical-log buffer flush on the primary database server completes only after the primary database server receives acknowledgment from the HDR secondary database server that the records were received.

To track synchronization, both the primary and HDR secondary server store the following information in their reserved pages:

- The ID of the logical-log file that contains the last completed checkpoint
- The position of the checkpoint record within the logical-log file
- The ID of the last logical-log file that was sent or received
- The page number of the last logical-log record that was sent or received

To view this information, run the **onstat -g dri ckpt** command.
Checkpoints between database servers in an HDR replication pair are synchronous. The primary server waits for the HDR secondary server to acknowledge that it received the checkpoint log record before the primary server completes its checkpoint. If the checkpoint does not complete within the time that is specified by the DRTIMEOUT configuration parameter, the primary database server assumes that a failure occurred.

HDR fully synchronous mode has the following requirements:

- The DRINTERVAL configuration parameter on the primary and HDR secondary server must be set to `0`.
- The DRTIMEOUT configuration parameter on the primary and HDR secondary server must be set to the same value.

Administration can be easier if the operating-system times on the primary and HDR secondary servers are synchronized.
To turn on fully synchronous data replication, set the DRINTERVAL configuration parameter to `0`, and then use one of the following methods:

- Set the HDR_TXN_SCOPE configuration parameter to FULL_SYNC.
- Run SET ENVIRONMENT HDR_TXN_SCOPE 'FULL_SYNC';

Log records are applied in the order in which they were received. When the log transmission buffer contains many log records, the application of those log records on the HDR secondary server requires more time, and performance can be negatively affected. If this situation occurs, consider using nearly synchronous mode for HDR data replication.

Related information:
DRINTERVAL configuration parameter
onstat -g dri command: Print high-availability data replication information
HDR_TXN_SCOPE session environment option

Nearly synchronous mode for HDR replication

When you use nearly synchronous mode for HDR replication, the primary server flushes the logical-log buffer to disk after receiving acknowledgement that the HDR secondary server received a transmitted transaction. The primary server does not wait for acknowledgement that the transaction was committed on the HDR secondary server.

When the log transmission buffer contains many log records, the application of those log records on the HDR secondary server requires more time. Nearly synchronous mode for HDR replication provides better performance than fully synchronous mode, and better data integrity than asynchronous mode.

The primary server stores the following near-synchronization information in its reserved page:

- The number of unprocessed data replication buffers queued to the **drprsend** thread.
- The log unique value, the page number for the most recently paged log.
- The pointer to the thread-control block (TCB), the thread id in parentheses, and the log sequence number (LSN) of the commit that was performed by that thread.
- The LSNs of commits that are waiting for acknowledgement of being received on the HDR secondary.

To view this information, run the **onstat -g dri que** command.
HDR nearly synchronous mode has the following requirements:

- The DRINTERVAL configuration parameters on the primary and HDR secondary server must be set to `-1`, or the DRINTERVAL configuration parameter on the primary server must be set to `0`.
- The DRTIMEOUT configuration parameters on the primary and HDR secondary server must be set to the same value.
- The operating-system time on the primary and HDR secondary servers must be synchronized.

To turn on nearly synchronous data replication, set the DRINTERVAL configuration parameter to `0`, and then use one of the following methods:

- Set the HDR_TXN_SCOPE configuration parameter to NEAR_SYNC.
- Run `SET ENVIRONMENT HDR_TXN_SCOPE 'NEAR_SYNC';`

Related information:
DRINTERVAL configuration parameter
onstat -g dri command: Print high-availability data replication information
HDR_TXN_SCOPE session environment option

Asynchronous mode for HDR replication

Asynchronous HDR replication means that the primary server does not wait for a response from the HDR secondary server before flushes the logical log to disk. Asynchronous HDR replication can increase replication speed, but transactions can be lost.

There are multiple ways to turn on asynchronous mode for HDR replication:

- Set the DRINTERVAL configuration parameter to a positive integer value.
- Set the DRINTERVAL configuration parameter to 0, and set the HDR_TXN_SCOPE configuration parameter to ASYNC.
- Run the following statement:

```
SET ENVIRONMENT HDR_TXN_SCOPE 'ASYNC';
```

In asynchronous mode, the primary database server flushes the logical-log to disk after it copies the contents of the logical-log buffer to the data-replication buffer. The primary database server sends the contents of the HDR buffer across the network when one of the following conditions occurs:

- The HDR buffer becomes full.
- The time interval since the records were sent to the HDR secondary database server exceeds the value of the primary server's DRINTERVAL configuration parameter.

To reduce the risk of lost transactions in a cluster that uses asynchronous replication, use unbuffered logging for all the databases. Unbuffered logging reduces the amount of time between transaction-record writing and transfer. If your primary server uses buffered logging, and you receive an `error -7350 Attempt to update a stale version of a row` message, switch to unbuffered logging.

If a failover does occur, but the primary server is restarted with data replication, transactions that were committed on the primary server and not committed on the secondary server are stored in a file that is specified by the DRLOSTFOUND configuration parameter.

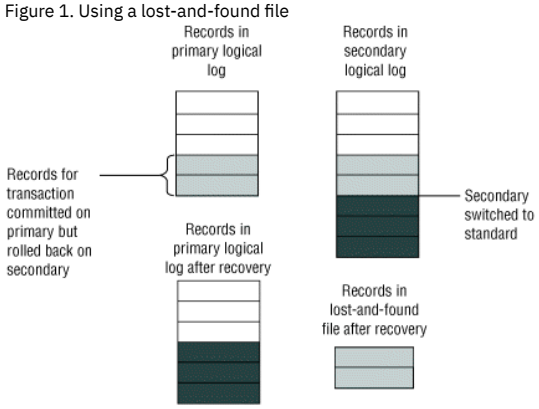
Related information:
DRINTERVAL configuration parameter
onstat -g dri command: Print high-availability data replication information
HDR_TXN_SCOPE session environment option

Lost-and-found transactions

With asynchronous updating, a transaction committed on the primary database server might not be replicated on the secondary database server. This situation can result if a failure occurs after the primary database server copies a commit record to the HDR buffer but before the primary database server sends that commit record to the secondary database server.

If the secondary database server is changed to a standard database server after a failure of the primary database server, it rolls back any open transactions. These transactions include any that were committed on the primary database server but for which the secondary database server did not receive a commit record. As a result, transactions are committed on the primary database server but not on the secondary database server. When you restart data replication after the failure, the database server places all the logical-log records from the lost transactions in a file (which the DRLOSTFOUND configuration parameter specifies) during logical recovery of the primary database server. The following figure illustrates the process.

Figure 1. Using a lost-and-found file



If the lost-and-found file is created on the computer that is running the primary database server after it restarts data replication, a transaction has been lost. The database server cannot reapply the transaction records in the lost-and-found file because conflicting updates might have occurred while the secondary database server was acting as a standard database server.

To reduce the risk of a lost transaction without running data replication in synchronous mode, use unbuffered logging for all the databases. This method reduces the amount of time between the writing and transfer of the transaction records from the primary database server to the secondary database server.

Data replication configuration examples

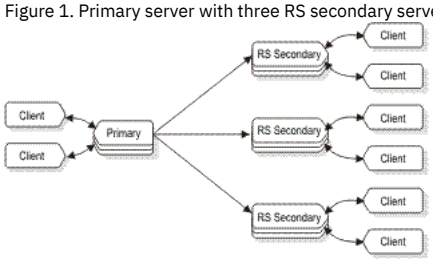
These topics describe some examples of how a data replication environment can be configured.

- Remote standalone secondary configuration examples
- Shared disk secondary configuration examples
- Enterprise Replication as part of the recoverable group
- High-availability clusters with Enterprise Replication configuration example
- Example of a complex failover recovery strategy

Remote standalone secondary configuration examples

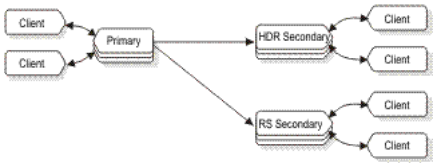
The following figure illustrates an example of a configuration consisting of multiple RS secondary servers. This configuration would be useful in a situation where the primary server is located a long distance from the RS secondary servers or if the network speed between the primary server and the RS secondary server is slow or erratic. Because RS secondary servers use fully duplexed communication protocols, and do not require synchronous checkpoints processing, the primary-server's performance is usually unaffected.

Figure 1. Primary server with three RS secondary servers

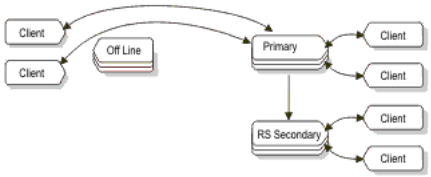


The next illustration shows an example of a configuration of an RS secondary server along with an HDR secondary server. In this example, the HDR secondary provides high availability while the RS secondary provides additional disaster recovery if both the primary and HDR secondary servers are lost. The RS secondary server can be geographically remote from the primary and HDR secondary servers so that a regional disruption such as an earthquake or flood would not affect the RS secondary server.

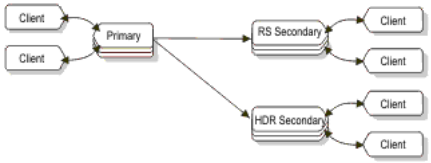
Figure 2. Primary server with HDR secondary and RS secondary servers



If a primary database server fails, it is possible to convert the existing HDR secondary server into the primary server, as in the following diagram:
Figure 3. Failover of primary server to HDR secondary server



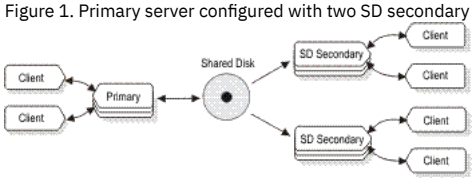
If the original primary is going to be offline for an extended period of time, then the RS secondary server can be converted to an HDR secondary server. Then when the original primary comes back online, it can be configured as an RS secondary server, as in the following illustration:
Figure 4. RS secondary server assuming role of HDR secondary server



Shared disk secondary configuration examples

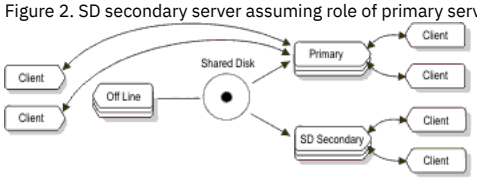
The following figure shows an example of a primary server with two SD secondary servers. In this case the role of the primary server can be transferred to either of the two SD secondary servers. This is true whether the primary must be taken out of service because of a planned outage, or because of failure of the primary server.

Figure 1. Primary server configured with two SD secondary servers



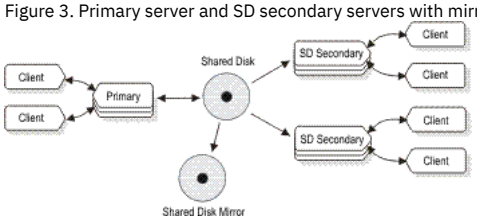
Because both of the SD secondary servers are reading from the same disk subsystem, they are both equally able to assume the primary server role. The following figure illustrates a situation in which the primary server is offline.

Figure 2. SD secondary server assuming role of primary server



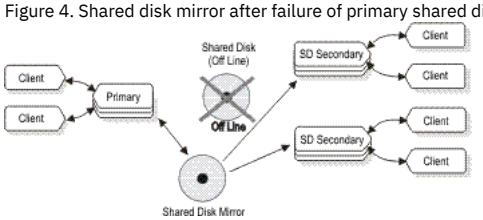
There are several ways to protect against hardware failure of the shared disk. Probably the most common way is to configure the disk array based on RAID technology (such as RAID-5). Another way to protect against disk failure is to use SAN (Storage Area Technology) to include some form of remote disk mirroring. Since SAN disks can be located a short distance from the primary disk and its mirror, this provides a high degree of availability for both the planned and unplanned outage of either the server or of the disk subsystem. The following illustration depicts such a configuration:

Figure 3. Primary server and SD secondary servers with mirrored disks



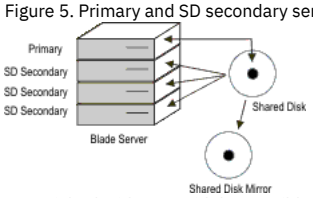
In the event of a disk failure, the servers can be reconfigured as in the following illustration:

Figure 4. Shared disk mirror after failure of primary shared disk



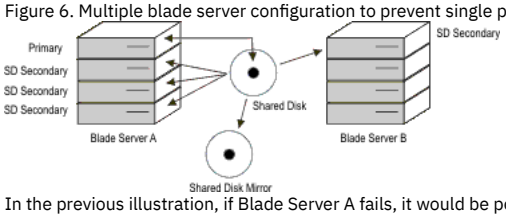
In addition to configuring a mirrored disk subsystem as in the previous illustration, you might want to configure additional servers. For example, you might want to use the primary and two SD secondary servers within a single blade server enclosure. By placing the server group within a single blade server, the blade server itself can become a failure point. The configuration in the following illustration is an attractive solution when you must periodically increase read processing ability such as when performing large reporting tasks.

Figure 5. Primary and SD secondary servers in a blade server



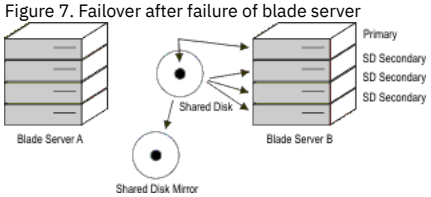
You might decide to avoid the possible failure point of a single blade server by using multiple blade servers, as in the following illustration.

Figure 6. Multiple blade server configuration to prevent single point of failure



In the previous illustration, if Blade Server A fails, it would be possible to transfer the primary server role to the SD secondary server on Blade Server B. Since it is possible to bring additional SD secondary servers online very quickly, it would be possible to dynamically add additional SD secondary servers to Blade Server B as in the following illustration.

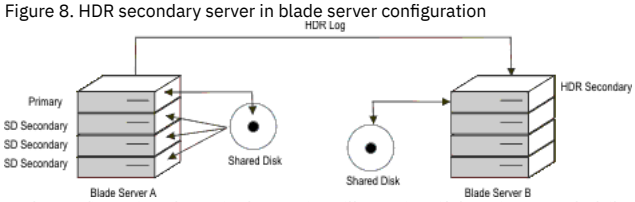
Figure 7. Failover after failure of blade server



Because of limits on the distance between the primary and mirrored disks that disk mirroring can support, you might be concerned about using shared disks and relying on shared disk mirroring to provide disk availability. For example, you might want significant distance between the two copies of the disk subsystem. In this case, you might choose to use either an HDR secondary or an RS secondary server to maintain the secondary copy of the disk subsystem. If the network connection is fairly fast (that is, if

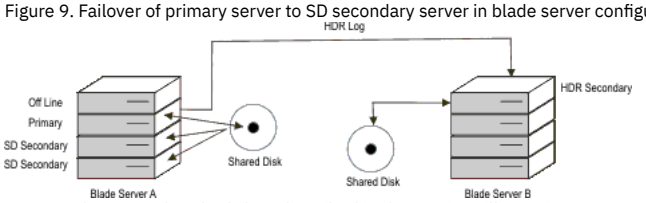
a ping to the secondary server is less than 50 milliseconds) you must consider using an HDR secondary server. For slower network connections, consider using an RS secondary server. The following illustration shows an example of an HDR secondary server in a blade server configuration.

Figure 8. HDR secondary server in blade server configuration



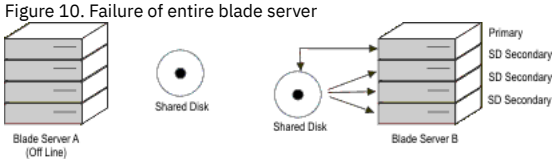
In the configuration shown in the previous illustration, if the primary node fails, but the shared disks are intact and the blade server is still functional, it is possible to transfer the primary server role from the first server in Blade Server A to another server in the same blade server. Changing the primary server would cause the source of the remote HDR secondary server to automatically reroute to the new primary server, as illustrated in the following diagram:

Figure 9. Failover of primary server to SD secondary server in blade server configuration



Suppose, however, that the failure described in the previous illustration was not a blade within the blade server, but the entire blade server. In this case you might be required to fail over to the HDR secondary. Since starting an SD secondary server is very quick, you can easily add additional SD secondary servers. Note that the SD secondary server can only work with the primary node; when the primary has been transferred to Blade Server B, then it becomes possible to start SD secondary servers on Blade Server B as well, as shown in the following illustration.

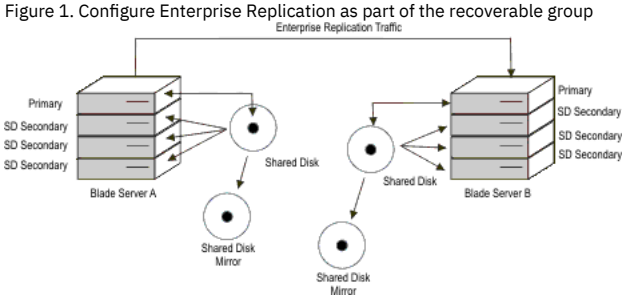
Figure 10. Failure of entire blade server



Enterprise Replication as part of the recoverable group

While Enterprise Replication does not support a SYNC (synchronous) mode of operation, it does provide the ability to support environments with multiple active servers. During a failover event, Enterprise Replication is able to reconcile database differences between two servers. You must consider Enterprise Replication as a means of improving synchronization between servers because each Enterprise Replication system maintains an independent logging system. A configuration using Enterprise Replication is shown in the following figure.

Figure 1. Configure Enterprise Replication as part of the recoverable group



High-availability clusters with Enterprise Replication configuration example

Suppose you require Enterprise Replication between two high-availability server clusters configured as follows:

Cluster 1:

- Primary server
- HDR secondary server
- SD secondary server 1
- SD secondary server 2
- RS secondary server 1
- RS secondary server 2

Cluster 2:

- Primary server
- HDR secondary server
- SD secondary server 1
- SD secondary server 2

- RS secondary server 1
- RS secondary server 2

Suppose further that each of the servers is named according to the following convention:

- First three characters: name of enterprise
- Character 4: host short number
- Characters 5, 6, and 7: cluster number
- Characters 8, 9, and 10: server type: "pri" for primary server, "sec" for secondary server
- Characters 11, 12, and 13: connection type: "shm" or "tcp"

For example, a server with the name: **srv4_1_pri_shm** is described as follows:

- srv = name of enterprise
- 4 = host short number
- _1_ = cluster number
- pri = this is a primary server
- shm = connection type uses shared memory communication

The following entries in the sqlhosts file would support the previous configuration:

```
srv4_1_pri_shm onipcshm sun-mach4 srv4_1_pri_shm
srv4_1_sec_shm onipcshm sun-mach4 srv4_1_sec_shm
srv5_1_rss_shm onipcshm sun-mach5 srv5_1_rss_shm
srv5_1_sds_shm onipcshm sun-mach5 srv5_1_sds_shm
srv6_1_rss_shm onipcshm sun-mach6 srv6_1_rss_shm
srv6_1_sds_shm onipcshm sun-mach6 srv6_1_sds_shm
srv_1_cluster group - - i=1
srv4_1_pri_tcp ontlitcp sun-mach4 21316 g=srv_1_cluster
srv4_1_sec_tcp ontlitcp sun-mach4 21317 g=srv_1_cluster
srv5_1_rss_tcp ontlitcp sun-mach5 21316 g=srv_1_cluster
srv5_1_sds_tcp ontlitcp sun-mach5 21317 g=srv_1_cluster
srv6_1_rss_tcp ontlitcp sun-mach6 21316 g=srv_1_cluster
srv6_1_sds_tcp ontlitcp sun-mach6 21317 g=srv_1_cluster

srv4_2_pri_shm onipcshm sun-mach4 srv4_2_pri_shm
srv4_2_sec_shm onipcshm sun-mach4 srv4_2_sec_shm
srv5_2_rss_shm onipcshm sun-mach5 srv5_2_rss_shm
srv5_2_sds_shm onipcshm sun-mach5 srv5_2_sds_shm
srv6_2_rss_shm onipcshm sun-mach6 srv6_2_rss_shm
srv6_2_sds_shm onipcshm sun-mach6 srv6_2_sds_shm
srv_2_cluster group - - i=2
srv4_2_pri_tcp ontlitcp sun-mach4 21318 g=srv_2_cluster
srv4_2_sec_tcp ontlitcp sun-mach4 21319 g=srv_2_cluster
srv5_2_rss_tcp ontlitcp sun-mach5 21318 g=srv_2_cluster
srv5_2_sds_tcp ontlitcp sun-mach5 21319 g=srv_2_cluster
srv6_2_rss_tcp ontlitcp sun-mach6 21318 g=srv_2_cluster
srv6_2_sds_tcp ontlitcp sun-mach6 21319 g=srv_2_cluster
```


Example of a complex failover recovery strategy

This topic describes a three-tiered server approach for achieving maximum availability in the case of a large region-wide disaster.

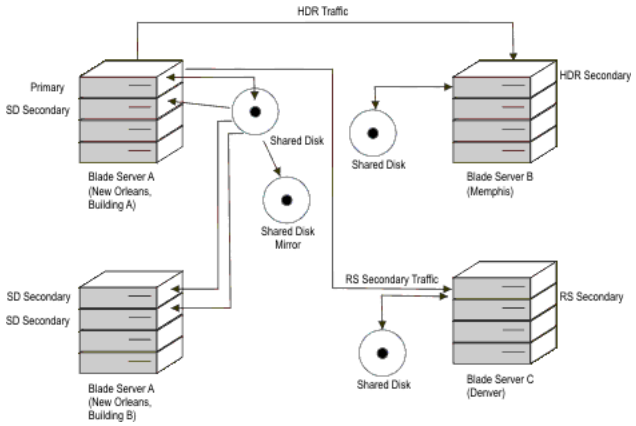
In general, an HDR Secondary server provides backup for SD secondary servers and provides support for a highly available system which is geographically remote from the main system. RS secondary servers provide additional availability for the HDR secondary and are viewed as a disaster-availability solution. If you must use an RS secondary server for availability, then you are forced to manually rebuild the other systems by performing backup and restore in order to return to normal operation. To further understand this, a scenario is presented in which a large region-wide disaster occurs, such as a hurricane.

To provide maximum availability to survive a regional disaster requires *layered* availability. The first layer provides availability solutions to deal with transitory local failures. For example, this might include having a couple of blade servers attached to a single disk subsystem running SD secondary servers. Placing the SD secondary servers in several locations throughout your campus makes it possible to provide seamless failover in the event of a local outage.

You might want to add a second layer to increase availability by including an alternative location with its own copy of the disks. To protect against a large regional disaster, you might consider configuring an HDR secondary server located some distance away, perhaps hundreds of miles. You might also want to make the remote system a blade server or some other multiple-server system. By providing this second layer, if a fail-over occurs and the remote HDR secondary became the primary, then it would be possible to easily start SD secondary servers at the remote site.

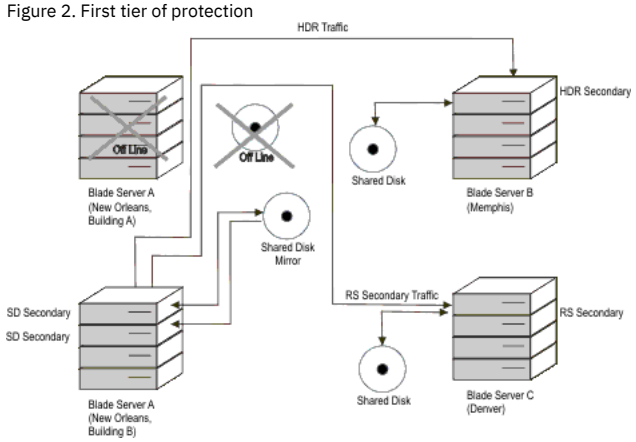
However, even a two-tiered approach might not be enough. A hurricane in one region can create tornadoes hundreds of miles away. To protect against this, consider adding a third tier of protection, such as an RS secondary server located one or more thousand miles away. This three-tier approach provides for additional redundancy that can significantly reduce the risk of an outage.

Figure 1. Configuration for three-tiered server availability



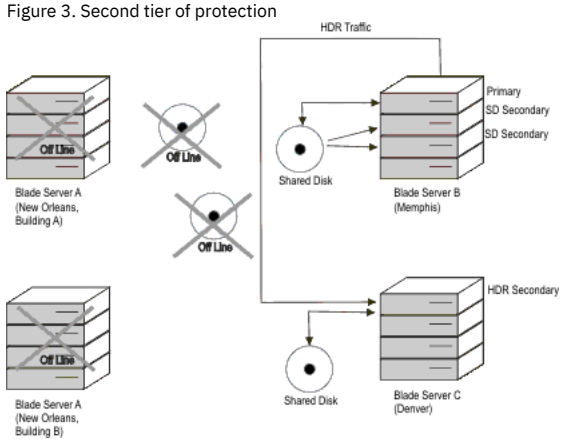
Now suppose that a local outage occurred in Building-A on the New Orleans campus. Perhaps a pipe burst in the machine room causing water damage to the blade server and the primary copy of the shared disk subsystem. You can switch the role of primary server to Building-B by running onmode -d make primary servername on one of the SD secondary servers running on the blade server in Building-B. This would cause all other secondary nodes to automatically connect to the new primary node.

Figure 2. First tier of protection



If there be a regional outage in New Orleans such that both building A and building B were lost, then you can shift the primary server role to Memphis. In addition, you might also want to make Denver into an HDR secondary and possibly add additional SD secondary servers to the machine in Memphis.

Figure 3. Second tier of protection



An even larger outage which affected both sites would require switching to the most remote system.

Figure 4. Third tier of protection

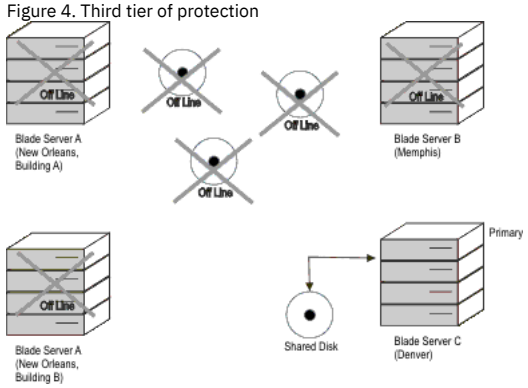


Table 1. Suggested configurations for various requirements

Requirement	Suggested configuration

Requirement	Suggested configuration
You periodically must increase reporting capacity	Use SD secondary servers
You are using SAN devices, which provide ample disk hardware availability, but are concerned about server failures	Use SD secondary servers
You are using SAN devices, which provide ample disk hardware mirroring, but also want a second set of servers that are able to be brought online if the main operation is lost (and the limitations of mirrored disks are not a problem)	Consider using two blade centers running SD secondary servers at the two sites
You want to have a backup site some moderate distance away, but cannot tolerate any loss of data during failover	Consider using two blade centers with SD secondary servers on the main blade center and an HDR secondary on the remote.
You want to have a highly available system in which no transaction is ever lost, but must also have a remote system on the other side of the world	Consider using a local HDR secondary server that is running fully synchronous mode or nearly synchronous mode for data replication, and also using an RS secondary server on the other side of the world.
You want to have a high availability solution, but because of the networks in your region, the best response time from a ping is about 200 ms	Consider using an RS secondary server
You want a backup site but you do not have any direct communication with the backup site	Consider using Continuous Log Restore with backup and recovery
You can tolerate a delay in the delivery of data as long as the data arrives eventually; however you must have quick failover in any case	Consider using SD secondary servers with hardware disk mirroring in conjunction with ER.
You require additional write processing power, can tolerate some delay in the delivery of those writes, require something highly available, and can partition the workload	Consider using ER with SD secondary servers

Troubleshooting high-availability cluster environments

A high-availability cluster environment requires little or no additional troubleshooting when compared with a stand-alone server environment. This topic explains the terminology used to describe high-availability cluster environments and provides some common troubleshooting procedures.

You use the diagnostic tools to display the configuration of a high-availability environment and to verify that your secondary servers are set up correctly to update data.

Transactions are processed by the servers very quickly. The **onstat** commands display status information only for the instant the command was run.

To update data on secondary servers, IBM® Informix® creates *proxy distributors* on both the primary and the secondary database servers. Each proxy distributor is assigned an ID that is unique within the cluster. The proxy distributor is responsible for sending DML update requests from secondary servers to the primary server. Secondary servers determine how many instances of the proxy distributors to create based on the UPDATABLE_SECONDARY setting in the secondary server's onconfig file.

For updatable secondary servers in a high-availability cluster environment, encryption from the updatable secondary server to primary server requires SMX encryption. To encrypt data sent from an updatable secondary server to the primary server, set the ENCRYPT_SMX configuration parameter on the secondary server. See ENCRYPT_SMX configuration parameter for more information.

When initializing an updatable secondary server in a high-availability cluster, the server remains in fast recovery mode until all open transactions, including open and prepared XA transactions, are complete. Applications cannot connect to the server while it is in fast recovery mode. The server remains in fast recovery mode until all open transactions are either committed or rolled back.

Use the **onstat -g proxy** command on the primary server to view information about all proxy distributors in the high-availability cluster.

```
onstat -g proxy

Secondary     Proxy       Reference     Transaction  Hot Row
Node          ID          Count         Count        Total
serv2         392         0             2            112
serv2         393         0             2            150
```

Examining the output from the **onstat** command in the previous example, there are two proxy distributors whose IDs are 392 and 393. The Transaction Count indicates the number of transactions currently being processed by each proxy distributor.
You run **onstat -g proxy** on a secondary server to view information about the proxy distributors that are able to service update requests from the secondary server.

```
onstat -g proxy

Primary     Proxy     Reference     Transaction  Hot Row
Node        ID        Count         Count        Total
serv1       392       0             2            112
serv1       393       0             2            150
```

In this example, the server is a shared disk (SD) secondary server, and is configured to update data. In addition, the server is connected to the primary server named **serv1**, and there are two proxy distributors, each with a transaction count of 2.
Use **onstat -g proxy all** on the primary server to display information about proxy distributors and *proxy agent threads*. One or more proxy agent threads are created by the proxy distributor to handle data updates from the secondary server.

```
onstat -g proxy all

Secondary   Proxy       Reference     Transaction  Hot Row
Node        ID          Count         Count        Total
serv2       392         0             2            1
serv2       393         0             2            0

TID  Flags       Proxy  Source  Proxy   Current  sqlerrno iserrno
```

```
                ID      SessID  TxnID   Seq
63    0x00000024  392     22      1       5       0       0
64    0x00000024  392     19      2       5       0       0
62    0x00000024  393     23      1       5       0       0
65    0x00000024  393     21      2       5       0       0
```

In the output of the **onstat -g proxy all** command, **TID** represents the ID of the proxy agent thread that is running on the primary server. **Source SessID** represents the ID of the user's session on the secondary server. **Proxy TxnID** displays the sequence number of the current transaction. Each **Proxy TxnID** is unique to the proxy distributor. **Current Seq** represents the sequence number of the current operation in the transaction being processed. Each database transaction sent to a secondary server is separated internally into one or more operations that are then sent to the primary server. The last two fields, **sqlerrno** and **iserrno**, display any SQL or ISAM/RSAM errors encountered in the transaction. An error number of 0 indicates completion with no errors.
Running **onstat -g proxy all** on the secondary server displays information about all of the sessions that are currently able to update data on secondary servers.

```
onstat -g proxy all

Primary   Proxy      Reference     Transaction  Hot Row
Node      ID         Count         Count        Total
serv1     3466       0             0            1
serv1     3465       0             1            0

Session  Proxy  Proxy  Proxy  Current  Pending  Reference
         ID     TID    TxnID  Seq      Ops      Count
19       3465   67     1      23       0        1
```

In the output from the **onstat -g proxy all** command run on the secondary server, **Session** represents the ID of a user's session on the secondary server. The **Proxy ID** and **Proxy TID** are the same as those displayed on the primary server. **Pending Ops** displays the number of operations that are waiting to be transferred to the primary server. **Reference Count** displays the number of threads in use for the transaction. When **Reference Count** displays 0 the transaction processing is complete.
To display detailed information about the current work being performed by a given distributor, use:

```
onstat -g proxy <proxy id> [proxy transaction id] [operation number]
```

The proxy transaction ID and operation number are both optional parameters. When supplied, the first number is considered the proxy transaction ID. If a secondary number follows it is interpreted as the operation number. If no proxy transaction ID exists, the command performs the same as: **onstat -**. Similarly, if no such operation number for the given proxy transaction ID exists, the command performs the same as: **onstat -**.
Use the following command to display information about whether a server is configured to allow updates to data. The command can be run either on the primary or secondary server:

```
onstat -g <server_type>
```

Examples:

```
onstat -g rss
onstat -g sds
onstat -g dri
```


Design data replication group clients

This topic explains various design considerations for clients that connect to database servers that are running data replication.

Also see Isolation levels on secondary servers for information about **committed read** and **committed read last committed** isolation levels on secondary servers.

- Use of temporary dbspaces for sorting and temporary tables

Use of temporary dbspaces for sorting and temporary tables

Even though the secondary database server is in read-only mode, it does write when it must sort or create a temporary table. Temporary dbspaces explains where the database server finds temporary space to use during a sort or for a temporary table.

To prevent the secondary database server from writing to a dbspace that is in logical-recovery mode, you must take the following actions:

1. Ensure that one or more temporary dbspaces exist. For instructions on creating a temporary dbspace, see Creating a dbspace that uses the default page size.
2. Perform one of the following actions:
   - Set the DBSPACETEMP parameter in the onconfig file of the secondary database server to the temporary dbspace or dbspaces.
   - Set the DBSPACETEMP environment variable of the client applications to the temporary dbspace or dbspaces.

Temporary tables created on secondary servers (SD secondary servers, RS secondary servers, and HDR secondary servers) must be created using the WITH NO LOG option. Alternatively, set the TEMPTAB_NOLOG configuration parameter to 1 or 2 on the secondary server to change the default logging mode for temporary tables to no logging. Tables created with logging enabled result in ISAM errors.

For SD secondary servers, set the SDS_TEMPDBS configuration parameter for configuring temporary dbspaces to be used by the SD secondary server.

For SD secondary servers, it is not necessary to explicitly add a temporary dbspace because the secondary server allocates the chunk specified by SDS_TEMPDBS when the server is started. It is only necessary to prepare the device that accepts the chunk.

If the primary server in a high-availability cluster fails and an SD secondary server takes over as the primary server, then the value set for the SDS_TEMPDBS configuration parameter on the SD secondary server is used for temporary dbspaces until the server is restarted. You must ensure that the value specified for the SDS_TEMPDBS

configuration parameter on the SD secondary server is different than the value specified on the primary server. After the SD secondary server is restarted, the DBSPACETEMP configuration parameter is used.

Performing basic administration tasks

These topics contain instructions on how to perform database server administration tasks when your system is running HDR.

- Changing the configuration parameters for an HDR replication pair
- Back up storage spaces and logical-log files
  When you use HDR, you must back up logical-log files and storage spaces only on the primary database server. Be prepared, however, to perform storage-space and logical-log backups on the secondary database server in case the type of the database server is changed to standard.
- Changing the logging mode of databases
- Add and drop chunks and storage spaces
- Renaming chunks
- Saving chunk status on the secondary database server
  For high-availability cluster servers, if the status of a chunk (down or online) is changed on the secondary database server, and that secondary server is restarted before a checkpoint is completed, the updated chunk status is not saved.
- Use and change mirroring of chunks
- Manage the physical log
- Manage the logical log
- Manage virtual processors
- Manage shared memory
- Configure SMX connections
  Server Multiplexer Group (SMX) is a communications interface that supports encrypted multiplexed network connections between servers in high availability environments. SMX provides a reliable, secure, high-performance communication mechanism between database server instances.
- Replicate an index to an HDR secondary database server
- Encrypting data traffic between HDR database servers
- Adjust LRU flushing and automatic tuning in HDR server pairs
- Cloning a primary server
  You can use the **ifxclone** utility to perform one-step server instantiation, allowing a primary server in a high-availability cluster to be cloned with minimum setup or configuration.
- Database updates on secondary servers
  You can enable applications connected to secondary servers to update database data. If you enable write operations on a secondary server, DELETE, INSERT, MERGE, and UPDATE operations are propagated to the primary server.
- Backup and restore with high-availability clusters
  You cannot perform most backup and restore operations on secondary servers.
- Change the database server mode
  If you change the mode of a database server in a high-availability cluster, replication stops.
- Changing the database server type
  You can change the type of either the primary or the secondary database server.
- Prevent blocking checkpoints on HDR servers
- Monitor HDR status

Changing the configuration parameters for an HDR replication pair

Certain configuration parameters must be set to the same value on both database servers in a HDR replication pair (as listed under Database server configuration requirements for clusters.) Configuration parameters that can have different values on each database server can be changed individually.

To make changes to onconfig files:

1. Bring each database server offline with the **onmode -k** option. If automatic failover by Connection Managers or automatic switchover from DRAUTO settings of 1 or 2 are enabled, bring the HDR secondary server offline first.
2. Change the parameters on each database server.
3. Starting with the database server that was last brought offline, bring each database server back online.
   For example, if you brought the HDR secondary database server offline last, bring the HDR secondary database server online first. Table 1 lists the procedures for bringing the primary and secondary database servers back online.

Back up storage spaces and logical-log files

When you use HDR, you must back up logical-log files and storage spaces only on the primary database server. Be prepared, however, to perform storage-space and logical-log backups on the secondary database server in case the type of the database server is changed to standard.

You must use the same backup and restore tool on both database servers.

The block size and tape size used (for both storage-space backups and logical-log backups) must be identical on the primary and secondary database servers.

You can use **ontape** to set the tape size to 0 to automatically use the full physical capacity of a tape.

Changing the logging mode of databases

You cannot turn on transaction logging for databases on the primary database server while you are using HDR. You can turn logging off for a database; however, subsequent changes to that database are not duplicated on the secondary database server.

To turn on database logging:

1. To turn HDR off, shut down the secondary database server.
2. Turn on database logging.
   After you turn on logging for a database, if you start data replication without performing the level-0 backup on the primary database server and restore on the secondary database server, the database on the primary and secondary database servers might have different data. This situation might cause data-replication problems.
3. Perform a level-0 backup on the primary database server and restore on the secondary database server. This procedure is described in Starting HDR for the First Time.

Add and drop chunks and storage spaces

You can perform disk-layout operations, such as adding or dropping chunks and dbspaces, only from the primary database server. The operation is replicated on the secondary database server. This arrangement ensures that the disk layout on both database servers in the replication pair remains consistent.

The directory path name or the actual file for chunks must exist before you create them. Make sure the path names (and offsets, if applicable) exist on the secondary database server before you create a chunk on the primary database server, or else the database server turns off data replication.

Tip: When adding a dbspace on the primary server of a high-availability cluster that has one or more SD secondary servers, the online.log of an SD secondary server might show this error: "Assert Failed: Page Check Error". If that happens, shut down and restart that SD secondary server. After restarting that SD secondary server, the newly added dbspace will be available and fully functional.

Renaming chunks

If you use symbolic links for chunk path names, you can rename chunks while HDR is operating. For instructions on renaming chunks, see the *IBM® Informix® Backup and Restore Guide*.

If you do not use symbolic links for chunk path names, you must take both database servers offline while renaming the chunks, for the time that it takes to complete a cold restore of the database server.

To rename chunks on a failed HDR server:

1. Change the mode of the undamaged server to standard.
2. Take a level-0 backup of the standard server.
3. Shut down the standard server.
4. Rename the chunks on the standard server during a cold restore from the new level-0 archive (for instructions, see the *IBM Informix Backup and Restore Guide*).
5. Start the standard server.
6. Take another level-0 archive of the standard server. Be sure the server is in standard mode.
7. Restore the failed server with the new level-0 backup and reestablish the HDR pair.

Saving chunk status on the secondary database server

For high-availability cluster servers, if the status of a chunk (down or online) is changed on the secondary database server, and that secondary server is restarted before a checkpoint is completed, the updated chunk status is not saved.

To ensure that the new chunk status is flushed to the reserved pages on the secondary database server, force a checkpoint on the primary database server and verify that a checkpoint also completes on the secondary database server. The new chunk status is now retained even if the secondary database server is restarted.

If the primary chunk on the secondary database server is down, you can recover the primary chunk from the mirror chunk.

To recover the primary chunk from the mirror chunk:

1. Run **onspaces -s** on the secondary database server to bring the primary chunk online.
2. Run **onmode -c** on the primary database server to force a checkpoint.
3. Run **onmode -m** on the primary database server to verify that a checkpoint was completed.

4. Run **onmode -m** on the secondary database server to verify that a checkpoint was also completed on the secondary database server.

After you complete these steps, the primary chunk is online when you restart the secondary database server.

Use and change mirroring of chunks

Before you can add a mirror chunk, the disk space for that chunk must already be allocated on both the primary and secondary database servers. If you want to mirror a dbspace on one of the database servers in the replication pair, you must create mirror chunks for that dbspace on both database servers. For general information about allocating disk space, see Allocate disk space.

Do not set the MIRROR configuration parameter to `1` unless you are using mirroring.

You can perform disk-layout operations from the primary database server only. Thus, you can add or drop a mirror chunk only from the primary database server. A mirror chunk that you add to or drop from the primary database server is added to or dropped from the secondary database server as well. You must perform mirror recovery for the newly added mirror chunk on the secondary database server. For more information, see Recover a mirror chunk.

When you drop a chunk from the primary database server, IBM® Informix® automatically drops the corresponding chunk on the secondary database server. This applies to both primary and mirror chunks.

When you turn mirroring off for a dbspace on the primary database server, does not turn mirroring off for the corresponding dbspace on the secondary database server. To turn off mirroring for a dbspace on the secondary database server independent of the primary server, use **onspaces -r**. For more information about turning off mirroring, see End mirroring.

You can take down a mirror chunk or recover a mirror chunk on either the primary or secondary database server. These processes are transparent to HDR.

Manage the physical log

The size of the physical log must be the same on both database servers. If you change the size and location of the physical log on the primary database server, this change is replicated to the secondary database server. ONCONFIG values on secondary are updated automatically.

For information about changing the size and location of the physical log, see Manage the physical log.

Manage the logical log

The size of the logical log must be the same on both database servers. You can add or drop a logical-log file with the **onparams** utility, as described in Manage logical-log files. IBM® Informix® replicates this change on the secondary database server; however, the LOGFILES parameter on the secondary database server is not updated. After you issue the **onparams** command from the primary database server, therefore, you must manually change the LOGFILES parameter to the appropriate value on the secondary database server. Finally, for the change to take effect, you must perform a level-0 backup of the root dbspace on the primary database server.

If you add a logical-log file to the primary database server, this file is available for use and flagged `F` as soon as you perform the level-0 backup. The new logical-log file on the secondary database server is still flagged `A`. However, this condition does not prevent the secondary database server from writing to the file.

Manage virtual processors

The number of virtual processors has no effect on data replication. You can configure and tune each database server in the pair individually.

Manage shared memory

If you make changes to the shared-memory ONCONFIG parameters on one database server, you must make the same changes at the same time to the shared-memory ONCONFIG parameters on the other database server. For the procedure for making this change, see Changing the configuration parameters for an HDR replication pair.

Configure SMX connections

Server Multiplexer Group (SMX) is a communications interface that supports encrypted multiplexed network connections between servers in high availability environments. SMX provides a reliable, secure, high-performance communication mechanism between database server instances.

Reduce latency between servers

You can reduce latency between high-availability servers by increasing the number of pipes that are used for SMX connections between the servers. Set the SMX_NUMPIPES configuration parameter to the number of pipes.

Obtain SMX statistics

You can use the **onstat** utility or system-monitoring interface (SMI) tables to view SMX connection statistics or SMX session statistics.

To view SMX connection statistics, use the **onstat -g smx** command.

To view SMX session statistics, use the **onstat -g smx ses** command.

Encrypt SMX connections

Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high availability configurations. If you set the ENCRYPT_SMX parameter to 1, encryption is used for SMX transactions only when the database server being connected to also supports encryption. If you set the ENCRYPT_SMX configuration parameter to `2` , only connections to encrypted database servers are allowed. Setting ENCRYPT_SMX to `0` disables encryption between servers.

Set the wait time for SMX activity between servers

You can set the SMX_PING_INTERVAL and SMX_PING_RETRY configuration parameters to adjust the interval that secondary server in a high-availability cluster waits for activity from the primary server. Use the SMX_PING_INTERVAL configuration parameter to specify the number of seconds in a timeout interval, where a secondary server waits for activity from the primary server in an SMX connection.

Use the SMX_PING_RETRY configuration parameter to specify the maximum number of times that a secondary server repeats the timeout interval that is specified by the SMX_PING_INTERVAL configuration parameter if a response from the primary server is not received. If the maximum number is reached without a response, the secondary server prints an error message in the online.log and closes the Server Multiplexer Group (SMX) connection.

Compress data through SMX connections

You can specify the level of compression that the database server uses before sending data from the source database server to the target database server with the SMX_COMPRESS configuration parameter. Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data.

Related information:
SMX_COMPRESS configuration parameter
SMX_PING_INTERVAL configuration parameter
SMX_NUMPIPES configuration parameter
SMX_PING_RETRY configuration parameter
ENCRYPT_SMX configuration parameter
onstat -g smx command: Print multiplexer group information

Replicate an index to an HDR secondary database server

If index page logging is enabled, index replication to the HDR secondary database server occurs automatically (see Index page logging). If index page logging is disabled, and an index on an HDR secondary database server becomes corrupted and must be rebuilt, you can either:

- Manually replicate the index from the primary server to the secondary server.
- Let the secondary server automatically replicate the index if you enabled the secondary server to do this.

To enable the secondary database server to automatically replicate the index, either:

- Set **onmode -d idxauto** to `on`.
- Set the value of the DRIDXAUTO configuration parameter to `1`.

After you set either of these values, when one of the threads on the secondary database server detects a corrupted index, the index is automatically replicated to the secondary database server. Restarting index replication can take up to the amount of time specified in seconds in the DRTIMEOUT configuration parameter.

Sometimes, you might want to replicate an index manually, for example, when you want to postpone index repair because the table is locked. If you want to be able to manually replicate an index on the HDR secondary server, turn off the automatic replication feature.

To turn off the automatic index replication feature, either:

- Set **onmode -d idxauto** to `off`.
- Set the DRIDXAUTO configuration parameter to `0`.

If **onmode -d idxauto** is set to off or DRIDXAUTO is set to `0` and the secondary server detects a corrupted index, you can manually replicate an index on the HDR secondary server by issuing an **onmode -d index** command in the following format.`onmode -d index database:[ownername].table#index`

For example:`onmode -d index cash_db:user_dx.table_12#index_z`

In the case of a fragmented index with one corrupted fragment, the **onmode -d idxauto** option only transfers the single affected fragment, whereas the **onmode -d index** option transfers the whole index.

Important: When turning the automatic index replication feature on or off, you can use either the **onmode** command or the DRIDXAUTO configuration parameter. If you use the **onmode** command, you are not required to stop and restart the database server. When you use the DRIDXAUTO parameter, the database server is restarted with the setting you specify. The **onmode** command does not change the DRIDXAUTO value. If you use the **onmode** command, you must manually change the value of DRIDXAUTO.
The online.log file produced by the secondary server contains information about any replicated index.

Related information:
onmode -d command: Replicate an index with data-replication
DRIDXAUTO configuration parameter

Encrypting data traffic between HDR database servers

To support encrypted HDR connections in conjunction with Communication Support Module (CSM) client/server encryption, two network ports must be configured:

- One network port must be configured for HDR.
- The other network port must be configured for CSM client/server connections.

You can use Informix® server encryption options to encrypt the data traffic between the database servers of an HDR pair. Do this when you want to ensure secure transmission of data.

After you enable encryption, the first database server in an HDR pair encrypts the data before sending the data to the other server in the pair. The server that receives the data, decrypts the data as soon as it receives the data.

For updatable secondary servers in a high-availability cluster environment, encryption from the updatable secondary server to primary server requires SMX encryption. To encrypt data sent from an updatable secondary server to the primary server, set the ENCRYPT_SMX configuration parameter on the secondary server. See ENCRYPT_SMX configuration parameter for more information.
Restriction: You cannot start HDR on a network connection that is configured to use CSM encryption for client/server connections.
Additional buffers or larger buffers might be necessary to accommodate the size of encrypted data.

To encrypt data traffic between two HDR database servers:

1. Set the following configuration parameters on the first server in the HDR pair.
   - ENCRYPT_HDR, which enables or disables HDR encryption
   - ENCRYPT_CIPHERS, which specifies the ciphers and modes to use for encryption
   - ENCRYPT_MAC, which controls the level of message authentication code (MAC) generation
   - ENCRYPT_MACFILE, which specifies a list of the full path names of MAC key files
   - ENCRYPT_SWITCH, which specifies the number of minutes between automatic renegotiations of ciphers and keys

   To change these parameters, follow the instructions in Changing the configuration parameters for an HDR replication pair.
2. Set the encryption configuration parameters on the secondary server. The ENCRYPT_HDR, ENCRYPT_CIPHERS, ENCRYPT_MAC, and the ENCRYPT_SWITCH configuration parameters must have the same values as the corresponding configuration parameters on the primary server. The ENCRYPT_MACFILE configuration parameter can have a different value on each server, but the files must contain the same MAC keys.

For example, specify the following information about the primary and secondary servers in an HDR pair:

Configuration parameter	Sample setting on primary server	Sample setting on secondary server
ENCRYPT_HDR	`1`	`1`
ENCRYPT_CIPHERS	`all`	`all`
ENCRYPT_MAC	`medium`	`medium`
ENCRYPT_MACFILE	`/vobs/tristan/sqldist/etc/mac1.dat`	`vobs/tristan/sqldist/etc/mac2.dat`
ENCRYPT_SWITCH	`60,60`	`60,60`

In this example, the file name in the ENCRYPT_MACFILE path for the primary server is mac1.dat and the file name in the ENCRYPT_MACFILE path for the secondary server is mac2.dat. Otherwise, all settings are the same on both servers.

Only use these configuration parameters to specify encryption information for HDR. You cannot specify HDR encryption information by using the CSM option in the sqlhosts file.

HDR encryption works in conjunction with Enterprise Replication encryption and operates whether Enterprise Replication encryption is enabled or not. When working in conjunction with each other, HDR and Enterprise Replication share the same ENCRYPT_CIPHER, ENCRYPT_MAC, ENCRYPT_MACFILE and ENCRYPT_SWITCH configuration parameters.

For more information about these configuration parameters, see the *IBM® Informix Administrator's Reference.*

Adjust LRU flushing and automatic tuning in HDR server pairs

When a server is configured for HDR, checkpoints triggered by the secondary database server are nonblocking. These types of checkpoints occur infrequently. If a nonblocking checkpoint is triggered by the secondary server, transactions are blocked on the primary server to make sure that the integrity of the secondary server is not compromised. If nonblocking checkpoints triggered by the secondary server occur on your system, you must tune LRU flushing more aggressively on the primary server to reduce transaction blocking.

To increase LRU flushing, reduce the values of **lru_min_dirty** and **lru_max_dirty** in the BUFFERPOOL configuration parameter.

Automatic LRU tuning can be turned on or off independently on each HDR node. The setting can be different on each HDR database server. For information about turning off automatic LRU tuning, see Turn automatic LRU tuning on or off.

For more information about LRU tuning, see the *IBM® Informix® Performance Guide*.

Cloning a primary server

You can use the **ifxclone** utility to perform one-step server instantiation, allowing a primary server in a high-availability cluster to be cloned with minimum setup or configuration.

You can use the **ifxclone** utility to create one of the following database server types:

- Standalone server
- Remote standalone (RS) secondary server
- High-availability data replication (HDR) secondary server
- Shared-disk (SD) secondary server
- Enterprise Replication server

Using the **ifxclone** utility, the database administrator can quickly, easily, and securely create a clone server from a running Informix® instance without requiring to back up data on the source server, and transfer and restore it to the clone server. The backup and restore processes are started simultaneously using the **ifxclone** utility and there is no requirement to read or write data to disk or tape.
Data is transferred from the source server to the target server over the network using encrypted Server Multiplexer Group (SMX) Connections.

You can automate the creation of clone instances by calling the **ifxclone** utility from a script.

- Creating a clone of a primary server
  Use the **ifxclone** utility to create a clone of a primary server.

Creating a clone of a primary server

Use the **ifxclone** utility to create a clone of a primary server.

The general steps to create a clone of a server are as follows:

1. Verify that the ENABLE_SNAPSHOT_COPY configuration parameter on the source server is set to `1`.
2. Verify that the CDR_AUTO_DISCOVER configuration parameter is set to 1, and that the REMOTE_SERVER_CFG file is set on all cluster servers.
3. Set the following environment variables on the target server:
   - INFORMIXDIR
   - INFORMIXSERVER
   - ONCONFIG
   - INFORMIXSQLHOSTS
4. On the target server, create all of the chunks that exist on the source server. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server. Follow these steps to create the chunks:
   a. On the source server, run the **onstat -d** command to display a list of chunks
   b. On the target server, log-in as user **informix** and use the commands **touch**, **chown**, and **chmod** to create the chunks. For example, to create a chunk that is named /usr/informix/chunks/rootdbs.chunk, follow these steps:

      ```
      $ su informix
      Password:
      $ touch /usr/informix/chunks/rootdbs.chunk
      $ chown informix:informix /usr/informix/chunks/rootdbs.chunk
      $ chmod 660 /usr/informix/chunks/rootdbs.chunk
      ```

   c. Repeat all of the commands in the previous step for each chunk reported by the **onstat -d** command.
5. Create an sqlhosts file on the target server's host, and add the target server's connectivity information to the file.
6. On the source server, run the **admin()** or **task()** function with the **cdr add trustedhost** argument, including the target server's trusted-host information.
7. While still logged in as user **informix**, run the **ifxclone** utility with the **--autoconf** option (**-a**) and other appropriate parameters on the target system on which the clone server is started.
   Note: If trusted-host information was added manually, do not use the **--autoconf** option; you must configure trusted hosts and sqlhosts file information manually.

Example: Cloning a primary server

For this example, you have the following system:

- **server_1** is the source server and has the following sqlhosts file entries:

  ```
  #dbservername  nettype   hostname           servicename  options
   server_1      onsoctcp  host1.example.com  123
  ```

- **server_1** has the following configuration parameter settings:
  - ENABLE_SNAPSHOT_COPY 1
  - CDR_AUTO_DISCOVER 1
  - REMOTE_SERVER_CFG authfile.server_1
- **server_1** has the following trusted-host file entries

```
#trustedhost
host1
host1.example.com
host2
host2.example.com
```

- **server_2** is the target server, is on **host2.example.com**, and uses port **456**.

1. On the target server, log in as user **informix** and use the **touch**, **chown**, and **chmod** commands to create, change the owner, and change the permissions for the chunks. The chunk paths must match the paths of the chunks on the source server. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server.
2. Run the **ifxclone** utility on the target server as user **informix**:

```
ifxclone -T -S server_1 -I host1.example.com -P 123 -t server_2
         -i host2.example.com -p 456 -a -k
```

The **ifxclone** utility modifies the sqlhosts file on the source server and creates a copy of the file on the new target server:

```
#dbservername  nettype   hostname           servicename  options
 server_1      onsoctcp  host1.example.com  123
 server_2      onsoctcp  host2.example.com  456
```

The **ifxclone** utility also propagates the trusted-host file on the source server to the target server.
Related information:
The ifxclone utility

Database updates on secondary servers

You can enable applications connected to secondary servers to update database data. If you enable write operations on a secondary server, DELETE, INSERT, MERGE, and UPDATE operations are propagated to the primary server.

Use the UPDATABLE_SECONDARY configuration parameter to control whether the secondary server can update data and to configure the number of connections that update operations use.

Both data definition language (DDL) statements and data manipulation language (DML) statements are supported on secondary servers.

The **dbimport** utility is supported on all updatable secondary servers.

You cannot use the **dbexport** utility on HDR secondary servers or shared disk (SD) secondary servers. The **dbexport** utility is supported on a remote standalone (RS) secondary server only if the server is set to stop applying log files. Use the STOP_APPLY configuration parameter to stop application of log files.

The **dbschema** utility is supported on all updatable secondary servers.

The **dbschema** utility is also supported on read-only secondary servers. However, the **dbschema** utility displays a warning message when running on these servers.

Most applications that use DDL or DML can run on any of the secondary servers in a high-availability cluster; however, the following DDL statements are not supported:

- CREATE DATABASE (with no logging)
- CREATE EXTERNAL TABLE
- CREATE RAW TABLE
- CREATE TEMP TABLE (with logging)
- CREATE XADATASOURCE
- CREATE XADATASOURCE TYPE
- DROP XADATASOURCE
- DROP XADATASOURCE TYPE
- UPDATE STATISTICS

In cluster environments, the SET CONSTRAINTS, SET INDEXES, and SET TRIGGERS statements are not supported on updatable secondary servers. Any session-level index, trigger, or constraint modes that the SET Database Object Mode statement specifies are not redirected for UPDATE operations on table objects in databases of secondary servers.

Client applications can insert, update, and delete rows on a secondary server only if the secondary server image matches that of the primary server. The following data types are supported:

- BIGINT
- BIGSERIAL
- BLOB
- BOOLEAN
- BSON
- BYTE (stored in the table)
- CHAR
- CLOB
- DATE
- DECIMAL
- DATETIME
- FLOAT
- INT
- INT8
- INTERVAL
- JSON
- MONEY

- NCHAR
- NVCHAR
- SERIAL
- SERIAL8
- SMALLFLOAT
- SMALLINT
- TEXT (stored in the table)
- VARCHAR

BYTE and TEXT data types that are stored in blobspaces are not supported because blobspace data is not replicated.

The following data types are also supported if they do not receive a pointer reference to a different partition:

- COLLECTION
- LIST
- LVARCHAR
- MULTISET
- ROW
- SET
- UDTVAR

Any difference between the primary server image and the secondary server image causes an SQL error and the rollback of any changes.

You cannot use the following utilities on HDR secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers:

- **archecker**
- **dbload**
- High-Performance Loader (HPL)
- **ondblog**
- **onload**
- **onparams**
- **onperf**
- **onsnmp**
- **onspaces**
- **onunload**

SD secondary servers are not supported in Windows environments.

Byte range locking is not supported on secondary servers configured for updates. Byte range locks on secondary servers are promoted to full object locks.

Replicate smart large objects

You might receive one or more of the following error messages while working with updatable secondary servers:

- 12014
- 12015
- 12233

These errors generally indicate a problem with a smart large object file descriptor. These errors can be caused by any of the following conditions:

- A smart large object identifier is passed to another transaction or process before committing the transaction. Because all objects including smart large objects are uncommitted until the transaction is committed, do not allow other transactions to use the smart large object. In particular, dirty reads can access locked smart large objects.
- Smart large objects are not closed after opening them. At the end of a transaction, all smart large objects must be closed on secondary servers, especially those that are created and then the transaction is rolled back. Leaving smart large object file descriptors open causes memory to remain allocated until the session terminates.
- Another process has deleted the smart large object on the primary server. Share locks are not automatically propagated from secondary servers to the primary server so a different secondary server might access a smart large object that has actually been deleted on the primary. These accesses work until either the log record containing the delete is replayed on the secondary server or the secondary server is updated by the primary server.

Three additional error codes might be returned when processing dirty read information.

- -126 (`ISAM error: bad row id`)
- -244 (`SQL error: Could not do a physical-order read to fetch next row`)
- -937

Try your query again if you receive any of the previous codes.

LOCK TABLE statement behavior on secondary servers

You can set an exclusive lock on a table from an updatable secondary server in a high-availability cluster. For exclusive mode locks requested from a secondary server, sessions can read the table but not update it. This behavior is similar to shared access mode on a secondary server; that is, when one session has an exclusive lock on a given table, no other session can obtain a shared or exclusive lock on that table.

On read-only secondary servers, the session issuing the LOCK TABLE statement does not lock the table and the database server does not return an error to the client.

Shared mode locks in a cluster behave the same as for a standalone server. After a LOCK TABLE statement runs successfully, users can read the table but cannot modify it until the lock is released.

- Isolation levels on secondary servers
- Transient types on high-availability cluster secondary servers
- Row versioning

Related concepts:
XA in high-availability clusters

Isolation levels on secondary servers

The following statements are supported on all types of secondary servers:

```
Set isolation to committed read
Set isolation to committed read last committed
```

Secondary servers on which Committed Read isolation is set can read locally committed data. They can also read data committed on the primary server when it becomes available and committed on the secondary server. Applications connected to a secondary server receive data that is currently committed on the secondary server. See Design data replication group clients for additional information about design considerations for clients that connect to database servers that are running data replication. The default isolation level on secondary servers is Dirty Read; however, setting an explicit isolation level enables the correct isolation level: Dirty Read, Committed Read, or Committed Read Last Committed.

Repeatable Read and Cursor Stability isolation levels are not supported. Using the SET ISOLATION statement with Cursor Stability and Repeatable Read levels is ignored.

After starting a secondary server, client applications connect to the server only when all transactions open at the startup checkpoint have either committed or rolled back.

If the UPDATABLE_SECONDARY configuration parameter is disabled (by being unset or being set to zero), a secondary data replication server is read-only. In this case, only the DIRTY READ or READ UNCOMMITTED transaction isolation levels are available on secondary servers.

If the UPDATABLE_SECONDARY parameter is enabled (by setting it to a valid number of connections greater than zero), a secondary data replication server can support the COMMITTED READ, COMMITTED READ LAST COMMITTED, or COMMITTED READ transaction isolation level, or the USELASTCOMMITTED session environment variable. Only DML statements of SQL (the DELETE, INSERT, UPDATE, and MERGE statements), and the **dbexport** utility, can support write operations on an updatable secondary server. (Besides UPDATABLE_SECONDARY, the STOP_APPLY and USELASTCOMMITTED configuration parameters must also be set to enable write operations by **dbexport** on a secondary data replication server.)

Use **onstat -g ses** or **onstat -g sql** to view isolation level settings. See the *IBM® Informix® Administrator's Reference* for more information.

- Set lock mode

Set lock mode

Issuing a SET LOCK MODE TO WAIT or SET LOCK MODE TO WAIT *n* statement on a secondary server sets the lock wait timeout value for that session just like on a primary server. The value set by SET LOCK MODE is used by the proxy thread created for the current session on the primary server when it performs updates from a secondary server. If the value for SET LOCK MODE is greater than the ONCONFIG parameter value of DEADLOCK_TIMEOUT, the value of DEADLOCK_TIMEOUT is used instead.

Transient types on high-availability cluster secondary servers

Transient unnamed complex data types (ROW, SET, LIST, and MULTISET) can be used on high-availability cluster secondary servers, whether the secondary servers are read-only or updatable. The following types of operations that use transient types are supported on secondary servers:

- SQL queries that use transient types
- SQL queries that use derived tables, collection subqueries, and XML functions (these statements implicitly use transient types)
- Temporary tables created with the CREATE TEMP statement that uses transient types

See the *IBM® Informix® Guide to SQL: Reference* and *IBM Informix Guide to SQL: Syntax* for information about complex data types.

Row versioning

Use row versioning to determine whether a row was changed and to detect collisions. With row versioning enabled, each row of a table is configured to contain both a checksum and a version number. When a row is first inserted, the checksum is generated automatically, and the version is set to 1. Every time the row is updated the version is incremented by one, while the checksum value is not changed. With row versioning, if a row is deleted and another row is reinserted in a table, it is possible to recognize that the row is different. By comparing the row checksum and row version between the secondary and the primary servers, it is possible to detect data collisions.

Web applications can use a version column to determine whether information contained in a previously retrieved object is still current. For example, a web application might display items for sale to a customer. When the customer decides to purchase an item, the application can check the version column of the item's row to determine whether any information about the item has changed.

If client applications can update data on the secondary servers in your environment, use row versioning to minimize network use, especially if your tables have many columns. Otherwise, entire rows on the secondary server are compared with entire rows on the primary server to determine whether updates occurred.

To add row versioning to an existing table, use the following syntax:

```
ALTER TABLE tablename add VERCOLS;
```

Similarly, you can delete row versioning from a table with the following syntax:

```
ALTER TABLE tablename drop VERCOLS;
```

To create a new table with row versioning, use the following syntax:

```
CREATE TABLE tablename (
   Column Name   Datatype
   Column Name   Datatype
   Column Name   Datatype
   ) with VERCOLS;
```

When row versioning is enabled, **ifx_row_version** is incremented by one each time the row is updated; however, row updates made by Enterprise Replication do not increment the row version. To update the row version on a server using Enterprise Replication, you must include the **ifx_row_version** column in the replicate participant definition.

Backup and restore with high-availability clusters

You cannot perform most backup and restore operations on secondary servers.

Before you can establish a server as an HDR (high-availability data replication) or RS (remote stand-alone) secondary server, you must perform a cold restore on it.

After you have set up a server as an HDR or RSS secondary server, you can only perform the following backup and restore operations:

- You can perform a logical restore on HDR or RS secondary servers when you are setting up a high-availability cluster.
- You can perform an external backup on an RS secondary server. For more information, see the *IBM® Informix® Backup and Restore Guide*.

SD secondary servers must be shut down during a cold restore of the primary server, but can be online during a warm restore, after they have been shut down and restarted.

Related tasks:
Starting an RS secondary server for the first time
Starting HDR for the First Time
Recovering a shared-disk cluster after data is damaged

Change the database server mode

If you change the mode of a database server in a high-availability cluster, replication stops.

To change the database server mode, use the **onmode** utility.

The following table summarizes the effects of changing the mode of the primary database server.

Table 1. Mode changes on the primary database server

On the primary	On the secondary	To restart HDR
From any mode to offline: (**onmode -k**)	Secondary displays: DR: Receive error. HDR is turned off. The mode remains read-only. If DRAUTO is set to 0, the mode remains read-only. If DRAUTO is set to 1, the secondary server switches to standard type and can accept updates. (If DRAUTO is set to 2 , the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails.)	Treat it like a failure of the primary. Two different scenarios are possible, depending on what you do with the secondary database server while the primary database server is offline. See these sections for information: • The secondary server was not changed to the primary server • The secondary server was changed to the primary server automatically See Restart if the primary server fails.
To online, quiescent, or administration mode: (**onmode -s** / **onmode -u**) (**onmode -j**)	Secondary does not receive errors. HDR remains on. Mode remains read-only.	Use **onmode -m** on the primary.

The following table summarizes the effects of changing the mode of the secondary database server.

Table 2. Mode changes on the secondary database server

On the secondary	On the primary	To restart HDR
Read-only offline (**onmode -k**)	Primary displays: DR: Receive error. HDR is turned off.	Treat it as you would a failure of the secondary. Follow the procedures in Restarting HDR or RS clusters if the secondary server fails.

On the secondary	On the primary	To restart HDR
Quiescent (**onmode -s**)	Primary does not receive errors. HDR remains on.	Use **onmode -m** on the secondary.

Administration mode operates the same way on an HDR secondary database server as it does on the primary database server.

Changing the database server type

You can change the type of either the primary or the secondary database server.

You can change the type of either the primary or the secondary database server.

You can change the database server type from secondary to standard only if HDR is turned off on the secondary database server. HDR is turned off when the data replication connection to the primary database server breaks or data replication fails on the secondary database server. When you take the standard database server offline and bring it back online, it does not attempt to connect to the other database server in the replication pair.

Use the following commands to switch the type:

- hdrmksec.[sh|bat] and hdrmkpri.[sh|bat] scripts

To switch the database server type using hdrmkpri and hdrmksec scripts:

1. Shut down the primary database server (**ServerA**): **onmode -ky**
2. With the secondary database server (**ServerB**) online, run the hdrmkpri.sh script on UNIX or hdrmkpri.bat script on Windows.Now **ServerB** is a primary database server.
3. For **ServerA**, run the hdrmksec.sh script on UNIX or hdrmksec.bat script on Windows.Now **ServerA** is a secondary database server.
4. Bring **ServerB** (primary database server) online.

The following commands can also be used to switch the server type:

1. Change **ServerA** to the primary server by running the following command:

   ```
   onmode -d make primary ServerA
   ```

   This command makes **ServerA** the primary server, and redirects any other secondary servers in the cluster to point to the new primary server. The command also shuts down the old HDR primary (**ServerB**) because only a single primary server can exist in a high-availability environment.
2. Initialize **ServerB** as the HDR secondary server by running the following command:
   - On UNIX systems:

     ```
     $INFORMIXDIR/bin/hdrmksec.sh ServerB
     ```

   - On Windows systems:

     ```
     hdrmksec.bat ServerB
     ```

Related concepts:
Manual switchover

Prevent blocking checkpoints on HDR servers

On an HDR secondary server, checkpoint processing must wait until the flushing of buffer pools is complete. You can configure non-blocking checkpoints on an HDR secondary server so that log data sent from the primary server is stored, or *staged*, in a directory until checkpoint processing is complete.

You configure non-blocking checkpoints on an HDR secondary server by setting the LOG_STAGING_DIR and LOG_INDEX_BUILDS configuration parameters. When non-blocking checkpoints is configured, log data sent from the primary server is staged in a directory specified by the LOG_STAGING_DIR configuration parameter. When the HDR secondary server finishes processing the checkpoint it reads and applies the log data stored in the staging area. When the staging directory is empty the HDR secondary server reads and applies log data as it is received from the primary server.

You enable non-blocking checkpoints by setting the LOG_STAGING_DIR configuration parameter on the HDR secondary server, and LOG_INDEX_BUILDS on both the primary server and the HDR secondary server. The value for LOG_INDEX_BUILDS must be the same on both the primary server and the HDR secondary server.

When the HDR secondary server encounters a checkpoint, it enters a *buffering* mode. While in buffering mode, the secondary server stages any log page data from the primary server into files in the staging directory.

When the HDR secondary server completes checkpoint processing, the server enters a *drain* mode. In this mode, the HDR secondary server reads data from the staging file and also receives new data from the primary server. After the staging area is empty, the HDR secondary server resumes normal operation.

Where log records are stored on the HDR server

The HDR secondary server creates additional directories named ifmxhdrstage_## in the directory specified by LOG_STAGING_DIR, where **##** is the instance specified by SERVERNUM. The directories are used to store the logical logs sent from the primary server during checkpoint processing. The files within ifmxhdrstage_## are purged when no longer required.

Interaction of non-blocking checkpoints with secondary server updates

You must consider the interaction between secondary server updates and non-blocking checkpoints on HDR secondary servers. If the HDR secondary server receives an update request, the updates are not applied until the HDR secondary server processes the corresponding log records. When non-blocking checkpoints are enabled on the HDR secondary server, a delay in the application of data on the secondary server might occur because log data is staged at the secondary server due to checkpoint processing.

- View statistics for nonblocking checkpoints on HDR servers

View statistics for nonblocking checkpoints on HDR servers

You use the **onstat** utility to view information about nonblocking checkpoints on primary servers and on HDR secondary servers.

To view information about staged logs, use the **onstat -g dri ckpt** command.

For an example of **onstat -g dri ckpt** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

Monitor HDR status

Monitor the HDR status of a database server to determine the following information:

- The database server type (primary, secondary, or standard)
- The name of the other database server in the pair
- Whether HDR is on
- The values of the HDR parameters

- Command-line utilities
- SMI tables
  The **sysdri** table provides information about the High-Availability Data-Replication status of the database server.

Command-line utilities

The header information displayed every time you run **onstat** has a field to indicate if a database server is operating as a primary or secondary database server.

The following example shows a header for a database server that is the primary database server in a replication pair, and in online mode:

```
IBM Informix Dynamic Server Version 11.50.UC1   -- online(Prim) -- Up 45:08:57
```

This example shows a database server that is the secondary database server in a replication pair, and in read-only mode.

```
IBM Informix Dynamic Server Version 11.50.UC1     -- Read-Only (Sec) -- Up 45:08:57
```

The following example shows a header for a database server that is not involved in HDR. The type for this database server is standard.

```
IBM Informix Dynamic Server Version 11.50.UC1     -- online -- Up 20:10:57
```

- The onstat -g dri option
- The oncheck -pr option

The onstat -g dri option

To obtain full HDR monitoring information, run the **onstat -g dri** option. The following fields are displayed:

- The database server type (primary, secondary, or standard)
- The HDR state (on or off)
- The paired database server
- The last HDR checkpoint
- The values of the HDR configuration parameters

For an example of **onstat -g dri** output, see the *IBM® Informix® Administrator's Reference*.

The oncheck -pr option

If your database server is running HDR, the reserved pages PAGE_1ARCH and PAGE_2ARCH store the checkpoint information that HDR uses to synchronize the primary and secondary database servers. An example of the relevant **oncheck -pr** output is given in the following example.

```
Validating Informix Database Server reserved pages - PAGE_1ARCH &
 PAGE_2ARCH
          Using archive page PAGE_1ARCH.

    Archive Level                 0
    Real Time Archive Began       01/11/95 16:54:07
    Time Stamp Archive Began      11913
    Logical Log Unique Id         3
    Logical Log Position          b018

    DR Ckpt Logical Log Id        3
    DR Ckpt Logical Log Pos       80018
    DR Last Logical Log Id        3
    DR Last Logical Log Page      128
```


SMI tables

The **sysdri** table provides information about the High-Availability Data-Replication status of the database server.

The **sysdri** table, described in these topics on the **sysmaster** database in the *IBM® Informix® Administrator's Reference*, contains the following columns.

type
: HDR server type

state
: HDR server state

name
: Database server name

intvl
: HDR buffer flush interval

timeout
: Network timeout

lostfound
: HDR lost-and-found path name

Obtain RS secondary server statistics

Use the **onstat** command to display information about the state of RS secondary servers. To display the number of RS secondary servers, information about the data that has been sent to the RS secondary servers, and information about what the RS secondary servers have acknowledged receiving, use the **onstat -g rss** command.

This command has options to show extended information about a single server or multiple secondary servers. For examples of **onstat -g rss** output, see information about the **onstat** utility in the *IBM® Informix® Administrator's Reference*.

Remove an RS secondary server

Remove an RS secondary server from a high-availability cluster by issuing the following command:

```
onmode -d delete RSS rss_servername
```


RS secondary server security

RS secondary servers support similar encryption rules as HDR. See Encrypting data traffic between HDR database servers for details.

See Configure SMX connections for additional information about setting up and configuring encryption between servers and RS secondary servers.

- Create or change a password on an RS secondary server
  You can create a password for an RS secondary server to provide authentication between the primary server and the secondary server when the cluster is established. The password is optional. The password is valid only the first time the primary and secondary connect to each other.

Create or change a password on an RS secondary server

You can create a password for an RS secondary server to provide authentication between the primary server and the secondary server when the cluster is established. The password is optional. The password is valid only the first time the primary and secondary connect to each other.

The password prevents an unwanted instance on the network from initializing and accepting transaction data from the primary. The password is independent of the **informix** user password.

You create the password with the same command that you use to specify the RS secondary server name. The name and password of each RS secondary server can be defined either before or after the level-0 backup of the primary server.

To set the RS secondary server name and password, run the **onmode -d add RSS** *rss_ha_alias password* command on the primary server. During secondary server setup, you include the password when you run the **onmode -d RSS** *rss_ha_alias password* command on the secondary server.

You can change the password only if the server is not connected to a high availability environment. Attempting to change the password of an RS secondary server that is already connected returns an error.

To change the password on an RS secondary server before the server is connected, use the **onmode -d change RSS** *password* command on the primary server.

Examples

The following commands establish an RS secondary server named **ServerB** using a password of `s#cure`. The server's HA_ALIAS configuration parameter is set to `ServerB`.

On the primary server:

```
onmode -d add RSS ServerB s#cure
```

On the secondary server:

```
onmode -d RSS ServerB s#cure
```

The following command changes the password of the RS secondary server named **ServerB** that is not connected to the primary to `s@fest`. The server's HA_ALIAS configuration parameter is set to `ServerB`.

```
onmode -d change RSS ServerB s@fest
```


Transaction completion during cluster failover

You can configure servers in a high-availability cluster environment to continue processing transactions after failover of the primary server.

Transactions running on any server except the failed primary server continue to run. Configure the cluster environment for the following results:

- Transactions running on secondary servers are not affected.
- Transactions running on the secondary server that becomes the primary server are not affected.
- Transactions running on the failed primary server are terminated.

Transaction completion after failover is not supported for smart large objects, XA transactions, and when running DDL statements on secondary servers.

When a failover occurs, the secondary servers in the cluster temporarily suspend running user transactions until the new primary server is running. After failover, the secondary servers resend the saved transactions to the new primary server. The new primary server resumes execution of the transactions from the surviving secondary servers.

When distributed transactions (transactions that span multiple database servers) are running, any transaction that is running on the primary server at the time of server failure is terminated.

When failover occurs, whether it is manual or performed by the Connection Manager, the database server that receives failover must have the most advanced log-replay position of all active servers in the cluster. If the database server that receives failover does not have the most advanced log replay position, all transactions in the cluster are terminated and rolled back. Because the primary and SD secondary server read from the same physical disk, failover to an SD secondary server should occur first. If the failover server is an HDR secondary server, SD secondary servers are shut down.

For best Connection Manager failover performance, use the FOC ORDER=ENABLED setting in the Connection Manager configuration file, and set the HA_FOC_ORDER configuration parameter on the cluster's primary server.

- Configuring the server so that transactions complete after failover
  Use the FAILOVER_TX_TIMEOUT configuration parameter to configure the servers in a high-availability cluster so that transactions complete after failover.

Configuring the server so that transactions complete after failover

Use the FAILOVER_TX_TIMEOUT configuration parameter to configure the servers in a high-availability cluster so that transactions complete after failover.

The value of FAILOVER_TX_TIMEOUT indicates the maximum number of seconds the server waits before rolling back transactions after failure of the primary server. Set FAILOVER_TX_TIMEOUT to the same value on all servers in the cluster. For example, to specify 20 seconds for transaction completion, set the value of the

FAILOVER_TX_TIMEOUT configuration parameter in the onconfig file to 20.

To disable transaction completion after failover, set the FAILOVER_TX_TIMEOUT configuration parameter to 0 on all servers in the cluster.

Related information:
FAILOVER_TX_TIMEOUT configuration parameter

Connection management through the Connection Manager

Connection Managers can control automatic failover for high-availability clusters, monitor client connections and direct requests to appropriate database servers, act as proxy servers and handle client/server communication, and prioritize connections between application servers and the primary server of a high-availability cluster. Connection Managers support high-availability clusters, replicate sets, server sets, and grids.

Automatic failover for database servers

If the Connection Manager detects that a primary server of a high-availability cluster has failed, it can promote a secondary server to the role of the primary server.

If you use multiple Connection Managers to manager failover for a cluster, you can enforce a consistent failover policy by setting the onconfig file HA_FOC_ORDER configuration parameter on the cluster's primary server. The value of the onconfig file HA_FOC_ORDER configuration parameter replaces the value of the FOC parameter's ORDER attribute in the configuration file of each Connection Manager that connects to the primary server.

Rule-based connection redirection and load balancing

Client applications can connect to a Connection Manager as if they are connecting to a database server. The Connection Manager gathers workload statistics from each server in the connection unit and uses service level agreements (SLAs) to manage and direct client connection requests to appropriate servers. When a client application makes a connection request through a redirect-mode SLA, the Connection Manager returns a database server's IP address and port number to the client application. The client application then uses the information to connect to the specified database server.

Connection Manager redirection takes place in the communication layer, so additional action is not required by client applications. Connection Managers can use redirection policies that are based on workload, latency, apply failures, or the apply backlog. Redirection can also be configured to occur round-robin.

Redirection-policy SLAs do not support connections from application that are compiled with Data Server Driver for JDBC and SQLJ version 3.5.1 or before, or with 3.00 or before.

Proxy-server connection management

Proxy-mode SLAs and redirect-mode SLAs are similar; in both cases, the Connection Manager gathers workload statistics from connection-unit servers, and controls which servers receive client connection request servers. When a client application makes a connection request through a proxy-mode SLA, client/server communication travels through the Connection Manager. When a database server is behind a firewall, Connection Managers can act as proxy servers, and handle client/server communication.

Proxy-policy SLAs do not have the same version restrictions that redirect-policy SLAs have. Connections from application that are compiled with any version of Data Server Driver for JDBC and SQLJ, or with any version of are supported.

Failover prioritization for application servers

You can install Connection Managers on the same hosts as application servers, and then prioritize the connections between each application server and the primary server of a high-availability cluster. This can help the highest priority application server maintain a connection to the cluster's primary server if a portion of the network fails.

- Configuring connection management
  To configure connection management, you must install software, set environments and connectivity information, create Connection Manager configuration files, and run the **oncmsm** utility.
- Monitoring and troubleshooting connection management
  Tools are available to monitor connection management, and help you diagnose potential problems.
- Strategies for increasing availability with Connection Managers
  You can increase the resiliency of your client/server communication environment.
- Configuration examples for connection management
  The following examples show steps for setting up connection management for various connection units and various systems.

Related concepts:
Redirection and connectivity for data-replication clients
Failover configuration for high-availability clusters
Overview of DRDA
Components supporting high availability and scalability

Configuring connection management

To configure connection management, you must install software, set environments and connectivity information, create Connection Manager configuration files, and run the **oncmsm** utility.

To configure and start connection management, complete the following steps:

1. Install at least one Connection Manager as part of the installation.
    a. If Connection Managers are installed on hosts where database servers are not installed, set each Connection Manager's host INFORMIXDIR environment variable to the directory the Connection Manager is installed into.
2. Modify connectivity information in the sqlhosts files that are on all client, database server, and Connection Manager hosts.
    a. If sqlhosts files are in host directories other than $INFORMIXDIR/etc, set host INFORMIXSQLHOSTS environment variables to the appropriate sqlhosts file location.
    b. If Connection Managers or clients are installed on hosts where database servers are not installed, create a sqlhosts file on each host, and then set each Connection Manager's or client's host INFORMIXSQLHOSTS environment variable to the location of the sqlhosts file.
3. If Connection Managers, application servers, or database servers are on an untrusted network, complete the following steps:
    a. Create a password file.
    b. Encrypt the password file by running the **onpassword** utility.
    c. Distribute the password file to the database servers that Connection Managers connect to. If the database servers are on other operating systems, distribute the unencrypted password file, and then encrypt it on the other operating systems.
4. On Connection Manager hosts, create a configuration file for each installed Connection Manager.
    a. If the configuration file is in a directory other than $INFORMIXDIR/etc, set the CMCONFIG environment variable to specify the file's location.
    b. If you define a service-level agreement that uses a transaction-latency or apply-failure redirection policy, start quality of data (QOD) monitoring by running the **cdr define qod** and **cdr start qod** commands.
5. Set onconfig parameters on the cluster database servers.
    a. If Connection Managers control failover for a high-availability cluster, set the DRAUTO configuration parameter to `3` on all managed cluster database servers, and set the HA_FOC_ORDER configuration parameter on the primary server of each cluster to a failover order.
    b. On each database server that uses multiple ports, set the DBSERVERALIASES configuration parameter to the alias names listed in Connection Manager and database server sqlhosts file entries.
6. Optional: Configure the cmalarmprogram script that is installed with Connection Managers.
7. Run the **oncmsm** utility to start Connection Managers.

- Creating Connection Manager configuration files
- Configuring environments and setting configuration parameters for connection management
  Before you start a Connection Manager, you must configure its environment.
- Defining sqlhosts information for connection management
  You must define sqlhosts network-connectivity information for client applications that connect to Connection Managers, Connection Managers that connect to database servers, and database servers that are part of a Connection Manager connection unit.
- Creating a password file for connecting to database servers on untrusted networks
  If a client, Connection Manager, or any of the database servers that a Connection Manager connects to are on an untrusted network, you can create encrypted password files to verify connection requests.
- Starting Connection Managers on UNIX and Linux
  Use the **oncmsm** utility to start a Connection Manager.
- Starting Connection Managers on Windows
  Use the **oncmsm** utility to start a Connection Manager.
- Stopping connection management
  When you no longer want a Connection Manager to manage connection units, run the **oncmsm** utility to stop the Connection Manager instance.

Related concepts:
The sqlhosts information
Related information:
The oncmsm utility
The onpassword utility
DBSERVERALIASES configuration parameter
DRAUTO configuration parameter
HA_FOC_ORDER configuration parameter
INFORMIXSQLHOSTS environment variable
INFORMIXDIR environment variable
cdr define qod
cdr start qod

Creating Connection Manager configuration files

To configure a Connection Manager, you must create a configuration file, and then load the configuration file by running the **oncmsm** utility.
A Connection Manager configuration file consists of two parts:

- The header, which contains Connection Manager parameters that are specific to the Connection Manager.
- The body, which consists of one or more connection unit sections that contain parameters and attributes that are specific to the defined connection units.

The following steps apply to all connection-unit types:

1. Create an ASCII text file in the $INFORMIXDIR/etc directory of the host the Connection Manager is installed on.
2. On the first line of the file, specify the NAME parameter, followed by a name for the Connection Manager. Connection Manager names must be unique in the domain of the connection units that are managed. For example:

```
NAME my_connection_manager_1
```

3. Specify optional header parameters. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log
```

4. Create the body of the configuration file by specifying at least one connection unit type followed by the name of the connection unit, and then opening and closing braces. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
}
```

5. Set the connection unit's INFORMIXSERVER parameter to the sqlhosts file entries for connection-unit participants. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
    INFORMIXSERVER group_name
}
```

6. If the Connection Manager manages connection requests, specify SLA parameters, SLA names, and DBSERVER attributes. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
   INFORMIXSERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=HDR,SDS,RSS
}
```

7. Specify the FOC parameter and PRIORITY attribute. If the Connection Manager manages failover, specify the ORDER attribute, as well. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
   INFORMIXSERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=HDR,SDS,RSS
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

8. Specify optional SLA parameter attributes. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
   INFORMIXSERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=(HDR,SDS,RSS) \
             POLICY=ROUNDROBIN
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

9. If the Connection Manager manages more than one connection unit, add the other connection units to the body of the configuration file. For example:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
   INFORMIXSERVER group_name
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=(HDR,SDS,RSS) \
             POLICY=ROUNDROBIN
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_2
{
   INFORMIXSERVER group_name_2
   SLA sla_3 DBSERVERS=PRI
   SLA sla_4 DBSERVERS=(HDR,SDS) \
             POLICY=ROUNDROBIN
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

- Parameters and format of the Connection Manager configuration file
  The following example shows the format of the Connection Manager configuration file, and shows which parameters and attributes can be set for each connection-unit type.

- Modifying Connection Manager configuration files
  Use the **oncmsm** utility to load a modified configuration file into a Connection Manager and change the Connection Manager's configuration.
- Converting older formats of the Connection Manager configuration file to the current format
  The Connection Manager configuration file in versions of before version 3.70.xC3 are incompatible with the current version of the Connection Manager. You must convert configuration files from versions before 3.70.xC3 by running the **oncmsm** utility.

Parameters and format of the Connection Manager configuration file

The following example shows the format of the Connection Manager configuration file, and shows which parameters and attributes can be set for each connection-unit type.

```
#****************************** HEADER ******************************

NAME connection_manager_instance_name

# Optional Parameters
MACRO name_1=value
MACRO name_2=value
MACRO name_n=value_n
.
.
.
LOCAL_IP ip_address_list
LOG value
LOGFILE path_and_filename
DEBUG value
CM_TIMEOUT seconds
EVENT_TIMEOUT seconds
SECONDARY_EVENT_TIMEOUT seconds
SQLHOSTS value
SSL_LABEL certificate_name

#****************************** BODY ******************************

# Replicate set connection-unit example

REPLSET unit_name_1
{
   INFORMIXSERVER sqlhosts_group_names_list

   #Optional Parameters and Attributes
   SLA sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA sla_name_2 DBSERVERS=value_list ...
   SLA sla_name_n DBSERVERS=value_list ...
   .
   .
   .
}

# High-availability cluster connection-unit example

CLUSTER unit_name_2
{
   INFORMIXSERVER sqlhosts_group_name
   FOC ORDER=value \
       PRIORITY=value \
       TIMEOUT=value

   #Optional Parameters and Attributes
   SLA sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA sla_name_2 DBSERVERS=value_list ...
   SLA sla_name_n DBSERVERS=value_list ...
.
.
.
   CMALARMPROGRAM path_and_filename
}
```

```
# Server set connection-unit example

SERVERSET unit_name_3
{
   INFORMIXSERVER sqlhosts_group_names_and_standalone_servers_list

   #Optional Parameter and Attributes
   SLA sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA sla_name_2 DBSERVERS=value_list ...
   SLA sla_name_n DBSERVERS=value_list ...
   .
   .
   .
}

# Grid connection-unit example

GRID unit_name_4
{
   INFORMIXSERVER server_list

   #Optional Parameter and Attributes
   SLA sla_name_1 DBSERVERS=value_list \

                  #Optional SLA Attributes
                  MODE=value \
                  USEALIASES=value \
                  POLICY=value \
                  WORKERS=number_of_threads \
                  HOST=host_name \
                  NETTYPE=network_protocol \
                  SERVICE=service_name \
                  SQLHOSTSOPT="options"
   SLA sla_name_2 DBSERVERS=value_list ...
   SLA sla_name_n DBSERVERS=value_list ...
   .
   .
   .
}

# Connection Unit n
connection_unit_type unit_name_n
{
   INFORMIXSERVER values
   .
   .
   .
}
#*************************************************************************
```

Tip: For increased readability, break long configuration-file lines by using a backslash (\ ) line-continuation character.
The following example shows a macro definition that uses two text-file lines, but is read as a single line:

```
MACRO servers=node1,node2,node3,node4,node5,node6,node7,node8, \
              node9,node10,node11,node12,node13,node14,node15
```

- CMALARMPROGRAM Connection Manager configuration parameter
  The CMALARMPROGRAM parameter specifies the path and file name of a program or script to run if failover processing encounters an error.
- CM_TIMEOUT Connection Manager configuration parameter
  The CM_TIMEOUT parameter specifies the number of seconds that a cluster of database servers waits to receive events from the failover-arbitrator Connection Manager. If the specified period elapses with no events received by the cluster, the primary server promotes the Connection Manager with the highest priority to the role of failover arbitrator.
- CLUSTER Connection Manager configuration parameter
  The CLUSTER parameter specifies that a connection unit is composed of a high-availability cluster, and specifies a name for that connection unit. A high-availability cluster is a primary server plus one or more secondary servers, which can be high-availability data replication secondary servers, shared-disk secondary servers, or remote stand-alone secondary servers.
- DEBUG Connection Manager configuration parameter
  The DEBUG parameter specifies whether logging of SQL and ESQL/C error messages is enabled or disabled.
- EVENT_TIMEOUT Connection Manager configuration parameter
  The EVENT_TIMEOUT parameter specifies the number of seconds that must elapse with no primary-server events before the active-arbiter Connection Manager starts failover processing. The Connection Manager waits for primary-server events or notifications from secondary servers that the primary server is offline. A primary-server event is an indication from the primary server that the server is still functioning, such as a sent performance-statistics or administration messages, or node-changes.
- FOC Connection Manager configuration parameter
  The FOC parameter and attributes specify the failover configuration for high-availability clusters, and specify the priority of the connection between the Connection Manager and the primary server.
- GRID Connection Manager configuration parameter
  The GRID parameter specifies that a connection unit is an Enterprise Replication grid, and specifies the name of the grid. A grid is a set of replication servers that you can administer as a unit.

- INFORMIXSERVER Connection Manager configuration parameter
  The INFORMIXSERVER parameter specifies database servers or Enterprise Replication nodes that the Connection Manager connects to after it starts. The values of the INFORMIXSERVER parameter are also used for providing database server information for service-level agreements when a Connection Manager's SQLHOSTS parameter is set to REMOTE or LOCAL+REMOTE.
- LOCAL_IP Connection Manager configuration parameter
  The LOCAL_IP parameter specifies IP addresses to monitor on the computer that is running the Connection Manager. The LOCAL_IP parameter is used with the FOC parameter's PRIORITY attribute to determine if database-failover occurs during a partial network failure.
- LOG Connection Manager configuration parameter
  The LOG parameter specifies logging for Connection Manager modes.
- LOGFILE Connection Manager configuration parameter
  The LOGFILE parameter specifies the name and location of the Connection Manager log file.
- MACRO Connection Manager configuration parameter
  The MACRO parameter specifies the name of a macro and a value that can be reused in the Connection Manager configuration file.
- NAME Connection Manager configuration parameter
  The NAME parameter specifies the name of the Connection Manager instance.
- REPLSET Connection Manager configuration parameter
  The REPLSET parameter specifies that a connection unit is an Enterprise Replication replicate set, and specifies the name of the replicate set. A replicate set combines several replicates to form a set that can be administered together as a unit.
- SECONDARY_EVENT_TIMEOUT Connection Manager configuration parameter
  The SECONDARY_EVENT_TIMEOUT parameter specifies the number of seconds that must elapse with no secondary-server events before the Connection Manager disconnects from a secondary server. A secondary-server event is an indication from a secondary server that the server is still functioning, such as a sent performance-statistics or administration messages.
- SERVERSET Connection Manager configuration parameter
  The SERVERSET parameter specifies that a connection unit is a server set, and specifies the name of the server set. A server set contains unrelated, standard servers that do not use replication or failover.
- SLA Connection Manager configuration parameter
  The SLA parameter defines service-level agreements that direct client requests to database servers.
- SQLHOSTS Connection Manager configuration parameter
  The SQLHOSTS parameter specifies where a Connection Manager can search for database servers that are specified by the INFORMIXSERVER parameter and DBSERVERS attribute.
- SSL_LABEL Connection Manager configuration parameter
  The SSL_LABEL parameter specifies the certificate label used for authentication by the CM when listening for an SSL connection.

CMALARMPROGRAM Connection Manager configuration parameter

The CMALARMPROGRAM parameter specifies the path and file name of a program or script to run if failover processing encounters an error.

Syntax

```
|--CMALARMPROGRAM--path_and_filename----------------------------|
```

Usage

The CMALARMPROGRAM parameter is optional, and is supported by the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

If the Connection Manager cannot find a server capable of receiving failover, it searches for ORDER-attribute servers at increasing intervals, up to 60 seconds, for a maximum of two days. If failover processing fails after eight attempts, the Connection Manager calls the program that is specified by the CMALARMPROGRAM parameter. The first eight failover attempts take approximately one minute.

Before you can use the cmalarmprogram script, you must edit the file, and set the script parameters.

The ALARMADMIN and ALARMPAGER parameters determine the level of Connection Manager event alarms that are sent to specified email addresses.

Table 1. Connection Manager event-alarm levels

Level of Connection Manager event alarm	Alarm type
0 (default)	None
1	Unimportant informational alarms
2	Informational alarms
3	Alarms requiring attention
4	Emergency alarms
5	Fatal error alarms

Example

In the following example, cmalarmprogram.sh is called if failover processing fails after eight attempts:

```
CMALARMPROGRAM $INFORMIXDIR/etc/cmalarmprogram.sh
```

Related reference:
FOC Connection Manager configuration parameter

CM_TIMEOUT Connection Manager configuration parameter

The CM_TIMEOUT parameter specifies the number of seconds that a cluster of database servers waits to receive events from the failover-arbitrator Connection Manager. If the specified period elapses with no events received by the cluster, the primary server promotes the Connection Manager with the highest priority to the role of failover arbitrator.

Syntax

Usage

The CM_TIMEOUT parameter is optional, and applies to CLUSTER connection units.

If the CM_TIMEOUT parameter is not specified, the timeout is 60 seconds.

Example

In the following example, if 100 seconds elapse without the servers of a high-availability cluster receiving any events from the current failover arbiter, the primary server of the cluster promotes an active Connection Manager to the role of failover arbiter:

```
CM_TIMEOUT 100
```


CLUSTER Connection Manager configuration parameter

The CLUSTER parameter specifies that a connection unit is composed of a high-availability cluster, and specifies a name for that connection unit. A high-availability cluster is a primary server plus one or more secondary servers, which can be high-availability data replication secondary servers, shared-disk secondary servers, or remote stand-alone secondary servers.

Syntax

Usage

Each CLUSTER parameter value must be unique within the Connection Manager configuration file.

CLUSTER parameter values cannot use multibyte characters.

CLUSTER connection units can use the following redirection policies:

- Round-robin
- Secondary apply backlog
- Workload

Each CLUSTER connection-unit definition that references a specific high-availability cluster must have a unique PRIORITY attribute value. For example, the following high-availability cluster, composed of **server_1** and **server_2**, must have unique PRIORITY values in each definition.

my_connection_manager_1's configuration file:

```
NAME my_connection_manager_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_1.log

CLUSTER my_cluster
{
   INFORMIXSERVER server_1,server_2
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

my_connection_manager_2's configuration file:

```
NAME my_connection_manager_2
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm_2.log
```

```
CLUSTER my_cluster
{
   INFORMIXSERVER server_1,server_2
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Example 1: Specifying a high-availability cluster as a CLUSTER connection unit

In the following example, a high-availability cluster composed of the servers in **my_server_group** is specified as the CLUSTER connection unit **my_cluster**:

```
CLUSTER my_cluster
{
   INFORMIXSERVER my_server_group
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED
       PRIORITY=1
}
```


DEBUG Connection Manager configuration parameter

The DEBUG parameter specifies whether logging of SQL and ESQL/C error messages is enabled or disabled.

Syntax

Table 1. Values for the DEBUG Connection Manager configuration parameter

DEBUG parameter value	Description
1	Enables logging of SQL and ESQL/C error messages.
0 (default)	Disables logging of SQL and ESQL/C error messages.

Usage

The DEBUG parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

Debug messages are created in the location that is specified by the LOGFILE parameter in the Connection Manager configuration file. If the LOGFILE parameter is not specified, then the Connection Manager creates $INFORMIXDIR/tmp/*connection_manager_name.pid*.log.

Connection Manager debug logging is enabled in the configuration file; you cannot enable debug mode from the command line.

Example

In the following example, a my_log is created in $INFORMIXDIR/tmp and logging of SQL and ESQL/C error messages is enabled.

```
LOGFILE $INFORMIXDIR/tmp/my_log
DEBUG 1
```


EVENT_TIMEOUT Connection Manager configuration parameter

The EVENT_TIMEOUT parameter specifies the number of seconds that must elapse with no primary-server events before the active-arbiter Connection Manager starts failover processing. The Connection Manager waits for primary-server events or notifications from secondary servers that the primary server is offline. A primary-server event is an indication from the primary server that the server is still functioning, such as a sent performance-statistics or administration messages, or node-changes.

Syntax

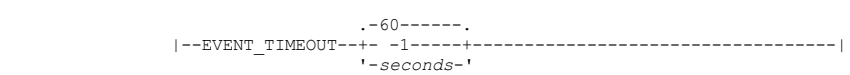


Table 1. Values for the EVENT_TIMEOUT Connection Manager configuration parameter

EVENT_TIMEOUT parameter value	Description
-1	Connection Manager waits indefinitely
0 to 30	Connection Manager waits 30 seconds.
> 30	Connection Manager waits the specified number of seconds.

Usage

The EVENT_TIMEOUT parameter is optional, and applies to CLUSTER connection units.

If EVENT_TIMEOUT is not specified in the Connection Manager configuration file, the Connection Manager waits 60 seconds for primary-server events or notifications from secondary servers that the primary server is offline before it starts failover processing.

Example

In the following example, the active-arbiter Connection Manager begins failover processing after 200 seconds elapse with no primary-server events.

```
EVENT_TIMEOUT 200
```

Related reference:
FOC Connection Manager configuration parameter

FOC Connection Manager configuration parameter

The FOC parameter and attributes specify the failover configuration for high-availability clusters, and specify the priority of the connection between the Connection Manager and the primary server.

Syntax

```
        .-ORDER=ENABLED--.
|--FOC--+-ORDER=DISABLED-+--PRIORITY=value----------------------->

>--+-----------------+-----------------------------------------|
   '-TIMEOUT=seconds-'
```

Attributes of the FOC Connection Manager configuration parameter

The FOC parameter is required by CLUSTER connection units.

Table 1. Attributes of the FOC parameter

Attribute of the FOC parameter	Description
ORDER	Disables automatic failover or specifies that the value of the primary server's HA_FOC_ORDER configuration parameter is used for automatic failover. If the ORDER attribute is not specified, the value of the primary server's HA_FOC_ORDER configuration parameter is used for automatic failover. If the ORDER attribute is not specified, and the primary server's HA_FOC_ORDER configuration parameter is not set, the failover order is SDS, HDR, and then RSS.
PRIORITY	Specifies the priority of connections between Connection Managers and the primary server of a cluster. The value must be a positive integer and unique among all Connection Manager CLUSTER units specified for a high-availability cluster. The lower the number, the higher the priority. For CLUSTER connection units, a PRIORITY value must be specified, even if ORDER is set to DISABLED.
TIMEOUT	Specifies the number of additional seconds the Connection Manager waits for primary-server events before the Connection Manager begins failover processing. The TIMEOUT attribute value applies after the EVENT_TIMEOUT parameter value is exceeded. For example, if the EVENT_TIMEOUT parameter is set to 60 and the TIMEOUT value is set to 10, 70 seconds of no primary server events must elapse before failover can begin. If the TIMEOUT attribute is not specified, failover begins immediately after the amount of time that is specified by the EVENT_TIMEOUT parameter value is exceeded.

Values for the ORDER attribute of the FOC Connection Manager configuration parameter

Table 2. Values for the ORDER attribute of the FOC Connection Manager configuration parameter.

ORDER attribute value	Description

ORDER attribute value	Description
ENABLED	Specifies that the Connection Manager can perform automatic failover. The Connection Manager uses the value of the primary server's onconfig file HA_FOC_ORDER configuration parameter to determine failover sequence. If the ORDER attribute is set to ENABLED, but the primary server's HA_FOC_ORDER configuration parameter is not specified, the failover order is SDS, HDR, and then RSS.

ORDER Usage

The ORDER attribute specifies if automatic failover is enabled. If failover cannot complete, the Connection Manager reattempts failover at increasing intervals, up to 60 seconds, for a maximum of two days.

To modify the number of seconds that must elapse before the active-arbiter Connection Manager starts failover processing, set the EVENT_TIMEOUT parameter.

If you configure failover, you can set the CMALARMPROGRAM parameter to specify a program or script to run if failover processing cannot complete after eight tries.

Important: If automatic failover is enabled, and failover processing begins, you must not manually restart the failed primary server until failover processing completes.

PRIORITY Usage

If you install Connection Managers on application-server hosts, you can use the PRIORITY attribute to prioritize the connections between the application servers and the primary server of a cluster.

If the network connection between a specific Connection Manager and primary database server fails, PRIORITY determines whether the Connection Manager starts failover. If failover would promote a database server that cannot communicate with an active, higher-priority Connection Manager, then failover does not occur. If failover would promote a database server that cannot communicate with an active, lower-priority Connection Manager, failover can occur.

Example 1: Configuring multiple Connection Managers for failover

In the following example, the HA_FOC_ORDER configuration parameter in the **my_primary_server_1** onconfig file is set:

```
HA_FOC_ORDER SDS,HDR,RSS
```

Three Connection Managers are installed and have the following configuration files:

cm_configuration_file_1

```
NAME my_connection_manager_1

CLUSTER my_cluster_1
{
   INFORMIXSERVER my_server_group_1
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=1
   CMALARMPROGRAM $INFORMIXDIR/etc/cmalarmprogram.sh
}
```

cm_configuration_file_2

```
NAME my_connection_manager_2

CLUSTER my_cluster_1
{
   INFORMIXSERVER my_server_group_1
   SLA sla_2 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=2
   CMALARMPROGRAM $INFORMIXDIR/etc/cmalarmprogram.sh
}
```

cm_configuration_file_3

```
NAME my_connection_manager_3

CLUSTER my_cluster_1
{
   INFORMIXSERVER my_server_group_1
   SLA sla_3 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=3
   CMALARMPROGRAM $INFORMIXDIR/etc/cmalarmprogram.sh
}
```

When the connection managers are initialized, they each search their hosts's sqlhosts file for a **my_server_group_1** entry and connect with the servers that are in the group. The value of the HA_FOC_ORDER configuration parameter in the onconfig file of the primary server in **my_server_group_1** is set for each of the Connection Manager, so that all the Connection Managers have a consistent failover policy. The value of the primary server's HA_FOC_ORDER configuration parameter replaces the values of the HA_FOC_ORDER configuration parameters in the onconfig files of all secondary servers in the cluster. This process maintains failover-order consistency if the primary server fails and then a Connection Manager restarts.

If a Connection Manager detects that the primary server failed, it first attempts to convert the most suitable SD secondary server to the primary server. If no SD secondary server is available, the Connection Manager attempts to convert the HDR secondary server into the primary server. If the HDR secondary server is not available, the Connection Manager attempts to convert the most suitable RS secondary server to the primary server.

If failover processing fails after eight attempts, the Connection Manager calls $INFORMIXDIR/etc/cmalarmprogram.sh.

Example 2: Configuring multiple Connection Managers to prioritize the connections between application servers and the primary server of a cluster

In the following example, the HA_FOC_ORDER configuration parameter in the **my_primary_server_2** onconfig file is set:

```
HA_FOC_ORDER HDR,RSS
```

The two Connection Managers, **my_important_app_cm** and **my_non_essential_app_cm**, are installed and configured:

cm_configuration_file_4

```
NAME important_app_cm
LOCAL_IP 192.0.2.0, 192.0.2.256

CLUSTER my_cluster
{
   INFORMIXSERVER my_server_group_1
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

cm_configuration_file_5

```
NAME non_essential_app_cm
LOCAL_IP 192.0.2.257, 192.0.2.258

CLUSTER my_cluster
{
   INFORMIXSERVER my_server_group_1,
   SLA sla_2 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

- The network state is:
- **important_app_cm** can connect to the primary server and RS secondary server, but cannot connect to the HDR secondary server.
- **non_essential_app_cm** can connect to the primary server and the HDR server, but cannot connect to the RS secondary server.

If the network between **non_essential_app_cm** and **my_primary_server** goes down, **non_essential_app_cm** attempts failover to the HDR secondary server. Because **important_app_cm** cannot connect to the HDR secondary server, and has higher priority than **non_essential_app_cm**, the failover attempt is blocked.

If, instead, the network between **important_app_cm** and **my_primary_server** goes down, **important_app_cm** attempts failover to the HDR secondary server. Because **important_app_cm** cannot connect to the HDR secondary server, it then attempts failover to the RS secondary server. **non_essential_app_cm** cannot connect to the RS secondary server, but it has a lower priority than **important_app_cm**, so failover occurs.

In both situations, the connection between **important_app_cm** and the primary server is prioritized over the connection between **non_essential_app_cm** and the primary server. Because the Connection Managers are on the application-server hosts, the connections between the application servers and the primary server are, essentially, prioritized.

Related tasks:
Example of configuring connection management for prioritizing connections and network monitoring
Related reference:
LOCAL_IP Connection Manager configuration parameter
CMALARMPROGRAM Connection Manager configuration parameter
EVENT_TIMEOUT Connection Manager configuration parameter
Related information:
HA_FOC_ORDER configuration parameter

GRID Connection Manager configuration parameter

The GRID parameter specifies that a connection unit is an Enterprise Replication grid, and specifies the name of the grid. A grid is a set of replication servers that you can administer as a unit.

Syntax

```
|--GRID--unit_name-----------------------------------------------|
```

Usage

Each connection-unit name must be unique within the Connection Manager configuration file.

GRID connection units can use the following redirection policies:

- Apply failure
- Round-robin
- Transaction latency
- Workload

Grid names can use multibyte characters.

Example

In the following example, the grid connection unit named **my_grid** is defined:

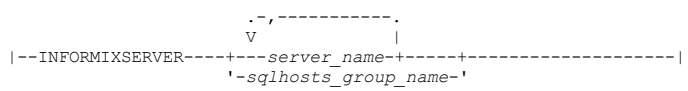
```
GRID my_grid
{
   INFORMIXSERVER my_server_group_1,my_server_group_2
   SLA sla_1 DBSERVERS=ANY
}
```


INFORMIXSERVER Connection Manager configuration parameter

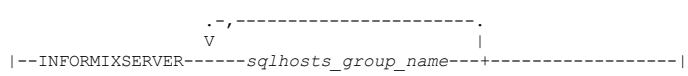
The INFORMIXSERVER parameter specifies database servers or Enterprise Replication nodes that the Connection Manager connects to after it starts. The values of the INFORMIXSERVER parameter are also used for providing database server information for service-level agreements when a Connection Manager's SQLHOSTS parameter is set to `REMOTE` or `LOCAL+REMOTE`.

Syntax

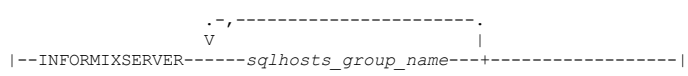
CLUSTER connection unit



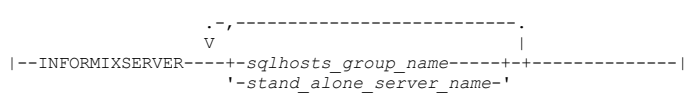
GRID connection unit



REPLSET connection unit



SERVERSET connection unit



Usage

The INFORMIXSERVER parameter is required, and is supported by the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

For Enterprise Replication domains, list group names for the nodes that receive client requests from the Connection Manager.

For server-set replication domains, list group names for the nodes that receive client requests from the Connection Manager. List standalone server names, as well.

For high-availability clusters, list the group name for the cluster, or list the names of database servers.

Example

In the following example, the Connection Manager connects to all database servers that are members of the **my_server_group** group in the Connection Manager sqlhosts file.

```
NAME my_connection_manager


CLUSTER my_cluster
{
   INFORMIXSERVER my_server_group
   FOC ORDER=ENABLED \
```

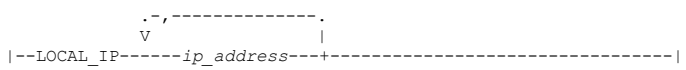
```
        PRIORITY=1
}
```

Related reference:
Group information

LOCAL_IP Connection Manager configuration parameter

The LOCAL_IP parameter specifies IP addresses to monitor on the computer that is running the Connection Manager. The LOCAL_IP parameter is used with the FOC parameter's PRIORITY attribute to determine if database-failover occurs during a partial network failure.

Syntax



Usage

The LOCAL_IP parameter is optional, and applies to CLUSTER connection units.

You must list the IP address of each network interface card to be monitored.

Example

In the following example, the Connection Manager monitors the network connection between its host and the primary server of a cluster through the network interface cards that have IP addresses of 192.0.2.0 and 192.0.2.1:

```
LOCAL_IP 192.0.2.0,192.0.2.1
```

Related tasks:
Example of configuring connection management for prioritizing connections and network monitoring
Related reference:
FOC Connection Manager configuration parameter

LOG Connection Manager configuration parameter

The LOG parameter specifies logging for Connection Manager modes.

Syntax

Table 1. Values for the LOG Connection Manager configuration parameter

LOG parameter value	Description
0 (default)	Specifies that logging is turned off.
1	The Connection Manager logs proxy-mode and redirect-mode SLA information.
2	The Connection Manager logs proxy-mode SLA information only. Data send-and-receive activities between clients and the Connection Manager is logged.
3	The Connection Manager logs proxy-mode SLA information only. Data content between clients and the Connection Manager is logged.

Usage

The LOG parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

Set the LOGFILE parameter to specify where log files are stored. If the LOGFILE parameter is not set, the Connection Manager creates a log file in the $INFORMIXDIR/tmp directory, with a name of *connection_manager_name.process_ID*.log.

Example

In the following example, the Connection Manager logs proxy-mode and redirect-mode SLA information to $INFORMIXDIR/tmp/my_cm.log.

```
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm.log
```


LOGFILE Connection Manager configuration parameter

The LOGFILE parameter specifies the name and location of the Connection Manager log file.

Syntax

```
|--LOGFILE--path_and_filename-----------------------------------|
```

Usage

The LOGFILE parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The path and file name of the log file display when the Connection Manager is started, and the log file is continuously updated with status information while the Connection Manager is running. If multiple Connection Managers are installed on the same host, specify a different path and file name for each Connection Manager.

Make sure that the directory for the log file exists, and that access to the file is enabled for the user that starts the Connection Manager.

If the LOGFILE parameter is not set, the Connection Manager creates a log file in the $INFORMIXDIR/tmp directory, with a name of *connection_manager_name.process_ID*.log.

Example

In the following example, the Connection Manager logs proxy-mode SLA information about data send-and-receive activities between clients and the Connection Manager. The information is logged to $INFORMIXDIR/tmp/my_cm.log.

```
LOG 2
LOGFILE $INFORMIXDIR/tmp/my_cm.log
```


MACRO Connection Manager configuration parameter

The MACRO parameter specifies the name of a macro and a value that can be reused in the Connection Manager configuration file.

Macro definition format

```
|--MACRO--name--=--value----------------------------------------|
```

Macro use format

```
|--${--macro_name--}----------------------------------------------|
```

Usage

The MACRO parameter is optional, and applies to the Connection Manager configuration file.

A macro can contain spaces, but not line breaks.

The MACRO parameter can be set multiple times to create multiple macros.

After a macro is defined, it can be used in other macros. For example:

```
MACRO WA=wa_server_1,wa_server_2,wa_server_3,wa_server_4
MACRO OR=or_server_1,or_server_2,or_server_3,or_server_4
MACRO ID=id_server_1,id_server_2,id_server_3,id_server_4
MACRO PNW=${WA),${OR),${ID)
```

Example 1: Using a macro in a service-level agreement

In the following example, the macro **CA** contains the names of eight servers.

```
NAME my_connection_manager_1
MACRO CA=ca_server_1,ca_server_2,ca_server_3,ca_server_4, \
         ca_server_5,ca_server_6,ca_server_7,ca_server_8

CLUSTER my_cluster_1
{
   INFORMIXSERVER group_name_1
   SLA sla_1 DBSERVERS=${CA}
   FOC ORDER=ENABLED PRIORITY=1
}
```

The macro expands in the following way:

```
{
   INFORMIXSERVER group_name_1
   SLA sla_1 DBSERVERS=ca_server_1,ca_server_2,ca_server_3,ca_server_4, \
         ca_server_5,ca_server_6,ca_server_7,ca_server_8
   FOC ORDER=ENABLED PRIORITY=1
}
```

Example 2: Using multiple macros in service-level agreements

In the following example, the macros **WA**, **OR**, and **ID** each contain the names of servers. Macro **ID** also contains parentheses to create a redirection-policy group.

```
NAME my_connection_manager_2
MACRO WA=wa_server_1,wa_server_2,wa_server_3
MACRO OR=or_server_1,or_server_2,or_server_3
MACRO ID=(id_server_1,id_server_2,id_server_3)

CLUSTER my_cluster_2
{
   INFORMIXSERVER group_name_2
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=${WA},${OR}
   SLA sla_3 DBSERVERS=${ID} POLICY=ROUNDROBIN
   FOC ORDER=ENABLED PRIORITY=1
}
```

The macros expand in the following ways:

```
{
   INFORMIXSERVER group_name_2
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=wa_server_1,wa_server_2,wa_server_3,or_server_1,or_server_2,or_server_3
   SLA sla_3 DBSERVERS=(id_server_1,id_server_2,id_server_3) POLICY=ROUNDROBIN
   FOC ORDER=ENABLED PRIORITY=1
}
```


NAME Connection Manager configuration parameter

The NAME parameter specifies the name of the Connection Manager instance.

Syntax

```
|--NAME--connection_manager_name--------------------------------|
```

Usage

The NAME parameter is required, and applies to the Connection Manager instance.

The name of the Connection Manager instance is necessary for monitoring, shutting down, or reloading the Connection Manager.

Connection Manager names must be unique in the domain of the connection units that are managed.

Example

In the following example, a Connection Manager instance is named **my_connection_manager**.

```
NAME my_connection_manager
```


REPLSET Connection Manager configuration parameter

The REPLSET parameter specifies that a connection unit is an Enterprise Replication replicate set, and specifies the name of the replicate set. A replicate set combines several replicates to form a set that can be administered together as a unit.

Syntax

```
|--REPLSET--unit_name-------------------------------------------|
```

Usage

Each connection-unit name must be unique within the Connection Manager configuration file.

REPLSET connection units can use the following redirection policies:

- Apply failure
- Round-robin
- Transaction latency
- Workload

Replicate set names can use multibyte characters.

Example

In the following example, the replicate-set connection unit named **my_replicate_set** is defined:

```
REPLSET my_replicate_set
{
   INFORMIXSERVER my_group_1,my_group_2
   SLA sla_1 DBSERVERS=ANY
}
```


SECONDARY_EVENT_TIMEOUT Connection Manager configuration parameter

The SECONDARY_EVENT_TIMEOUT parameter specifies the number of seconds that must elapse with no secondary-server events before the Connection Manager disconnects from a secondary server. A secondary-server event is an indication from a secondary server that the server is still functioning, such as a sent performance-statistics or administration messages.

Syntax

```
                          .-60------.
|--SECONDARY_EVENT_TIMEOUT--+- -1-----+-------------------------|
                          '-seconds-'
```

Table 1. Values for the SECONDARY_EVENT_TIMEOUT Connection Manager configuration parameter

SECONDARY_EVENT_TIMEOUT parameter value	Description
-1	The Connection Manager waits indefinitely
0 to 30	The Connection Manager waits 30 seconds.
> 30	The Connection Manager waits the specified number of seconds.

Usage

The SECONDARY_EVENT_TIMEOUT parameter is optional, and applies CLUSTER connection units.

If the SECONDARY_EVENT_TIMEOUT parameter is not specified in the Connection Manager configuration file, the Connection Manager waits 60 seconds for secondary-server events before it disconnects from the server.

Example

In the following example, the Connection Manager disconnects from a secondary server after 300 seconds elapse with no secondary-server events.

```
SECONDARY_EVENT_TIMEOUT 300
```


SERVERSET Connection Manager configuration parameter

The SERVERSET parameter specifies that a connection unit is a server set, and specifies the name of the server set. A server set contains unrelated, standard servers that do not use replication or failover.

Syntax

```
|--SERVERSET--unit_name-----------------------------------------|
```

Usage

Each connection-unit name must be unique within the Connection Manager configuration file.

SERVERLSET connection units can use the following redirection policies

- Round-robin
- Workload

Server set names cannot use multibyte characters.

Example

In the following example, the server-set connection unit named **my_server_set** is defined:

```
SERVERSET my_server_set
{
   INFORMIXSERVER my_group_1,my_group_2
   SLA sla_1 DBSERVERS=ANY
}
```


SLA Connection Manager configuration parameter

The SLA parameter defines service-level agreements that direct client requests to database servers.

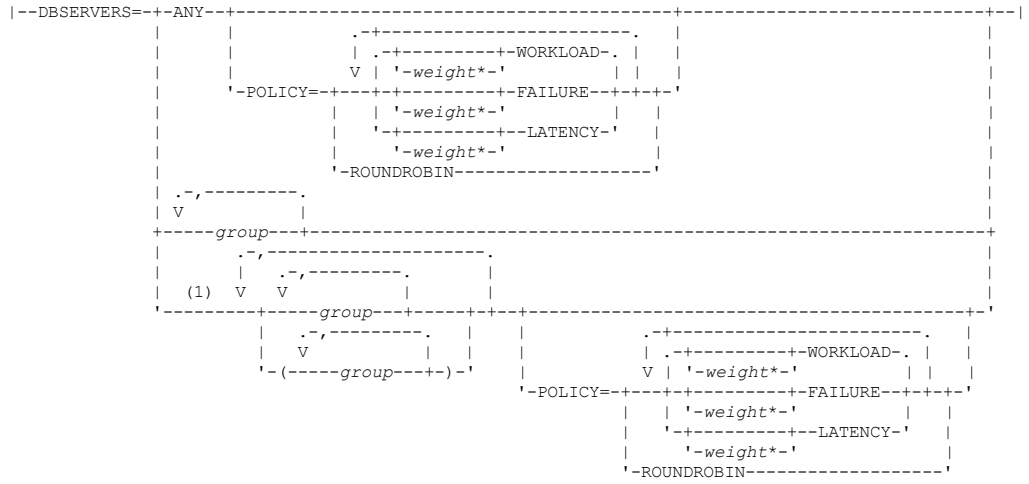
Syntax

```
>>-SLA--sla_name--+-| CLUSTER SLA syntax |---------+------------>
                 +-| GRID or REPLSET SLA syntax |-+
                 '-| SERVERSET SLA syntax |-------'

>--+---------------------+--+---------------------+-------------->
   |       .-REDIRECT-. |  |             .-ON--. |
   '-MODE=-+-PROXY----+-'  '-USEALIASES=-+-OFF-+-'

>--+---------------------------------+--------------------------->
   |          .-4-----------------. |
   '-WORKERS=-+-Number_of_threads-+-'

>--+----------------------+--+----------------------+---------->
   '-HOST=-+-host_name--+-'  '-NETTYPE=-+-drsocssl-+-'
           '-ip_address-'               +-drsoctcp-+
                                        +-onsocssl-+
                                        '-onsoctcp-'

>--+-----------------------------+------------------------------->
   '-SERVICE=-+-port_number--+-'
              '-service_name-'

>--+---------------------------------+--------------------------><
   |               .-,----------.    |
   |               V            |    |
   '-SQLHOSTSOPT="-----option---+-"-'
```

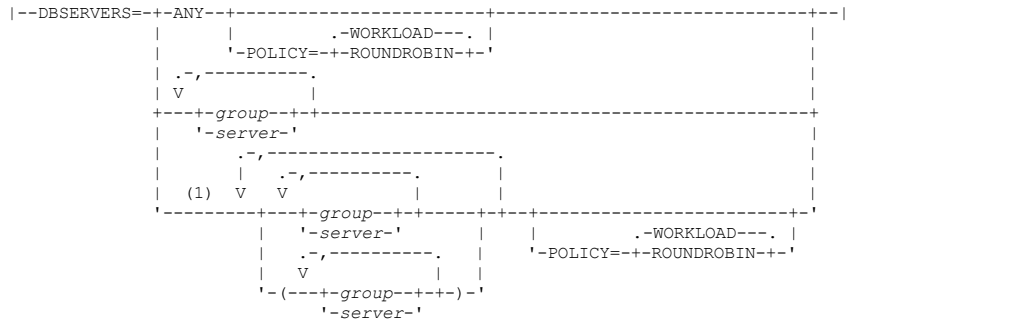
CLUSTER SLA syntax fragment

```
|--DBSERVERS=-+-ANY--+-------------------------------------------------------+-------------------------+--|
              |      '-POLICY=-+-ROUNDROBIN-------------------------------+-'                         |
              |                | .-WORKLOAD-.                             |                           |
              |                '-+----------+-+------------------------+-'                           |
              |                                  '-+SECAPPLYBACKLOG:pages-'                          |
              | .-,-----------.                                                                      |
              | V             | (1)                                                                  |
              +---+-group---+-+----------------------------------------------------------------------+
              |   +-HDR-----+                                                                        |
              |   +-PRI-----+                                                                        |
              |   +-primary-+                                                                        |
              |   +-RSS-----+                                                                        |
              |   +-SDS-----+                                                                        |
              |   '-server--'                                                                        |
              | .-,------------------------.                                                         |
              | |   .-,------------.       |                                                         |
              | V   V              |       |                                                         |
              '---+---+-group---+-+-----+-+--+-------------------------------------------------------+-'
                  |   +-HDR-----+       |    '-POLICY=-+-ROUNDROBIN-------------------------------+-'
                  |   +-PRI-----+       |              | .-WORKLOAD-.                             |
                  |   +-primary-+       |              '-+----------+-+------------------------+-'
                  |   +-RSS-----+       |                              '-+SECAPPLYBACKLOG:pages-'
                  |   +-SDS-----+       |
```

GRID or REPLSET SLA syntax fragment



SERVERSET SLA syntax fragment



Notes:

1. You must use at least one cluster keyword or a group of values that are enclosed by parentheses if you specify a redirection policy.

SLA parameter attributes

Table 1. The attributes of the SLA Connection Manager configuration parameter

Attribute name	Description
DBSERVERS	Specifies servers, server aliases, server groups, or server types for directing connection requests. Use the ANY keyword, the SDS or RSS cluster keywords, or enclose a group of values in parentheses to enable a redirection policy for that group.
HOST	Specifies a database server's host. The value in the SLA is used, rather than the value in the Connection Manager's host sqlhosts file.
MODE	Specifies whether connection requests go through the Connection Manager or if the Connection Manager provides connection information to the source of a connection request. The default value is REDIRECT.
NETTYPE	Specifies the network protocol of a database server. The value in the SLA is used, rather than the value in the Connection Manager's host sqlhosts file.
POLICY	Specifies how the Connection Manager redirects client connection requests to the servers specified in the DBSERVER attribute. Redirection policy applies to the ANY keyword, the SDS and RSS cluster keywords, and to a group of values that are enclosed in parentheses. Parentheses within parentheses are ignored. The default value is WORKLOAD. Workload, apply-failure, and transaction-latency policies can be given relative weights.
SERVICE	Specifies a database server's port number or service name. The value in the SLA is used, rather than the value in the Connection Manager's host sqlhosts file.

Attribute name	Description
SQLHOSTSOPT	Specifies connectivity options for a database server that is specified in a SLA. Enclose all connectivity options in a single pair of quotation marks. The value in the SLA is used, rather than the value in the Connection Manager's host sqlhosts file.
USEALIASES	Specifies whether the Connection Manager can redirect client connection requests to database server aliases specified by the DBSERVERALIASES configuration parameter. The default value is ON.
WORKERS	Specifies the number of worker threads that are allocated to the SLA. When a service-level agreement is specified, the Connection Manager creates an SLA listener process to intercept client connection requests. The SLA listener process can have one or more worker threads. The default value is 4.

DBSERVERS attribute values

Table 2. Values of the DBSERVERS attribute.

Attribute value	Value
ANY	Specifies that connection requests can be sent to any available database server in the specified cluster, grid, or replicate set. For a SERVERSET connection unit, ANY specifies that connection requests can be sent to any available database server specified by the SERVERSET connection-unit's INFORMIXSERVER parameter. You do not need to enclose ANY in parentheses to apply a redirection policy to it. If no redirection policy is specified, ANY uses the workload redirection policy.
group	Specifies a group entry in the Connection Manager's host sqlhosts file. Connection requests can be sent to the members of the group.
HDR	Is a cluster keyword that specifies that connection requests can be sent to the high-availability data replication server. HDR is supported only by CLUSTER connection units.
PRI or PRIMARY	Is a cluster keyword that specifies that connection requests can be sent to the primary database server. PRI and PRIMARY are supported only by CLUSTER connection units.
RSS	Is a cluster keyword that specifies that connection requests can be sent to remote standalone secondary servers. RSS is supported only by CLUSTER connection units. You do not need to enclose RSS in parentheses to apply a redirection policy to the servers it specifies. If no redirection policy is specified, RSS uses the workload redirection policy.
SDS	Is a cluster keyword that specifies that connection requests can be sent to shared-disk secondary servers. SDS is supported only by CLUSTER connection units. You do not need to enclose SDS in parentheses to apply a redirection policy to the servers it specifies. If no redirection policy is specified, SDS uses the workload redirection policy.
server	Specifies server or alias entry in the Connection Manager's host sqlhosts file. Connection requests can be sent to the server.

MODE attribute values

Table 3. Values of the MODE attribute.

Attribute value	Value
PROXY	Specifies that the Connection Manager acts as a proxy server for client connections. Use proxy mode for the following cases: • A firewall is preventing a client application from connecting to database servers. • You do not want to recompile applications are compiled with Data Server Driver for JDBC and SQLJ version 3.5.1 or before, or with 3.00 or before. Because a proxy-server Connection Manager handles all client/server communication, configure multiple Connection Manager instances, to avoid a Connection Manager becoming a single point of failure. Note: For proxy mode, you must set your operating system to allow the maximum number of file descriptors. For example, use the **ulimit** command on UNIX operating systems.
REDIRECT (default)	Specifies that client connections use redirect mode, which configures the Connection Manager to return the appropriate database server name, IP address, and port number to the requesting client application. The client application then uses the returned IP address and port number to connect to the specified database server. Redirection-policy SLAs do not support connections from application that are compiled with Data Server Driver for JDBC and SQLJ version 3.5.1 or before, or with 3.00 or before.

POLICY attribute values

Table 4. Values of the POLICY attribute.

Attribute value	Value

Attribute value	Value
FAILURE	Specifies that connection requests are directed or proxied to the replication server with the fewest apply failures. The apply-failure policy is supported by the following connection units: • REPLSET • GRID To use the apply-failure policy, you must enable quality of data (QOD) monitoring by running the **cdr define qod** and **cdr start qod** commands. To use the apply-failure policy for a grid, the grid must have a replication-enabled table.
LATENCY	Specifies that connection requests are directed or proxied to the replication server with the lowest transaction latency. The transaction-latency policy is supported by the following connection units: • REPLSET • GRID The attribute value does not indicate a specific transaction latency period; the Connection Manager uses a formula with relative values to decide where to redirect client connection requests. To use the transaction-latency policy, you must enable quality of data monitoring by running the **cdr define qod** and **cdr start qod** commands. To use the transaction-latency policy for a grid, the grid must have a replication-enabled table.
ROUNDROBIN	Specifies that connection requests are directed or proxied in a repeating, ordered fashion (round-robin) to a group of servers. If you use a round-robin policy, the DBSERVERS attribute values are used to create round-robin groups. Servers that are specified more than one time in a DBSERVERS-attribute group value are treated as single participants in a round-robin group. For example, if a SLA has the following definition: `SLA sla_1 DBSERVERS=(server_1,server_3,server_1,server_2) \` `POLICY=ROUNDROBIN` The round-robin group participants are: • **server_1** • **server_2** • **server_3** The round-robin policy is supported by the following connection units: • CLUSTER • REPLSET • GRID • SERVERSET The round-robin policy is supported by the following software: • All IBM® Informix® server versions. • Connection Managers from 3.70.xC8 and later, and 4.10xC2 and later.
SECAPPLYBACKLOG:*number_of_pages*	Specifies that if a secondary server's apply backlog exceeds *number_of_pages*, the Connection Manager does not redirect or proxy new connections to server. For example: `SLA sla_1 DBSERVERS=(server_1,server_2,server_3) \` `POLICY=SECAPPLYBACKLOG:500` The Connection Manager sends connection requests to whichever of **server_1**, **server_2**, or **server_3** has an apply backlog below 500 pages and the lowest workload. To view the apply backlogs of all servers in a cluster, run the **onstat -g** cluster command. To view the apply backlog for a specific secondary server, run one of the following commands: • **onstat -g dri** • **onstat -g sds** • **onstat -g rss** The apply-backlog policy is supported by CLUSTER connection units. The apply-backlog policy is supported by the following software: • IBM Informix server versions 11.70.xC8 and later, and 12.10.xC2 and later. • Connection Managers from 3.70.xC8 and later, and 4.10xC2 and later.

Attribute value	Value
WORKLOAD (default)	Specifies that connection requests are directed or proxied to the database server with the lowest workload. Workload calculations are based on the number of virtual processors a server has and the number of threads in the server's ready queue. The WORKLOAD policy is supported by the following connection units: • CLUSTER • GRID • REPLSET • SERVERSET

USEALIASES attribute values

Table 5. Values of the USEALIASES attribute.

Attribute value	Value
ON (default)	Specifies that the Connection Manager can direct client connection requests to server aliases specified by a database server's DBSERVERALIASES configuration parameter.
OFF	Specifies that the Connection Manager cannot direct client connection requests to server aliases specified by a database server's DBSERVERALIASES configuration parameter.

HOST attribute values

Table 6. Values of the HOST attribute.

Attribute value	Value
host_name	Specifies a host name or host alias for a database server.
ip_address	Specifies a TCP/IP address for a database server.

NETTYPE attribute values

Table 7. Values of the NETTYPE attribute.

Attribute value	Value
drsocssl	Specifies secured sockets layer (SSL) protocol for Distributed Relational Database Architecture™ (DRDA).
drsoctcp	Specifies TCP/IP protocol for Distributed Relational Database Architecture
onsocssl	Specifies secured sockets layer protocol.
onsoctcp	Specifies sockets with TCP/IP protocol.

SERVICE attribute values

Table 8. Values of the SERVICE attribute.

Attribute value	Value
port_number	Specifies a port number.
service_name	Specifies a service name.

Usage

The SLA parameter is optional, and is supported by the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

Client applications use the SLA name to connect to the database servers or database-server types that are specified by the value of the DBSERVERS attribute. For each SLA, a listener thread is installed at the specified port on the server to detect incoming client requests. The SLA parameter can be specified multiple times in the same configuration file; however, each SLA name must be unique.

Example 1: Connection request redirection from a service-level agreement

The following example shows a simple Connection Manager configuration. The configuration specifies a single SLA.

```
NAME my_connection_manager_1

CLUSTER my_cluster_1
{
   INFORMIXSERVER my_server_group_1
   SLA sla_1 DBSERVERS=SDS,HDR,PRI
```

```
   FOC ORDER=ENABLED PRIORITY=1
}
```

`CONNECT TO @sla_1` connection requests as are directed in the following way:

1. Connect to any available SD secondary servers.
2. If SD secondary servers are unavailable, connect to the HDR secondary server.
3. If the HDR secondary server is unavailable, connect to the primary server.

Example 2: Defining multiple service-level agreements

The following example shows a simple Connection Manager configuration. The configuration specifies two SLAs.

```
NAME my_connection_manager_2

CLUSTER my_cluster_2
{
   INFORMIXSERVER my_server_group_2
   SLA sla_1 DBSERVERS=server_1
   SLA sla_2 DBSERVERS=server_2,server_3
   FOC ORDER=ENABLED PRIORITY=1
}
```

This example configures the Connection Manager for a high-availability cluster and defines two SLAs:

- `CONNECT TO @sla_1` connection requests are directed to **server_1**.
- `CONNECT TO @sla_2` connection requests are directed to **server_2**. If **server_2** is not available, connection requests are directed to **server_3**.

Example 3: Redirection policies in service-level agreements

The following example shows a Connection Manager configuration that uses a workload-balancing redirection policy. The configuration specifies a single SLA.

```
NAME my_connection_manager_3

CLUSTER my_cluster_3
{
   INFORMIXSERVER my_server_group_3
   SLA sla_1 DBSERVERS=(server_1,server_2) \
             POLICY=WORKLOAD
   SLA sla_2 DBSERVERS=(server_3,server_4,server_5) \
             POLICY=ROUNDROBIN
   SLA sla_3 DBSERVERS=(server_6,server_7,server_8) \
             POLICY=ROUNDROBIN+SECAPPLYBACKLOG:400
   FOC ORDER=ENABLED PRIORITY=1
}
```

- `CONNECT TO @sla_1` connection requests are directed to whichever of **server_1** and **server_2** has the lowest workload. WORKLOAD is the default redirection policy, so specifying `POLICY=WORKLOAD` in the SLA is not required.
- `CONNECT TO @sla_2` connection requests are directed round-robin to **server_3**, **server_4** and **server_5**.
- `CONNECT TO @sla_3` connection requests are directed round-robin to **server_6**, **server_7** and **server_8**. If a server's apply backlog is 400 pages or greater, that server is ignored in the round-robin order and does not receive connection requests until its apply backlog falls below 400 pages.

Example 4: Proxy and redirect mode in service-level agreements

In redirect mode, the Connection Manager responds to client redirection requests by returning a specified database server's IP address and port number to the client application. The client application then uses the IP address and port number to connect to the database server. In proxy mode, the Connection Manager acts as a proxy server, and client requests are routed through the Connection Manager. Redirect mode is the default if no SLA mode is specified.

```
NAME my_connection_manager_4

CLUSTER my_cluster_4
{
   INFORMIXSERVER my_server_group_4
   SLA sla_1 DBSERVERS=ANY \
             MODE=REDIRECT #Default value, so is not required for the SLA definition
   SLA sla_2 DBSERVERS=ANY \
             MODE=PROXY
   FOC ORDER=ENABLED PRIORITY=1
}
```

- `CONNECT TO @sla_1` connection requests result in the Connection Manager returning the IP address and port number for a cluster server to the client application.
- `CONNECT TO @sla_2` connection requests are directed through the Connection Manager to a cluster server.

Example 5: Adjusting timeout values for the Connection Manager and cluster servers

The following example shows a Connection Manager configuration where default timeout values are changed:

```
NAME my_connection_manager_5
CM_TIMEOUT 300
EVENT_TIMEOUT 45
SECONDARY_EVENT_TIMEOUT 50

CLUSTER my_cluster_5
{
   INFORMIXSERVER my_server_group_5
   SLA sla_1 DBSERVERS=ANY
```

```
   FOC ORDER=ENABLED PRIORITY=1
}
```

- If a cluster does not receive any events from the Connection Manager within 300 seconds, the primary server of the cluster promotes the next available Connection Manager to the role of failover arbitrator.
- If the Connection Manager does not receive any events from the primary server within 45 seconds, the Connection Manager begins failover processing, and attempts to promote a secondary server to the primary server.
- If the Connection Manager does not receive any events from a secondary server within 50 seconds, the Connection Manager disconnects from the secondary server.

Example 6: Macros and workload balancing in service-level agreements

The following example shows a Connection Manager configuration that uses defined macros in SLAs.

```
NAME my_connection_manager_6
MACRO CA=ca_server_1,ca_server_2,ca_server_3
MACRO NY=ny_server_1,ny_server_2,ny_server_3

REPLSET my_replicate_set_1
{
   INFORMIXSERVER my_er_group_1,my_er_group_2
   SLA sla_1 DBSERVERS=${CA}
   SLA sla_2 DBSERVERS=(${NY}) \
             POLICY=ROUNDROBIN
}
```

In this example, two macros are defined:

- `CA`, which is composed of **ca_server_1**, **ca_server_2**, and **ca_server_3**.
- `NY`, which is composed of **ny_server_1**, **ny_server_2**, and **ny_server_3**.

The Connection Manager redirects client connection requests as follows:

- `CONNECT TO @sla_1` connection requests are directed to **ca_server_1**. If **ca_server_1** is unavailable, connection requests are directed to **ca_server_2**. If **ca_server_2** is also unavailable, connection requests are directed to **ca_server_3**.
- `CONNECT TO @sla_2` connection requests are directed round-robin to **ny_server_1**, **ny_server_2**, and **ny_server_3**.

Example 7: Quality of data redirection policies in service-level agreements

The following example shows a Connection Manager configuration that uses transaction-latency and apply-failure redirection policies to direct connection requests in a grid. Quality-of-data (QOD) monitoring is turned on with the **cdr define qod** and **cdr start qod** commands.

```
NAME my_connection_manager_7

GRID my_grid_1
{
   INFORMIXSERVER my_server_group_1,my_server_group_2,my_server_group_3
   SLA sla_1 DBSERVERS=ANY \
             POLICY=LATENCY
   SLA sla_2 DBSERVERS=ANY \
             POLICY=FAILURE
   SLA sla_3 DBSERVERS=ANY \
             POLICY=2*Failure+LATENCY
}
```

- `CONNECT TO @sla_1` connection requests are directed to the server with the lowest transaction latency.
- `CONNECT TO @sla_2` connection requests are directed to the server with the lowest number of apply failures.
- `CONNECT TO @sla_3` connection requests are directed to the server with the lowest number of apply failures and lowest transaction latency. The smallest apply-failure count is twice as important as low transaction latency in the Connection Manager's calculations.

Example 8: sqlhosts connectivity information in service-level agreements

The following example shows a Connection Manager configuration that uses attribute values instead of the values in its host sqlhosts file for directing connection requests.

```
NAME my_connection_manager_8

SERVERSET my_server_set
{
   INFORMIXSERVER server_1,server_2,server_3
   SLA sla_1 DBSERVERS=server_1 \
             NETTYPE=onsoctcp
             HOST=host_1 \
             SERVICE=port_1 \
   SLA sla_2 DBSERVERS=server_2 \
             NETTYPE=onsoctcp
             HOST=host_2 \
             SERVICE=port_2 \
}
```

The Connection Manager uses the values of the HOST, SERVICE, and NETTYPE attributes, rather than the values in its host sqlhosts file for directing connection requests.

Example 9: Controlling connection-requests with aliases

The following example shows a Connection Manager configuration that specifies which database-server aliases SLAs can use.

The onconfig file for **server_1** has the following parameter setting:

```
DBSERVERALIASES server_1_alias_1,server_1_alias_2
```

The sqlhosts file that **server_1** uses has the following entries:

```
#dbservername      nettype   hostname   servicename  options
my_group_9         group     -          -            e=server_1_alias_2
server_1           onsoctcp  my_host_1  my_port_1    g=my_group_9
server_1_alias_1   onsoctcp  my_host_1  my_port_2    g=my_group_9
server_1_alias_2   onsoctcp  my_host_1  my_port_3    g=my_group_9
```

The Connection Manager configuration file for **server_1** has the following entries:

```
NAME my_connection_manager_9

SERVERSET my_server_set_2
{
   INFORMIXSERVER my_group_9
   SLA sla_1 DBSERVERS=server_1
   SLA sla_2 DBSERVERS=server_1 \
             USEALIASES=OFF
   SLA sla_3 DBSERVERS=server_1_alias_1
   SLA sla_4 DBSERVERS=server_1_alias_1 \
             USEALIASES=OFF
}
```

The Connection Manager directs client requests in the following ways:

- `CONNECT TO @sla_1` and `CONNECT TO @sla_3` requests can be directed to **server_1**, through **my_port_1**, **my_port_2**, or **my_port_3**.
- `CONNECT TO @sla_2` requests are directed to **server_1** through **my_port_1** only.
- `CONNECT TO @sla_4` requests are directed to **server_1** through **my_port_2** only.

Related reference:
Group information

SQLHOSTS Connection Manager configuration parameter

The SQLHOSTS parameter specifies where a Connection Manager can search for database servers that are specified by the INFORMIXSERVER parameter and DBSERVERS attribute.

Syntax

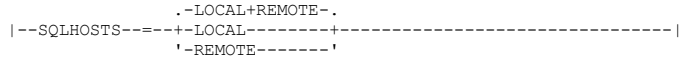


Table 1. Values for the SQLHOSTS Connection Manager configuration parameter

SQLHOSTS parameter value	Description
LOCAL	The Connection Manager searches the local sqlhosts file for requested database server instances.
REMOTE	The Connection Manager searches for requested database server instances in remote sqlhosts files. The Connection Manager searches the sqlhosts files of database servers that are specified by a connection-unit's INFORMIXSERVER parameter.
LOCAL+REMOTE (default)	The Connection Manager searches the local sqlhosts file for requested database server instances. If a database server cannot be found, the Connection Manager searches the sqlhosts files of database servers that are specified by a connection-unit's INFORMIXSERVER parameter.

Usage

The SQLHOSTS parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The SQLHOSTS option is useful when you want to restrict client applications from accessing one or more database servers in a high-availability cluster.

Example

In the following example, the SQLHOSTS parameter is used to limit the servers that the Connection Manager connects to.

```
NAME my_connection_manager
SQLHOSTS LOCAL

CLUSTER my_cluster
{
   INFORMIXSERVERS my_servers
   SLA my_sla DBSERVERS=server_1,server_2,server_3,server_4
```

```
   FOC ORDER=ENABLED PRIORITY=1
}
```

The local sqlhosts file has the following entries:

```
#dbservername       nettype    hostname    servicename    options
 my_servers         group      -           -              c=1,e=server_3
 server_1           onsoctcp   host_1      port_1         g=my_servers
 server_2           onsoctcp   host_2      port_2         g=my_servers
 server_3           onsoctcp   host_3      port_3         g=my_servers
```

Remote database-server sqlhosts files have the following entries:

```
#dbservername       nettype    hostname    servicename    options
 my_servers         group      -           -              c=1,e=server_4
 server_1           onsoctcp   host_1      port_1         g=my_servers
 server_2           onsoctcp   host_2      port_2         g=my_servers
 server_3           onsoctcp   host_3      port_3         g=my_servers
 server_4           onsoctcp   host_3      port_3         g=my_servers
```

server_4 is not defined in the local sqlhosts file, and the Connection Manager is configured to not search remote sqlhosts files, so the Connection Manager does not send connection requests to **server_4**.

Related reference:
Group information

SSL_LABEL Connection Manager configuration parameter

The SSL_LABEL parameter specifies the certificate label used for authentication by the CM when listening for an SSL connection.

```
SSL_LABEL certificate_name
```

The certificate name can be a single value, combination of alphanumeric characters, symbols or numbers.

Usage

The SSL_LABEL parameter is optional, and applies to the following connection units:

- CLUSTER
- GRID
- REPLSET
- SERVERSET

The SSL_LABEL option is useful when you want the CM to use the name of certificate specified in SSL_LABEL to authenticate incoming SSL connections.

Example

```
SSL_LABEL  cm1ListeningCertificate
```


Modifying Connection Manager configuration files

Use the **oncmsm** utility to load a modified configuration file into a Connection Manager and change the Connection Manager's configuration.

If you are using multiple Connection Managers, you can run the **onstat -g cmsm** command to display the names of Connection Manager instances.

1. Modify the existing Connection Manager configuration file. The default location of the configuration file is the $INFORMIXDIR/etc directory.
2. On the Connection Manager's host, run the **oncmsm** utility with the **r** parameter and the name of the Connection Manager instance. For example:

   ```
   oncmsm -r connection_manager_name
   ```

   Because multiple instances of the Connection Manager can be active at the same time, you must specify the name of the Connection Manager instance.

After you reload a Connection Manager's configuration file, the new configuration immediately takes effect.

Related information:
The oncmsm utility
onstat -g cmsm command: Print Connection Manager information

Converting older formats of the Connection Manager configuration file to the current format

The Connection Manager configuration file in versions of before version 3.70.xC3 are incompatible with the current version of the Connection Manager. You must convert configuration files from versions before 3.70.xC3 by running the **oncmsm** utility.

If you have multiple configuration files to convert, combine related SLA definitions into a single configuration file.

You can convert Connection Manager configuration files, even if a Connection Manager is running.

To convert a Connection Manager file to the current format:

1. Log on to the computer on which the updated Client SDK is installed.
2. Run the **oncmsm** utility, specifying old and new configuration file names.

After the configuration file is converted, it can be loaded into a Connection Manager.

Example: Converting a configuration file and loading it into a Connection Manager

For the following example:

- You are using a UNIX operating system.
- You updated a Connection Manager from 3.50.xC9 to 4.10xC1, and must update your configuration file.
- The Connection Manager was using a configuration file that is named configuration_file_old and located in $INFORMIXDIR/etc.
- You want to convert the file to the new format and rename the file to configuration_file_new.

Run the following commands:

1. `oncmsm -c configuration_file_old -n configuration_file_new`
2. `oncmsm -c configuration_file_new`

Related information:
The oncmsm utility

Configuring environments and setting configuration parameters for connection management

Before you start a Connection Manager, you must configure its environment.

To set environment variables:

- For UNIX C use the appropriate shell command. The following examples use C shell (csh).
- For Windows, use the Environment tab of the **setnet32** utility.

1. If clients or Connection Managers are installed on hosts where database servers are not installed, set each host's INFORMIXDIR environment variable to the directory the client or Connection Manager is installed in. Run the following command:

   ```
   setenv INFORMIXDIR path
   ```

2. If clients or Connection Managers are installed on hosts where database servers are not installed, set each host's INFORMIXSQLHOSTS environment variable to the location of the sqlhosts file that the host uses. Run the following command:

   ```
   setenv INFORMIXSQLHOSTS path_and_filename
   ```

3. If a Connection Manager's configuration file is in a directory other than $INFORMIXDIR/etc, set the CMCONFIG environment variable to the directory. Run the following command:

   ```
   setenv CMCONFIG path_and_filename
   ```

4. If Connection Managers control failover for a high-availability cluster, set the onconfig file DRAUTO configuration parameter to 3 on all managed cluster database servers. For example:

   ```
   DRAUTO 3
   ```

5. If Connection Managers control failover for a high-availability cluster, set the onconfig file HA_FOC_ORDER configuration parameter on the primary server of each cluster to a failover order. For example:

   ```
   HA_FOC_ORDER SDS,HDR,RSS
   ```

6. On each database server that uses multiple ports, set the onconfig file DBSERVERALIASES configuration parameter to the alias names listed in Connection Manager and database server sqlhosts file entries. For example:
   A Connection Manager's host has the following sqlhost file entries:

   ```
   #dbservername   nettype    hostname    servicename    options
    my_servers     group      -           -              c=1,e=server_3
    server_1       onsoctcp   host_1      port_1         s=6,g=my_servers
    server_2       onsoctcp   host_2      port_2         s=6,g=my_servers
    server_3       onsoctcp   host_3      port_3         s=6,g=my_servers

    my_aliases     group      -           -              c=1,e=a_server_3
    a_server_1     onsoctcp   host_1      port_4         g=my_aliases
    a_server_2     onsoctcp   host_2      port_5         g=my_aliases
    a_server_3     onsoctcp   host_3      port_6         g=my_aliases
   ```

   In the onconfig file for **server_1**, set the following value:

```
DBSERVERALIASES a_server_1
```

In the onconfig file for **server_2**, set the following value:

```
DBSERVERALIASES a_server_2
```

In the onconfig file for **server_3**, set the following value:

```
DBSERVERALIASES a_server_3
```

Related information:
INFORMIXDIR environment variable
INFORMIXSQLHOSTS environment variable
CMCONFIG environment variable
DBSERVERALIASES configuration parameter
DRAUTO configuration parameter
HA_FOC_ORDER configuration parameter

Defining sqlhosts information for connection management

You must define sqlhosts network-connectivity information for client applications that connect to Connection Managers, Connection Managers that connect to database servers, and database servers that are part of a Connection Manager connection unit.

If Connection Managers or clients are installed on hosts where database servers are not installed, you must create a sqlhosts file on each host.

Entries in an sqlhosts file can specify connection information for the following connection-unit components:

- Database servers
- Aliases for database servers that are using secure ports
- Connection Manager service-level agreements (SLAs)
- Groups that can contain database servers, database-server aliases, or SLAs.

All database servers that a Connection Manager connects to must be listed in the sqlhosts file that the Connection Manager uses. If the Connection Manager is monitoring a high-availability cluster, the sqlhosts file the Connection Manager uses must contain entries for all cluster servers.

1. Create entries in each database server's host sqlhosts file. You can modify one sqlhosts file, and then distribute it to the hosts of other database servers.
2. Create entries in each Connection Manager's host sqlhosts file. You can create one sqlhosts file, and then distribute it to the hosts of other Connection Managers.
3. Create entries in each client application's host sqlhosts file. You can create one sqlhosts file, and then distribute it to the hosts of other client applications.
4. If a host has multiple database servers that are installed on it, if the sqlhosts file is in a directory other than $INFORMIXDIR/etc, or if you are using a network-connectivity file other than $INFORMIXDIR/etc/sqlhosts, set the host's INFORMIXSQLHOSTS environment variable to the location of the sqlhosts file.

If sqlhosts file entries use the `s=6` option to define secure ports, use the information in the sqlhosts file to create a password file.

- Defining sqlhosts information for connection management of high-availability clusters
  You must define sqlhosts network-connectivity information for connection management of high-availability clusters.
- Defining sqlhosts information for connection management of high-availability clusters that use secure ports
  You must define sqlhosts network-connectivity information for connection management of high-availability clusters. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.
- Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA)
  Connection Managers support Distributed Relational Database Architecture™ (DRDA) connections for high-availability clusters. You must define sqlhosts network-connectivity information for connection management of high-availability clusters that use DRDA.
- Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA) and secure ports
  Connection Managers support Distributed Relational Database Architecture (DRDA) connections for high-availability clusters. You must define sqlhosts network-connectivity information for connection management of high-availability clusters that use DRDA. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.
- Defining sqlhosts information for connection management of grids and replicate sets
  You must define sqlhosts network-connectivity information for connection management of replicate sets or grids.
- Defining sqlhosts information for connection management of grids and replicate sets that use secure ports
  You must define sqlhosts network-connectivity information for connection management of replicate sets or grids. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.
- Defining sqlhosts information for connection management high-availability replication systems
  You must define sqlhosts network-connectivity information for connection management of high-availability replication systems.
- Defining sqlhosts information for connection management of high-availability replication systems that use secure ports
  You must define sqlhosts network-connectivity information for connection management of high-availability replication systems. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.
- Defining sqlhosts information for connection management of server sets
  You must define sqlhosts network-connectivity information for connection management of server sets.

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable

Defining sqlhosts information for connection management of high-availability clusters

You must define sqlhosts network-connectivity information for connection management of high-availability clusters.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each database server in the cluster. For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1
 server_2       onsoctcp   host_2     port_2
 server_3       onsoctcp   host_3     port_3
```

2. On the host of each Connection Manager, add a group entry that contains each database server in the cluster, and add group options to the database-server entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     -          -                        c=1,e=server_3
 server_1       onsoctcp   host_1     port_1        g=my_servers
 server_2       onsoctcp   host_2     port_2        g=my_servers
 server_3       onsoctcp   host_3     port_3        g=my_servers
```

3. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_1     DBSERVERS=PRI
   SLA sla_secondaries_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_2     DBSERVERS=PRI
   SLA sla_secondaries_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername       nettype    hostname   servicename   options
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2

 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4
```

4. On the host of each client application, create sqlhosts file entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members.

```
#dbservername       nettype    hostname   servicename  options
 g_primary          group      -          -            c=1,e=sla_primary_2
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1    g=g_primary
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2    g=g_primary

 g_secondaries      group      -          -            c=1,e=sla_secondaries_2
 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3    g=g_secondaries
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4    g=g_secondaries
```

Client connection requests to **@g_primary** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2** to provide the client application with connection information for the primary server.

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable

Defining sqlhosts information for connection management of high-availability clusters that use secure ports

You must define sqlhosts network-connectivity information for connection management of high-availability clusters. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each database server in the cluster, and specify the `s=6` secure-port option. For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1        s=6
 server_2       onsoctcp   host_2     port_2        s=6
 server_3       onsoctcp   host_3     port_3        s=6
```

2. On the host of each Connection Manager and database server, create sqlhosts file alias entries for each database server. For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1        s=6
 a_server_1     onsoctcp   host_1     port_4

 server_2       onsoctcp   host_2     port_2        s=6
 a_server_2     onsoctcp   host_2     port_5

 server_3       onsoctcp   host_3     port_3        s=6
 a_server_3     onsoctcp   host_3     port_6
```

3. On the host of each Connection Manager, add a group entry the individual entries. Add group options to the database server and database server alias entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     group      -          -             c=1,e=a_server_3
 server_1       onsoctcp   host_1     port_1        s=6,g=my_servers
 a_server_1     onsoctcp   host_1     port_4        g=my_servers
 server_2       onsoctcp   host_2     port_2        s=6,g=my_servers
 a_server_2     onsoctcp   host_2     port_5        g=my_servers
 server_3       onsoctcp   host_3     port_3        s=6,g=my_servers
 a_server_3     onsoctcp   host_3     port_6        g=my_servers
```

   A password file that is encrypted through the **onpassword** utility is required for connectivity through secure ports. The entries in the previously shown sqlhosts file are represented in the following password file.

```
my_servers  a_server_1  user_1  my_password_1
my_servers  a_server_2  user_2  my_password_2
my_servers  a_server_3  user_3  my_password_3

server_1    a_server_1  user_1  my_password_1
server_2    a_server_2  user_2  my_password_2
server_3    a_server_3  user_3  my_password_3

a_server_1  a_server_1  user_1  my_password_1
a_server_2  a_server_2  user_2  my_password_2
a_server_3  a_server_3  user_3  my_password_3
```

4. In each database server's onconfig file, set the DBSERVERALIASES parameter to that database server's alias.
   The onconfig file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

   The onconfig file entry for **server_2**:

```
DBSERVERALIASES a_server_2
```

   The onconfig file entry for **server_3**:

```
DBSERVERALIASES a_server_3
```

5. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
   The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_1     DBSERVERS=PRI
   SLA sla_secondaries_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED PRIORITY=1
}
```

   The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_2     DBSERVERS=PRI
```

```
   SLA sla_secondaries_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED PRIORITY=2
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername       nettype    hostname   servicename   options
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2

 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4
```

6. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members.

```
#dbservername       nettype    hostname   servicename   options
 g_primary          group      -          -             c=1,e=sla_primary_2
 sla_primary_1      onsoctcp   cm_host_1  cm_port_1     g=g_primary
 sla_primary_2      onsoctcp   cm_host_2  cm_port_2     g=g_primary

 g_secondaries      group      -          -             c=1,e=sla_secondaries_2
 sla_secondaries_2  onsoctcp   cm_host_1  cm_port_3     g=g_secondaries
 sla_secondaries_2  onsoctcp   cm_host_2  cm_port_4     g=g_secondaries
```

Client connection requests to **@g_primary** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2** to provide the client application with connection information for the primary server.

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable
DBSERVERALIASES configuration parameter

Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA)

Connection Managers support Distributed Relational Database Architecture™ (DRDA) connections for high-availability clusters. You must define sqlhosts network-connectivity information for connection management of high-availability clusters that use DRDA.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each database server in the cluster. For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1
 server_2       onsoctcp   host_3     port_2
 server_3       onsoctcp   host_5     port_3
```

2. In each database server's onconfig file, set the DBSERVERALIASES parameter to specify an alias for the server.
The onconfig file entry for **server_1**:

```
DBSERVERALIASES drda_1
```

The onconfig file entry for **server_2**:

```
DBSERVERALIASES drda_2
```

The onconfig file entry for **server_3**:

```
DBSERVERALIASES drda_3
```

3. On the host of each Connection Manager, add entries for the DRDA aliases. Use a DRDA protocol for the `nettype` value. For example:

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1
 server_2       onsoctcp   host_2     port_2
 server_3       onsoctcp   host_3     port_3

 drda_1         drsoctcp   host_1     port_4
 drda_2         drsoctcp   host_2     port_5
 drda_3         drsoctcp   host_3     port_6
```

4. On the host of each Connection Manager, add a group entry for the group of database server and add a group entry for the group of DRDA aliases. Add group options to the database server and DRDA alias entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername   nettype    hostname   servicename   options
 my_servers     group      -          -             c=1,e=server_3
 server_1       onsoctcp   host_1     port_1        g=my_servers
 server_2       onsoctcp   host_2     port_2        g=my_servers
```

```
server_3        onsoctcp    host_3      port_3          g=my_servers

drda_aliases    group       -           -               c=1,e=drda_3
drda_1          drsoctcp    host_1      port_4          g=drda_aliases
drda_2          drsoctcp    host_2      port_5          g=drda_aliases
drda_3          drsoctcp    host_3      port_6          g=drda_aliases
```

5. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file. The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_1          DBSERVERS=PRI
   SLA sla_primary_drda_1     DBSERVERS=PRI
   SLA sla_secondaries_1      DBSERVERS=SDS,HDR
   SLA sla_secondaries_drda_1 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

   The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

CLUSTER my_cluster
{
   INFORMIXSERVER my_servers
   SLA sla_primary_2          DBSERVERS=PRI
   SLA sla_primary_drda_2     DBSERVERS=PRI
   SLA sla_secondaries_2      DBSERVERS=SDS,HDR
   SLA sla_secondaries_drda_2 DBSERVERS=SDS,HDR
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

   Add the following entries to each client application's host sqlhosts file:

```
#dbservername              nettype     hostname    servicename   options
 sla_primary_1             onsoctcp    cm_host_1   cm_port_1
 sla_primary_2             onsoctcp    cm_host_2   cm_port_2

 sla_secondaries_2         onsoctcp    cm_host_1   cm_port_3
 sla_secondaries_2         onsoctcp    cm_host_2   cm_port_4

 sla_primary_1_drda        drsoctcp    cm_host_1   cm_port_5
 sla_primary_2_drda        drsoctcp    cm_host_2   cm_port_6

 sla_secondaries_2_drda    drsoctcp    cm_host_1   cm_port_7
 sla_secondaries_2_drda    drsoctcp    cm_host_2   cm_port_8
```

6. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members.

```
#dbservername              nettype     hostname    servicename   options
 g_primary                 group       -           -             c=1,e=sla_primary_2
 sla_primary_1             onsoctcp    cm_host_1   cm_port_1     g=g_primary
 sla_primary_2             onsoctcp    cm_host_2   cm_port_2     g=g_primary

 g_secondaries             group       -           -             c=1,e=sla_secondaries_2
 sla_secondaries_2         onsoctcp    cm_host_1   cm_port_3     g=g_secondaries
 sla_secondaries_2         onsoctcp    cm_host_2   cm_port_4     g=g_secondaries

 g_primary_drda            group       -           -             c=1,e=sla_primary_2_drda
 sla_primary_1_drda        drsoctcp    cm_host_1   cm_port_5     g=g_primary_drda
 sla_primary_2_drda        drsoctcp    cm_host_2   cm_port_6     g=g_primary_drda

 g_secondaries_drda        group       -           -             c=1,e=sla_secondaries_2_drda
 sla_secondaries_2_drda    drsoctcp    cm_host_1   cm_port_7     g=g_secondaries_drda
 sla_secondaries_2_drda    drsoctcp    cm_host_2   cm_port_8     g=g_secondaries_drda
```

Client connection requests to **@g_primary_drda** are sent by **drsoctcp** protocol to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1_drda** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2_drda** to provide the client application with connection information for the primary server.

Related concepts:
The sqlhosts information
Related tasks:
Configuring connectivity between Informix database servers and IBM Data Server clients
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable
DBSERVERALIASES configuration parameter

Defining sqlhosts information for high-availability clusters that use Distributed Relational Database Architecture (DRDA) and secure ports

Connection Managers support Distributed Relational Database Architecture™ (DRDA) connections for high-availability clusters. You must define sqlhosts network-connectivity information for connection management of high-availability clusters that use DRDA. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

The Connection Manager's sqlhosts file must contain entries for all database servers that it connects to.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each database server in the cluster, and specify the `s=6` secure-port option. For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 server_2      onsoctcp  host_3    port_2       s=6
 server_3      onsoctcp  host_5    port_3       s=6
```

2. On the host of each Connection Manager and database server, add DRDA alias entries. Use a DRDA protocol for the `nettype` value, and specify the `s=6` secure-port option. For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 server_2      onsoctcp  host_2    port_2       s=6
 server_3      onsoctcp  host_3    port_3       s=6

 drda_1        drsoctcp  host_1    port_4       s=6
 drda_2        drsoctcp  host_2    port_5       s=6
 drda_3        drsoctcp  host_3    port_6       s=6
```

3. On the host of each Connection Manager and database server, create sqlhosts file alias entries for each database server and each DRDA alias. For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 a_server_1    onsoctcp  host_1    port_7

 server_2      onsoctcp  host_2    port_2       s=6
 a_server_2    onsoctcp  host_2    port_8

 server_3      onsoctcp  host_3    port_3       s=6
 a_server_3    onsoctcp  host_3    port_9

 drda_1        drsoctcp  host_1    port_4       s=6
 a_drda_1      drsoctcp  host_1    port_10

 drda_2        drsoctcp  host_2    port_5       s=6
 a_drda_2      drsoctcp  host_2    port_11

 drda_3        drsoctcp  host_3    port_6       s=6
 a_drda_3      drsoctcp  host_3    port_12
```

4. On the host of each Connection Manager, add group entries for the groups of database servers and DRDA entries. Add group options to the individual entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname  servicename  options
 g_servers     group     -         -            c=1,e=a_server_3
 server_1      onsoctcp  host_1    port_1       s=6,g=g_servers
 a_server_1    onsoctcp  host_1    port_7       g=g_servers
 server_2      onsoctcp  host_2    port_2       s=6,g=g_servers
 a_server_2    onsoctcp  host_2    port_8       g=g_servers
 server_3      onsoctcp  host_3    port_3       s=6,g=g_servers
 a_server_3    onsoctcp  host_3    port_9       g=g_servers

 g_drda        group     -         -            c=1,e=a_drda_3
 drda_1        drsoctcp  host_1    port_4       s=6,g=g_drda
 a_drda_1      drsoctcp  host_1    port_10      g=g_drda
 drda_2        drsoctcp  host_2    port_5       s=6,g=g_drda
 a_drda_2      drsoctcp  host_2    port_11      g=g_drda
 drda_3        drsoctcp  host_3    port_6       s=6,g=g_drda
 a_drda_3      drsoctcp  host_3    port_12      g=g_drda
```

A password file that is encrypted through the **onpassword** utility is required for connectivity through secure ports. The entries in the previously shown sqlhosts file are represented in the following password file.

```
g_servers   a_server_1  user_1  password_1
g_servers   a_server_2  user_2  password_2
g_servers   a_server_3  user_3  password_3

server_1    a_server_1  user_1  password_1
server_2    a_server_2  user_2  password_2
server_3    a_server_3  user_3  password_3

a_server_1  a_server_1  user_1  password_1
a_server_2  a_server_2  user_2  password_2
a_server_3  a_server_3  user_3  password_3

g_drda      a_drda_1    user_1  password_1
g_drda      a_drda_2    user_2  password_2
```

```
g_drda      a_drda_3    user_3  password_3

drda_1      a_drda_1    user_1  password_1
drda_2      a_drda_2    user_2  password_2
drda_3      a_drda_3    user_3  password_3

a_drda_1    a_drda_1    user_1  password_1
a_drda_2    a_drda_2    user_2  password_2
a_drda_3    a_drda_3    user_3  password_3
```

5. In each database server's onconfig file, set the DBSERVERALIASES parameter to that database server's aliases.
   The onconfig file entry for **server_1**:

   ```
   DBSERVERALIASES a_server_1,drda_1,a_drda_1
   ```

   The onconfig file entry for **server_2**:

   ```
   DBSERVERALIASES a_server_2,drda_2,a_drda_2
   ```

   The onconfig file entry for **server_3**:

   ```
   DBSERVERALIASES a_server_3,drda_3,a_drda_3
   ```

6. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
   The first Connection Manager's configuration file has the following entries:

   ```
   NAME connection_manager_1

   CLUSTER my_cluster
   {
      INFORMIXSERVER g_servers,g_drda
      SLA sla_primary_1          DBSERVERS=PRI
      SLA sla_primary_drda_1     DBSERVERS=PRI
      SLA sla_secondaries_1      DBSERVERS=SDS,HDR
      SLA sla_secondaries_drda_1 DBSERVERS=SDS,HDR
      FOC ORDER=ENABLED \
          PRIORITY=1
   }
   ```

   The second Connection Manager's configuration file has the following entries:

   ```
   NAME connection_manager_2

   CLUSTER my_cluster
   {
      INFORMIXSERVER g_servers,g_drda
      SLA sla_primary_2          DBSERVERS=PRI
      SLA sla_primary_drda_2     DBSERVERS=PRI
      SLA sla_secondaries_2      DBSERVERS=SDS,HDR
      SLA sla_secondaries_drda_2 DBSERVERS=SDS,HDR
      FOC ORDER=ENABLED \
          PRIORITY=2
   }
   ```

   Add the following entries to each client application's host sqlhosts file:

   ```
   #dbservername            nettype   hostname   servicename  options
    sla_primary_1           onsoctcp  cm_host_1  cm_port_1
    sla_primary_2           onsoctcp  cm_host_2  cm_port_2

    sla_secondaries_2       onsoctcp  cm_host_1  cm_port_3
    sla_secondaries_2       onsoctcp  cm_host_2  cm_port_4

    sla_primary_1_drda      drsoctcp  cm_host_1  cm_port_5
    sla_primary_2_drda      drsoctcp  cm_host_2  cm_port_6

    sla_secondaries_2_drda  drsoctcp  cm_host_1  cm_port_7
    sla_secondaries_2_drda  drsoctcp  cm_host_2  cm_port_8
   ```

7. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members.

   ```
   #dbservername            nettype    hostname   servicename  options
    g_primary               group      -          -            c=1,e=sla_primary_2
    sla_primary_1           onsoctcp   cm_host_1  cm_port_1    g=g_primary
    sla_primary_2           onsoctcp   cm_host_2  cm_port_2    g=g_primary

    g_secondaries           group      -          -            c=1,e=sla_secondaries_2
    sla_secondaries_2       onsoctcp   cm_host_1  cm_port_3    g=g_secondaries
    sla_secondaries_2       onsoctcp   cm_host_2  cm_port_4    g=g_secondaries

    g_primary_drda          group      -          -            c=1,e=sla_primary_2_drda
    sla_primary_1_drda      drsoctcp   cm_host_1  cm_port_5    g=g_primary_drda
    sla_primary_2_drda      drsoctcp   cm_host_2  cm_port_6    g=g_primary_drda

    g_secondaries_drda      group      -          -            c=1,e=sla_secondaries_2_drda
    sla_secondaries_2_drda  drsoctcp   cm_host_1  cm_port_7    g=g_secondaries_drda
    sla_secondaries_2_drda  drsoctcp   cm_host_2  cm_port_8    g=g_secondaries_drda
   ```

Client connection requests to **@g_primary_drda** are sent by **drsoctcp** protocol to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_primary_1_drda** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_primary_2_drda** to provide the client application with connection information for the primary server.

Related concepts:
The sqlhosts information
Related tasks:
Configuring connectivity between Informix database servers and IBM Data Server clients
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable
DBSERVERALIASES configuration parameter

Defining sqlhosts information for connection management of grids and replicate sets

You must define sqlhosts network-connectivity information for connection management of replicate sets or grids.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and replication server, create sqlhosts file entries for each replication server. For example:

```
#dbservername  nettype   hostname   servicename  options
 server_1      onsoctcp  host_1     port_1
 server_2      onsoctcp  host_2     port_2
 server_3      onsoctcp  host_3     port_3
 server_4      onsoctcp  host_4     port_4
```

2. On the host of each Connection Manager and replication server, create a sqlhosts file group entry for each replication server. Add group options to each replication-server entry. Use the `i=unique_number` group-entry option to assign an identifier to the group for Enterprise Replication. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 g_server_1    group     -          -            i=1,e=server_1
 server_1      onsoctcp  host_1     port_1       g=g_server_1

 g_server_2    group     -          -            i=2,e=server_2
 server_2      onsoctcp  host_2     port_2       g=g_server_2

 g_server_3    group     -          -            i=3,e=server_3
 server_3      onsoctcp  host_3     port_3       g=g_server_3

 g_server_4    group     -          -            i=4,e=server_4
 server_4      onsoctcp  host_4     port_4       g=g_server_4
```

3. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   INFORMIXSERVER g_server_1,g_server_2,g_server_3,g_server_4
   SLA sla_1 DBSERVERS=ANY
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   INFORMIXSERVER g_server_1,g_server_2,g_server_3,g_server_4
   SLA sla_2 DBSERVERS=ANY
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

4. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         onsoctcp  -          -             c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1     g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2     g=g_sla
```

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for a replication server.
Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:

Defining sqlhosts information for connection management of grids and replicate sets that use secure ports

You must define sqlhosts network-connectivity information for connection management of replicate sets or grids. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and replication server, create sqlhosts file entries for each replication server, and specify the `s=6` secure-port option. For example:

```
#dbservername  nettype   hostname   servicename  options
 server_1      onsoctcp  host_1     port_1       s=6
 server_2      onsoctcp  host_2     port_3       s=6
 server_3      onsoctcp  host_3     port_5       s=6
 server_4      onsoctcp  host_4     port_7       s=6
```

2. On the host of each Connection Manager and replication server, create a sqlhosts file alias entry for each replication server. For example:

```
#dbservername  nettype   hostname   servicename  options
 server_1      onsoctcp  host_1     port_1       s=6
 a_server_1    onsoctcp  host_1     port_2

 server_2      onsoctcp  host_2     port_3       s=6
 a_server_2    onsoctcp  host_2     port_4

 server_3      onsoctcp  host_3     port_5       s=6
 a_server_3    onsoctcp  host_3     port_6

 server_4      onsoctcp  host_4     port_7       s=6
 a_server_4    onsoctcp  host_4     port_8
```

   The aliases are used by the **cdr utility**, which cannot connect to a secure port.
3. On the host of each Connection Manager and replication server, create a sqlhosts file group entry for each replication server and alias pair. Use the `i=unique_number` group-entry option to assign an identifier to the group for Enterprise Replication. Add group options to each replication server and alias entry. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 g_server_1    group     -          -            i=1,e=a_server_1
 server_1      onsoctcp  host_1     port_1       g=g_server_1,s=6
 a_server_1    onsoctcp  host_1     port_2       g=g_server_1

 g_server_2    group     -          -            i=2,e=a_server_2
 server_2      onsoctcp  host_2     port_3       g=g_server_2,s=6
 a_server_2    onsoctcp  host_2     port_4       g=g_server_2

 g_server_3    group     -          -            i=3,e=a_server_3
 server_3      onsoctcp  host_3     port_5       g=g_server_3,s=6
 a_server_3    onsoctcp  host_3     port_6       g=g_server_3

 g_server_4    group     -          -            i=4,e=a_server_4
 server_4      onsoctcp  host_4     port_7       g=g_server_4,s=6
 a_server_4    onsoctcp  host_4     port_8       g=g_server_4
```

   A password file that is encrypted through the **onpassword** utility is required for connectivity through secure ports. The entries in the previously shown sqlhosts file are represented in the following password file.

```
g_server_1  a_server_1  user_1  my_password_1
server_1    a_server_1  user_1  my_password_1
a_server_1  a_server_1  user_1  my_password_1

g_server_2  a_server_2  user_2  my_password_2
server_2    a_server_2  user_2  my_password_2
a_server_2  a_server_2  user_2  my_password_2

g_server_3  a_server_3  user_3  my_password_3
server_3    a_server_3  user_3  my_password_3
a_server_3  a_server_3  user_3  my_password_3

g_server_4  a_server_4  user_4  my_password_4
server_4    a_server_4  user_4  my_password_4
a_server_4  a_server_4  user_4  my_password_4
```

4. In each replication server's onconfig file, set the DBSERVERALIASES parameter to that database server's aliases.
   The onconfig file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

   The onconfig file entry for **server_2**:

```
DBSERVERALIASES a_server_2
```

   The onconfig file entry for **server_3**:

```
DBSERVERALIASES a_server_3
```

The onconfig file entry for **server_4**:

```
DBSERVERALIASES a_server_4
```

5. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file. The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   INFORMIXSERVER g_server_1,g_server_2,g_server_3,g_server_4
   SLA sla_1 DBSERVERS=ANY
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   INFORMIXSERVER g_server_1,g_server_2,g_server_3,g_server_4
   SLA sla_2 DBSERVERS=ANY
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

6. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         group     -          -            c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1    g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2    g=g_sla
```

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for the primary server.

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable
DBSERVERALIASES configuration parameter

Defining sqlhosts information for connection management high-availability replication systems

You must define sqlhosts network-connectivity information for connection management of high-availability replication systems.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

For this example, you are setting up Enterprise Replication between the primary servers of three high-availability clusters.

Cluster 1:

- server_1 (primary)
- server_2 (SD secondary)
- server_3 (HDR secondary)
- server_4 (RS secondary)

Cluster 2:

- server_5 (primary)
- server_6 (SD secondary)
- server_7 (HDR secondary)
- server_8 (RS secondary)

Cluster 3:

- server_9 (primary)
- server_10 (SD secondary)
- server_11 (HDR secondary)
- server_12 (RS secondary)

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each server. For example:

```
#dbservername  nettype   hostname   servicename  options
server_1       onsoctcp  host_1     port_1
server_2       onsoctcp  host_1     port_2
server_3       onsoctcp  host_2     port_3
server_4       onsoctcp  host_3     port_4

server_5       onsoctcp  host_4     port_5
server_6       onsoctcp  host_4     port_6
server_7       onsoctcp  host_5     port_7
server_8       onsoctcp  host_6     port_8

server_9       onsoctcp  host_7     port_9
server_10      onsoctcp  host_7     port_10
server_11      onsoctcp  host_8     port_11
server_12      onsoctcp  host_9     port_12
```

2. On the host of each Connection Manager and database server, create a sqlhosts file group entry for each replication-server entry and each cluster. Add group options to each database server entry. Use the `i=unique_number` group-entry option to assign an identifier to the group for Enterprise Replication. Use the `c=1` group-entry option for cluster groups, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 cluster_1     group     -          -            i=1,c=1,e=server_4
 server_1      onsoctcp  host_1     port_1       g=cluster_1
 server_2      onsoctcp  host_1     port_2       g=cluster_1
 server_3      onsoctcp  host_2     port_3       g=cluster_1
 server_4      onsoctcp  host_3     port_4       g=cluster_1

 cluster_2     group     -          -            i=2,c=1,e=server_8
 server_5      onsoctcp  host_4     port_5       g=cluster_2
 server_6      onsoctcp  host_4     port_6       g=cluster_2
 server_7      onsoctcp  host_5     port_7       g=cluster_2
 server_8      onsoctcp  host_6     port_8       g=cluster_2

 cluster_3     group     -          -            i=3,c=1,e=server_12
 server_9      onsoctcp  host_7     port_9       g=cluster_3
 server_10     onsoctcp  host_7     port_10      g=cluster_3
 server_11     onsoctcp  host_8     port_11      g=cluster_3
 server_12     onsoctcp  host_9     port_12      g=cluster_3
```

3. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   INFORMIXSERVER cluster_1,cluster_2,cluster_3
   SLA sla_1 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   INFORMIXSERVER cluster_1
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_2
{
   INFORMIXSERVER cluster_2
   FOC ORDER=ENABLED \
       PRIORITY=1
}

CLUSTER my_cluster_3
{
   INFORMIXSERVER cluster_3
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   INFORMIXSERVER cluster_1,cluster_2,cluster_3
   SLA sla_2 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   INFORMIXSERVER cluster_1
   FOC ORDER=ENABLED \
       PRIORITY=2
}

CLUSTER my_cluster_2
{
   INFORMIXSERVER cluster_2
   FOC ORDER=ENABLED \
```

```
       PRIORITY=2
}

CLUSTER my_cluster_3
{
   INFORMIXSERVER cluster_3
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

4. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         onsoctcp  -          -            c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1    g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2    g=g_sla
```

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for a replication server.

Related concepts:
The sqlhosts information

Related reference:
Group information

Related information:
High-availability replication systems
INFORMIXSQLHOSTS environment variable

Defining sqlhosts information for connection management of high-availability replication systems that use secure ports

You must define sqlhosts network-connectivity information for connection management of high-availability replication systems. If Connection Managers, database servers, or client applications are outside of a trusted network, you must also create an encrypted password file for security.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

For this example, you are setting up Enterprise Replication between the primary servers of two high-availability clusters.

Cluster 1:

- **server_1** (primary)
- **server_2** (SD secondary)
- **server_3** (HDR secondary)
- **server_4** (RS secondary)

Cluster 2:

- **server_5** (primary)
- **server_6** (SD secondary)
- **server_7** (HDR secondary)
- **server_8** (RS secondary)

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each database server. For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 server_2      onsoctcp  host_1    port_2       s=6
 server_3      onsoctcp  host_2    port_3       s=6
 server_4      onsoctcp  host_3    port_4       s=6

 server_5      onsoctcp  host_4    port_5       s=6
 server_6      onsoctcp  host_4    port_6       s=6
 server_7      onsoctcp  host_5    port_7       s=6
 server_8      onsoctcp  host_6    port_8       s=6
```

2. On the host of each Connection Manager and database server, create a sqlhosts file alias entry for each database server. For example:

```
#dbservername  nettype   hostname  servicename  options
 server_1      onsoctcp  host_1    port_1       s=6
 a_server_1    onsoctcp  host_1    port_9

 server_2      onsoctcp  host_1    port_2       s=6
 a_server_2    onsoctcp  host_1    port_10

 server_3      onsoctcp  host_2    port_3       s=6
 a_server_3    onsoctcp  host_2    port_11
```

```
server_4      onsoctcp  host_3    port_4       s=6
a_server_4    onsoctcp  host_3    port_12

server_5      onsoctcp  host_4    port_5       s=6
a_server_5    onsoctcp  host_4    port_13

server_6      onsoctcp  host_4    port_6       s=6
a_server_6    onsoctcp  host_4    port_14

server_7      onsoctcp  host_5    port_7       s=6
a_server_7    onsoctcp  host_5    port_15

server_8      onsoctcp  host_6    port_8       s=6
a_server_8    onsoctcp  host_6    port_16
```

The aliases are used by the **cdr utility**, which cannot connect to a secure port.

3. On the host of each Connection Manager and database server, create a sqlhosts file group entry for each cluster. Add group options to each database server entry. Use the `i=unique_number` group-entry option to assign an identifier to the group for Enterprise Replication. Use the `c=1` group-entry option for cluster groups, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname  servicename  options
 cluster_1     group     -          -           i=1,c=1,e=a_server_4
 server_1      onsoctcp  host_1    port_1       s=6,g=cluster_1
 a_server_1    onsoctcp  host_1    port_9       g=cluster_1
 server_2      onsoctcp  host_1    port_2       s=6,g=cluster_1
 a_server_2    onsoctcp  host_1    port_10      g=cluster_1
 server_3      onsoctcp  host_2    port_3       s=6,g=cluster_1
 a_server_3    onsoctcp  host_2    port_11      g=cluster_1
 server_4      onsoctcp  host_3    port_4       s=6,g=cluster_1
 a_server_4    onsoctcp  host_3    port_12      g=cluster_1

 cluster_2     group     -          -           i=1,c=1,e=a_server_8
 server_5      onsoctcp  host_4    port_5       s=6,g=cluster_2
 a_server_5    onsoctcp  host_4    port_13      g=cluster_2
 server_6      onsoctcp  host_4    port_6       s=6,g=cluster_2
 a_server_6    onsoctcp  host_4    port_14      g=cluster_2
 server_7      onsoctcp  host_5    port_7       s=6,g=cluster_2
 a_server_7    onsoctcp  host_5    port_15      g=cluster_2
 server_8      onsoctcp  host_6    port_8       s=6,g=cluster_2
 a_server_8    onsoctcp  host_6    port_16      g=cluster_2
```

A password file that is encrypted through the **onpassword** utility is required for connectivity through secure ports. The entries in the previously shown sqlhosts file are represented in the following password file.

```
cluster_1   a_server_1  user_1  password_1
cluster_1   a_server_2  user_2  password_2
cluster_1   a_server_3  user_3  password_3
cluster_1   a_server_4  user_4  password_4

cluster_2   a_server_5  user_5  password_5
cluster_2   a_server_6  user_6  password_6
cluster_2   a_server_7  user_7  password_7
cluster_2   a_server_8  user_8  password_8

server_1    a_server_1  user_1  password_1
server_2    a_server_2  user_2  password_2
server_3    a_server_3  user_3  password_3
server_4    a_server_4  user_4  password_4
server_5    a_server_5  user_5  password_5
server_6    a_server_6  user_6  password_6
server_7    a_server_7  user_7  password_7
server_8    a_server_8  user_8  password_8

a_server_1  a_server_1  user_1  password_1
a_server_2  a_server_2  user_2  password_2
a_server_3  a_server_3  user_3  password_3
a_server_4  a_server_4  user_4  password_4
a_server_5  a_server_5  user_5  password_5
a_server_6  a_server_6  user_6  password_6
a_server_7  a_server_7  user_7  password_7
a_server_8  a_server_8  user_8  password_8
```

4. In each database server's onconfig file, set the DBSERVERALIASES parameter to that database server's aliases. For example:
The onconfig file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

5. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1

REPLSET my_replset
{
   INFORMIXSERVER cluster_1,cluster_2
   SLA sla_1 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   INFORMIXSERVER cluster_1
   FOC ORDER=ENABLED \
```

```
        PRIORITY=1
}

CLUSTER my_cluster_2
{
   INFORMIXSERVER cluster_2
   FOC ORDER=ENABLED \
        PRIORITY=1
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2

REPLSET my_replset
{
   INFORMIXSERVER cluster_1,cluster_2
   SLA sla_2 DBSERVERS=ANY
}

CLUSTER my_cluster_1
{
   INFORMIXSERVER cluster_1
   FOC ORDER=ENABLED \
        PRIORITY=2
}

CLUSTER my_cluster_2
{
   INFORMIXSERVER cluster_2
   FOC ORDER=ENABLED \
        PRIORITY=2
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         onsoctcp  cm_host_1  cm_port_1
 sla_2         onsoctcp  cm_host_2  cm_port_2
```

6. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option, so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 g_sla         onsoctcp  -          -             c=1,e=sla_2
 sla_1         onsoctcp  cm_host_1  cm_port_1     g=g_sla
 sla_2         onsoctcp  cm_host_2  cm_port_2     g=g_sla
```

Client connection requests to **@g_sla** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_2** to provide the client application with connection information for a replication server.

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
High-availability replication systems
INFORMIXSQLHOSTS environment variable
DBSERVERALIASES configuration parameter

Defining sqlhosts information for connection management of server sets

You must define sqlhosts network-connectivity information for connection management of server sets.

The Connection Manager's sqlhosts file must contain entries for all database servers that it connects to.

To use a file other than $INFORMIXDIR/etc/sqlhosts on a specific host, set the host's INFORMIXSQLHOSTS environment variable to the alternative file.

1. On the host of each Connection Manager and database server, create sqlhosts file entries for each database server. For example:

```
#dbservername   nettype    hostname   servicename   options
 standalone_1   onsoctcp   host_1     port_1

 standalone_2   onsoctcp   host_2     port_2

 standalone_3   onsoctcp   host_3     port_3
```

2. On the host of each client application, create an sqlhosts file entry for each service-level agreement (SLA) in each Connection Manager configuration file.
The first Connection Manager's configuration file has the following entries:

```
NAME connection_manager_1
MACRO servers=standalone_1,standalone_2,standalone_3

SERVERSET ${servers}
{
```

```
   INFORMIXSERVER ${servers}
   SLA sla_1a DBSERVERS=standalone_1
   SLA sla_2a DBSERVERS=standalone_2
   SLA sla_3a DBSERVERS=standalone_3
}
```

The second Connection Manager's configuration file has the following entries:

```
NAME connection_manager_2
MACRO servers=standalone_1,standalone_2,standalone_3

SERVERSET ${servers}
{
   INFORMIXSERVER ${servers}
   SLA sla_1b DBSERVERS=standalone_1
   SLA sla_2b DBSERVERS=standalone_2
   SLA sla_3b DBSERVERS=standalone_3
}
```

Add the following entries to each client application's host sqlhosts file:

```
#dbservername  nettype   hostname   servicename  options
 sla_1a        onsoctcp  cm_host_1  cm_port_1
 sla_1b        onsoctcp  cm_host_2  cm_port_2

 sla_2a        onsoctcp  cm_host_1  cm_port_3
 sla_2b        onsoctcp  cm_host_2  cm_port_4

 sla_3a        onsoctcp  cm_host_1  cm_port_5
 sla_3b        onsoctcp  cm_host_2  cm_port_6
```

3. On the host of each client application, create sqlhosts file group entries for each group of SLA entries, and add group options to the SLA entries. Use the `c=1` group-entry option so that connection-attempt starting points in the list of group members is random. Use the `e=last_member` group-entry option so that the entire sqlhosts is not scanned for group members. For example:

```
#dbservername  nettype   hostname   servicename  options
 sla_1         group     -          -            c=1,e=sla_1b
 sla_1a        onsoctcp  cm_host_1  cm_port_1    g=sla_1
 sla_1b        onsoctcp  cm_host_2  cm_port_2    g=sla_1

 sla_2         group     -          -            c=1,e=sla_2b
 sla_2a        onsoctcp  cm_host_1  cm_port_3    g=sla_2
 sla_2b        onsoctcp  cm_host_2  cm_port_4    g=sla_2

 sla_3         group     -          -            c=1,e=sla_3b
 sla_3a        onsoctcp  cm_host_1  cm_port_5    g=sla_3
 sla_3b        onsoctcp  cm_host_2  cm_port_6    g=sla_3
```

Client connection requests to **@sla_1** are directed to one of the Connection Managers. If **connection_manager_1** receives the request, it uses **sla_1a** to provide the client application with connection information for the primary server. If **connection_manager_2** receives the request, it uses **sla_1b** to provide the client application with connection information for a replication server.

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
INFORMIXSQLHOSTS environment variable

Creating a password file for connecting to database servers on untrusted networks

If a client, Connection Manager, or any of the database servers that a Connection Manager connects to are on an untrusted network, you can create encrypted password files to verify connection requests.

In certain situations, an encrypted password file is required for trusted network environments, such as when a local system account attempts to connect to a database server in a high-availability cluster or Enterprise Replication domain, or when the user ID does not exist on a database server. The password file provides the correct system-level access, so that a local system account or a Windows account can connect directly to a remote server.

The password file has separate entries for the following items:

- Each Enterprise Replication group
- Each High-availability cluster group
- Each High-availability cluster server
- Each Enterprise Replication server that is in a group that is also configured for high-availability
- Each database server's alternative server alias, if the database server is using a secure port for communication

A password file entry contains the following information:

- The name of an alternative server to connect to if a connection cannot be made to the listed server or group. For example, *alternative_server_name* is used when *server_or_group_name* uses a secure port, as specified by the `s=6` option in an sqlhosts file entry.
- The user ID for a database server or the database servers in a group. User IDs must have the following privileges:
  - Permission to connect to the **sysadmin** database
  - CONNECT permission on the remote servers
  - On UNIX operating systems, membership in the group **informix** DBSA group
  - On Windows operating systems, membership in the **Informix-Admin** DBSA group

Only user **informix** has all of these privileges by default
- The password for a server

1. On a Connection Manager host, use a text editor to create an ASCII text file to be used as a password file. Save the file to the $INFORMIXDIR/tmp directory. If you have a high-availability replication system, your password file contains password information for replication servers and cluster servers.
   Note: The password file must not contain comments.
   The replication-server entries of the password file have the following format:

   ```
   group_name              database_server_alias  user_name  database_server_password
   database_server_name    database_server_alias  user_name  database_server_password
   database_server_alias   database_server_alias  user_name  database_server_password
   ```

   For example:

   ```
   group_1    unsecure_server_alias_1  user_1  password_1
   server_1   unsecure_server_alias_1  user_1  password_1
   alias_1    unsecure_server_alias_1  user_1  password_1

   group_2    unsecure_server_alias_2  user_2  password_2
   server_2   unsecure_server_alias_2  user_2  password_2
   alias_2    unsecure_server_alias_2  user_2  password_2

   group_n    unsecure_server_alias_n  user_n  password_n
   server_n   unsecure_server_alias_n  user_n  password_n
   alias_n    unsecure_server_alias_n  user_n  password_n
   ```

   The cluster-server entries of the password file have the following format:

   ```
   alias_group_name  db_server_alias  user_name  db_server_password

   db_server_name    db_server_alias  user_name  db_server_password
   ```

   For example:

   ```
   alias_group_1  unsecure_alias_1  user_1  password_1
   alias_group_1  unsecure_alias_2  user_2  password_2
   alias_group_1  unsecure_alias_n  user_n  password_n

   alias_group_2  unsecure_alias_1  user_1  password_1
   alias_group_2  unsecure_alias_2  user_2  password_2
   alias_group_2  unsecure_alias_n  user_n  password_n

   alias_group_n  unsecure_alias_1  user_1  password_1
   alias_group_n  unsecure_alias_2  user_2  password_2
   alias_group_n  unsecure_alias_n  user_n  password_n

   server_1       unsecure_alias_1  user_1  password_1
   server_2       unsecure_alias_2  user_2  password_2
   server_n       unsecure_alias_n  user_n  password_n
   ```

2. Encrypt the password file with the **onpassword** utility and an encryption key. For example, if your password file is $INFORMIXDIR/tmp/my_passwords.txt, and the encryption key you want to use is **my_secret_encryption_key_efgh**, run the following command:

   ```
   onpassword -k my_secret_encryption_key_efgh -e my_passwords.txt
   ```

   This example creates the encrypted passwd_file file in the $INFORMIXDIR/etc directory.

   To later decrypt the password file, you must enter the same key that was used to encrypt the password file. If you lose the encryption key that was used to encrypt a password file, re-encrypt the original ASCII text password file. If the ASCII text password file was deleted, you must create a new one.
3. Distribute $INFORMIXDIR/etc/passwd_file to all the database servers that Connection Managers or the **cdr** utility connects to, and to all Connection Managers.
   Note: An encrypted password file that is created on one type of operating system is not supported on a different type of operating system. On each operating system, you must run the **onpassword** utility with the same text file and encryption key.

- Modifying encrypted password information
  Modify the information in the encrypted passwd_file file by running the **onpassword** utility.

Related tasks:
Example: Configuring connection management for untrusted networks
Related information:
The onpassword utility
DBSERVERALIASES configuration parameter

Modifying encrypted password information

Modify the information in the encrypted passwd_file file by running the **onpassword** utility.

Modify the encrypted passwd_file file when the following events occur:

- Database servers are added to or removed from a high-availability cluster or replication domain
- sqlhosts file server aliases or groups change
- User IDs or server passwords change
- You want to change your encryption key

1. Decrypt the passwd_file file by running **onpassword** utility, specifying the previously used encryption key and a name for the output file. For example, if you previously encrypted the file, and used **my_secret_encryption_key_asdf** as the encryption key, run the following command:

```
onpassword -k my_secret_encryption_key_asdf -d my_passwords.txt
```

The **onpassword** utility creates the ASCII text my_passwords.txt output file in the $INFORMIXDIR/etc directory.

2. Optional: Open the file with a text editor, and modify the information in the file.
3. Encrypt the password file with the **onpassword** utility, specifying an encryption key and the name of the text file. For example:

```
onpassword -k my_secret_encryption_key_lmnop -e my_passwords.txt
```

This example uses the new encryption key, **my_secret_encryption_key_lmnop**, and creates the encrypted passwd_file file in the $INFORMIXDIR/etc directory.

4. Redistribute passwd_file to all the database servers that the Connection Manager or **cdr** utility connects to, replacing the previous $INFORMIXDIR/etc/passwd_file files. If you update the passwd_file on multiple operating systems, you must run the **onpassword** utility on each type of operating system, and use the same text file and encryption key.

Related information:
The onpassword utility

Starting Connection Managers on UNIX and Linux

Use the **oncmsm** utility to start a Connection Manager.

A Connection Manager can be started before the database servers it manages are started. If a connection unit is managed by a Connection Manager, the Connection Manager connects to database servers in the connection unit after the database servers start.

If the database servers of a connection unit are online before a Connection Manager is initialized, start the Connection Manager, and then direct client application requests to the Connection Manager instead of the database servers.

1. Start the Connection Manager by running the **oncmsm** utility with the **c** parameter and the name of the Connection Manager configuration file.

```
oncmsm -c configuration_file
```

The default location for the configuration file is the $INFORMIXDIR/etc directory.

2. If you enabled Connection Manager logging by setting the LOG and LOGFILE parameters in the Connection Manager's configuration file, check the log to verify that the Connection Manager successfully started. The following message displays is the Connection Manager started:

```
Connection Manager started successfully
```

After a Connection Manager initializes, it attempts to connect to the database servers or groups of database servers that are specified by the INFORMIXSERVER parameter in the Connection Manager's configuration file. The Connection Manager then searches for and connects to all the database servers that are in each specified server's cluster, grid, or replicate set.
Related information:
The oncmsm utility

Starting Connection Managers on Windows

Use the **oncmsm** utility to start a Connection Manager.

A Connection Manager can be started before the database servers it manages are started. If a connection unit is managed by a Connection Manager, the Connection Manager connects to database servers in the connection unit after the database servers start.

If the database servers of a connection unit are online before a Connection Manager is initialized, start the Connection Manager, and then direct client application requests to the Connection Manager instead of the database servers.

1. Install the Connection Manager as a service by running the **oncmsm** utility with the **i** parameter, the **c** parameter, and the name of the Connection Manager configuration file.

```
oncmsm -i -c configuration_file
```

The default location for the configuration file is the %INFORMIXDIR%\etc directory.

2. Initialize the Connection Manager by running the **oncmsm** utility with the name of the Connection Manager that is specified in the Connection Manager configuration file.

```
oncmsm connection_manager_name
```

The following message is displayed:

```
Specify the user and password to run this service.
Press <ENTER> to run Connection Manager as 'localsystem'
```

3. Enter the **informix** user ID and password, or press Enter to automatically create an **informix-admin** group and assign it access rights.
4. If you enabled Connection Manager logging by setting the LOG and LOGFILE parameters in the Connection Manager's configuration file, check the log to verify that the Connection Manager successfully started. The following message displays if the Connection Manager started:

```
Connection Manager started successfully
```

After a Connection Manager initializes, it attempts to connect to the database servers or groups of database servers that are specified by the INFORMIXSERVER parameter in the Connection Manager's configuration file. The Connection Manager then searches for and connects to all the database servers that are in each specified server's cluster, grid, or replicate set.
Related information:

The oncmsm utility

Stopping connection management

When you no longer want a Connection Manager to manage connection units, run the **oncmsm** utility to stop the Connection Manager instance.

If you are using multiple Connection Managers, you can run **onstat -g cmsm** to display the names of Connection Manager instances.

1. Log on to the computer on which the Connection Manager instance is running.
2. Run the **oncmsm** utility with the **-k** parameter. For example:

   ```
   oncmsm -k connection_manager_name
   ```

3. If quality of data (QOD) monitoring is turned on, and you want to stop it, run the **cdr stop qod** command on the master server. For example:

   ```
   cdr stop qod -c server_1
   ```

The Connection Manager stops.

To unistall a stopped Connection Manager from a Windows operating system, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -u connection_manager_name
```

Related information:
The oncmsm utility

Monitoring and troubleshooting connection management

Tools are available to monitor connection management, and help you diagnose potential problems.

The following options are available for connection management monitoring and troubleshooting:

- Set the LOG and LOGFILE parameters in Connection Manager configuration files. Log files contain information about service level agreements, failover configuration, and status information. The location of the log file is displayed when the Connection Manager is started.
- Run **onstat -g cmsm** to display information about Connection Manager instances.
- Set the CMALARMPROGRAM parameter in Connection Manager configuration files, and configure the cmalarmprogram script to handle event alarms.
- If the Connection Manager raises an event alarm:
  - The INFORMIXCMNAME environment variable stores the name of the Connection Manager instance that raised the alarm.
  - TheINFORMIXCMCONUNITNAME environment variable stores the name of the Connection Manager connection unit that raised the alarm.

Related information:
The oncmsm utility
onstat -g cmsm command: Print Connection Manager information
Managing client connections

Strategies for increasing availability with Connection Managers

You can increase the resiliency of your client/server communication environment.

Install multiple network-interface cards (NICs) on client, Connection Manager, and database-server hosts

You can prevent a network-connectivity failure by installing multiple NICs on each host in a connection-management domain. If a NIC fails, the host can use other available NICs.

Install multiple Connection Managers

You can prevent a Connection Manager from becoming a single point of failure in your system by installing multiple Connection Managers to manage a domain.

Install Connection Managers separate from database servers

To prevent simultaneous Connection Manager and database server failure if a host fails, install Connection Managers on hosts that are not running database servers.

Use keywords in Connection Manager service-level-agreements

If a Connection Manager manages a high-availability cluster, you can use the cluster keywords to maintain connection consistency after failover. If database servers in a cluster switch roles after failover, the Connection Manager can use cluster keywords to continue directing client connection requests to the appropriate cluster-server type.

The cluster keywords are:

- HDR - High-availability data replication server
- RSS - Remote standalone secondary server
- SDS - Shared-disk secondary server
- PRI - Primary server
- PRIMARY - Primary server

Use group entries in sqlhosts files and Connection Manager configuration files

Create database-server groups in the host sqlhosts file of each Connection Manager that manages a high-availability cluster. Then, specify sqlhosts groups, rather than individual server names, in Connection Manager configuration files. If you specify database server names for a connection unit's INFORMIXSERVER parameter, and those specified servers are offline when a Connection Manager restarts, the Connection Manager is unable to reconnect to the cluster. If you specify a sqlhosts group for a connection unit's INFORMIXSERVER parameter, and at least one of the group's database servers is online when a Connection Manager restarts, the Connection Manager can reconnect to the cluster.

Specify the `c=1` option for sqlhosts group entries so that the connection-attempt starting point for a list of group members is random. For example:

```
#dbservername   nettype    hostname    servicename    options
 my_servers     -          -                          c=1
 server_1       onsoctcp   host_1      port_1         g=my_servers
 server_2       onsoctcp   host_2      port_2         g=my_servers
 server_3       onsoctcp   host_3      port_3         g=my_servers
```

Related concepts:
The sqlhosts information
Related reference:
Group information
Related information:
HA_FOC_ORDER configuration parameter

Configuration examples for connection management

The following examples show steps for setting up connection management for various connection units and various systems.

You can use these examples as a basis for developing your own connection-management system.

- Example of configuring connection management for a high-availability cluster
  This example shows steps that are required to configure connection management for a high-availability cluster.
- Example of configuring connection management for a grid or replicate set
  You can use a Connection Manager to route client connections for the replication servers of a grid or replicate set.
- Example of configuring connection management for a high-availability replication system
  You can use a Connection Manager to route client connections for the participants of a replicate set and to control failover for high-availability clusters that participate in Enterprise Replication.
- Example: Configuring connection management for untrusted networks
  This example shows steps that are required to configure connection management for an untrusted network.
- Example of configuring connection management for prioritizing connections and network monitoring
  You can install Connection Managers on the hosts of application servers, and then prioritize the connections between specific application servers and the primary server of a high-availability cluster. This configuration allows the highest priority application server to maintain its connection to the cluster's primary server if a portion of the network fails.
- Example of configuring for an SSL connection
  This example shows the steps to configure CM to listen for SSL connections from database clients.

Example of configuring connection management for a high-availability cluster

This example shows steps that are required to configure connection management for a high-availability cluster.

For this example, you have a high-availability cluster on a trusted network. The cluster consists of three servers:

- A primary server (**server_1**)
- A shared-disk secondary server (**server_2**)
- An HDR secondary server (**server_3**)

The cluster supports the following application services:

- Online transaction processing (OLTP), which runs only on the primary server
- Payroll services, which can run on the primary server or HDR secondary server
- Reporting services, which can run on any of the secondary servers

Your system has the following needs:

- The database servers' workloads are balanced.
- The Connection Managers control failover.
- If failover occurs, the SD secondary server takes priority over the HDR secondary server.
- If the primary server fails, the Connection Managers can still connect to the cluster after restarting.
- The system can withstand the failure of a Connection Manager.
- The system can withstand a network-interface card (NIC) failure on each host.

To configure connection management:

1. Install at least two network interface cards on each host. This prevents the failure of a network interface card from causing Connection Manager or database server failure.
2. Install two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on. This installation strategy prevents a Connection Manager from becoming a single point of failure, and prevents the simultaneous failure of database servers and Connection Managers if a host fails.
   You can install Connection Managers on application-server hosts if you want to prioritize an application server's connectivity to the primary cluster server.
3. On each host Connection Manager host, set the INFORMIXDIR environment to the directory the Connection Manager was installed into. Run the following command:

   ```
   setenv INFORMIXDIR path
   ```

4. Create a configuration file in each Connection Manager installation's $INFORMIXDIR/etc directory.
   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $INFORMIXDIR/tmp/my_cm1_log.log
   LOCAL_IP 192.0.2.0,192.0.2.1

   CLUSTER cluster_1
   {
      INFORMIXSERVER servers_1
      SLA oltp_1    DBSERVERS=primary
      SLA payroll_1 DBSERVERS=(PRI,HDR) \
                    POLICY=WORKLOAD
      SLA report_1  DBSERVERS=(SDS,HDR) \
                    POLICY=WORKLOAD
      FOC ORDER=ENABLED \
          PRIORITY=1
      CMALARMPROGRAM $INFORMIXDIR/etc/CMALARMPROGRAM.sh
   }
   ```

   The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

   ```
   NAME connection_manger_2
   LOG 1
   LOGFILE $INFORMIXDIR/tmp/my_cm2_log.log
   LOCAL_IP 192.0.2.2,192.0.2.3

   CLUSTER cluster_1
   {
      INFORMIXSERVER cluster_1
      SLA oltp_2    DBSERVERS=primary
      SLA payroll_2 DBSERVERS=(PRI,HDR)\
                    POLICY=WORKLOAD
      SLA report_2  DBSERVERS=(SDS,HDR) \
                    POLICY=WORKLOAD
      FOC ORDER=ENABLED \
          PRIORITY=2
      CMALARMPROGRAM $INFORMIXDIR/etc/CMALARMPROGRAM.sh
   }
   ```

   The configuration file specifies the following information and behavior:
   - Logging is enabled, and the log files are $INFORMIXDIR/tmp/my_cm1_log.log and $INFORMIXDIR/tmp/my_cm2_log.log.
   - **connection_manager_1** monitors 192.0.2.0 and 192.0.2.1 and **connection_manager_2** monitors 192.0.2.2 and 192.0.2.3 for network failure.
   - When the Connection Managers start, they each search their sqlhosts files for **cluster_1** entry, and then connect to the servers in that group.
   - `CONNECT TO @oltp_1` and `CONNECT TO @oltp_2` connection requests are directed to the primary server.
   - `CONNECT TO @payroll_1` and `CONNECT TO @payroll_2` connection requests are directed to whichever of the primary and HDR secondary servers has the lowest workload.
   - `CONNECT TO @report_1` and `CONNECT TO @report_2` connection requests are directed to the secondary server that has the lowest workload.
   - The connection between **connection_manager_1** and the primary server is prioritized over the connection between **connection_manager_2** and the primary server. Failover that would break the connectivity between **connection_manager_1** and the primary server is blocked.
   - If failover processing fails after eight attempts, $INFORMIXDIR/etc/CMALARMPROGRAM.sh is called.

   Certain parameters and attributes are not included in this configuration file, so the Connection Manager has the following default behavior:
   - The EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for primary-server events before failover processing begins.
   - The MODE attributes of the SLA parameters are not set, so the Connection Managers return connection information for **server_1**, **server_2**, and **server_3** to client applications, rather than acting as proxy servers.
   - The SECONDARY_EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for secondary-server events before the Connection Manager disconnects from the secondary server.
   - The HOST, NETTYPE, SERVICE, and SQLHOSTSOPT attributes of the SLA parameters are not set, so each Connection Manager uses connection information in local and remote sqlhosts files.
   - The SQLHOSTS parameter is not set, so each Connection Manager first searches its local sqlhosts file, and then remote database server sqlhosts files for connectivity information related to **server_1**, **server_2**, and **server_3**.
   - The WORKERS attributes of the SLA parameters are not set, so four worker threads are allocated to each of the SLAs.
5. Set the onconfig file DRAUTO configuration parameter on all database servers to `3`

```
DRAUTO 3
```

This setting specifies that a Connection Manager controls failover arbitration.

6. Set the onconfig file HA_FOC_ORDER configuration parameter on **server_1** to `SDS,HDR`

```
HA_FOC_ORDER SDS,HDR
```

After the Connection Managers start, and connect to **server_1**, the HA_FOC_ORDER value replaces the value of the `ORDER` attributes in each Connection Manager's configuration file.

If **server_1** fails, the Connection Managers attempt failover to the SD secondary server. If the SD secondary server is also unavailable, the Connection Managers attempt failover to the HDR secondary server.

7. Optional: Configure the cmalarmprogram script on each Connection Manager host. Event alarms can be sent to specified email addresses.
8. Add entries to the sqlhosts files on **server_1** and **server_2**'s host and on **server_3**'s host.

```
#dbservername  nettype    hostname   servicename   options
 server_1      onsoctcp   host_1     port_1
 server_2      onsoctcp   host_1     port_2
 server_3      onsoctcp   host_2     port_3
```

9. Create a sqlhosts file on each Connection Manager.

```
#dbservername  nettype    hostname   servicename   options
 cluster_1     group      -          -             c=1,e=server_3
 server_1      onsoctcp   host_1     port_1        g=cluster_1
 server_2      onsoctcp   host_1     port_2        g=cluster_1
 server_3      onsoctcp   host_2     port_3        g=cluster_1
```

If a Connection Manager restarts after a primary-server failure, it is able to connect to other database servers in the cluster because the **cluster_1** group is defined.

10. Create a sqlhosts file on each client host.

```
#dbservername  nettype    hostname   servicename   options
 oltp          group      -          -             c=1,e=oltp_2
 oltp_1        onsoctcp   cm_host_1  cm_port_1     g=oltp
 oltp_2        onsoctcp   cm_host_2  cm_port_2     g=oltp

 report        group      -          -             c=1,e=report_2
 report_1      onsoctcp   cm_host_1  cm_port_3     g=report
 report_2      onsoctcp   cm_host_2  cm_port_4     g=report

 payroll       group      -          -             c=1,e=payroll_2
 payroll_1     onsoctcp   cm_host_1  cm_port_5     g=payroll
 payroll_2     onsoctcp   cm_host_2  cm_port_6     g=payroll
```

If a Connection Manager fails, client applications can still connect to the other Connection Manager because the **oltp**, **report**, and **payroll** groups are defined.

- `CONNECT TO @oltp` connection requests are directed through one of the Connection Managers to the primary server.
- `CONNECT TO @payroll` connection requests are directed through one of the Connection Managers to whichever of the primary and HDR secondary servers has the lowest workload.
- `CONNECT TO @report` connection requests are directed through one of the Connection Managers to the secondary server that has the lowest workload.

11. Set each INFORMIXSQLHOSTS environment variable to the sqlhosts file location by running the **setenv** command on each Connection Manager and client host.

```
setenv INFORMIXSQLHOSTS path_and_file_name
```

12. Run the **oncmsm** utility on each Connection Manager host, to start each Connection Manager.
On the host of **connection_manager_1**:

```
oncmsm -c cm_1.cfg
```

On the host of **connection_manager_2**:

```
oncmsm -c cm_2.cfg
```

13. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Related reference:
Group information
Related information:
The oncmsm utility
HA_FOC_ORDER configuration parameter
INFORMIXSQLHOSTS environment variable

Example of configuring connection management for a grid or replicate set

You can use a Connection Manager to route client connections for the replication servers of a grid or replicate set.

For this example, you have a replicate set that consists of four replication servers:

- **server_1**
- **server_2**
- **server_3**
- **server_4**

The replication set supports reporting services, which can run on any of the replication servers.

Your system has the following needs

- Client requests are directed to the replication server with the fewest apply failures.
- The system can withstand the failure of a Connection Manager.
- The system can withstand a network-interface card (NIC) failure on each host.

To configure connection management:

1. Install at least two network interface cards on each host. This prevents the failure of a network interface card from causing Connection Manager or database server failure.
2. Install two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on. This installation strategy prevents a Connection Manager from becoming a single point of failure, and prevents the simultaneous failure of database servers and Connection Managers if a host fails.
3. On each Connection Manager host, set the INFORMIXDIR environment to the directory the Connection Manager was installed into. Run the following command:

   ```
   setenv INFORMIXDIR path
   ```

4. Create a configuration file in each Connection Manager installation's $INFORMIXDIR/etc directory.
   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $INFORMIXDIR/tmp/my_cm1_log.log

   REPLSET replicate_set_1
   {
           INFORMIXSERVER  g_server_1,g_server_2,g_server_3,g_server_4
           SLA report_1    DBSERVERS=ANY \
                POLICY=FAILURE
   }
   ```

   The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

   ```
   NAME connection_manger_2
   LOG 1
   LOGFILE $INFORMIXDIR/tmp/my_cm2_log_.log

   REPLSET replicate_set_1
   {
           INFORMIXSERVER  g_server_1,g_server_2,g_server_3,g_server_4
           SLA report_2    DBSERVERS=ANY \
                POLICY=FAILURE
   }
   ```

   The configuration file specifies the following information and behavior:
   - Logging is enabled, and the log files are $INFORMIXDIR/tmp/my_cm1_log.log and $INFORMIXDIR/tmp/my_cm2_log.log.
   - When the Connection Managers start, they each search their sqlhosts files for **g_server_1**, **g_server_2**, **g_server_3**, and **g_server_4** entries, and then connect to the servers **server_1**, **server_2**, **server_3**, and **server_4** that are in those groups.
   - `CONNECT TO @report_1` and `CONNECT TO @report_2` connection requests are directed to the replication server that has the fewest apply failures.

   Certain parameters and attributes are not included in this configuration file, so the Connection Manager has the following default behavior:
   - The MODE attributes of the SLA parameters are not set, so the Connection Managers return connection information for **server_1**, **server_2**, **server_3**, and **server_4** to client applications, rather than acting as proxy servers.
   - The HOST, NETTYPE, SERVICE, and SQLHOSTSOPT attributes of the SLA parameters are not set, so each Connection Manager uses connection information in local and remote sqlhosts files.
   - The SQLHOSTS parameter is not set, so each Connection Manager first searches its local sqlhosts file, and then remote database server sqlhosts files for connectivity information related to **server_1**, **server_2**, **server_3**, and **server_4**.
   - The WORKERS attributes of the SLA parameters are not set, so four worker threads are allocated to each of the SLAs.
5. Add entries to thesqlhosts files on the hosts of each database server, **connection_manger_1**, and **connection_manger_2**.

   ```
   #dbservername  nettype   hostname   servicename  options
    g_server_1    group     -          -            i=1,e=server_1
    server_1      onsoctcp  host_1     port_1       g=g_server_1

    g_server_2    group     -          -            i=2,e=server_2
    server_2      onsoctcp  host_2     port_2       g=g_server_2

    g_server_3    group     -          -            i=3,e=server_3
    server_3      onsoctcp  host_3     port_3       g=g_server_3

    g_server_4    group     -          -            i=4,e=server_4
    server_4      onsoctcp  host_4     port_4       g=g_server_4
   ```

6. Create a sqlhosts file on each client host.

   ```
   #dbservername  nettype    hostname   servicename   options
    report        group      -          -             c=1,e=report_2
    report_1      onsoctcp   cm_host_1  cm_port_3     g=report
    report_2      onsoctcp   cm_host_2  cm_port_4     g=report
   ```

   If a Connection Manager fails, client applications can still connect to the other Connection Manager because the **report** group is defined.

   `CONNECT TO @report` connection requests are directed through one of the Connection Managers to the replication server that has the fewest apply failures.

7. Set each INFORMIXSQLHOSTS environment variable to the sqlhosts file location by running the **setenv** command on each Connection Manager and client host.

   ```
   setenv INFORMIXSQLHOSTS path_and_file_name
   ```

8. Turn on quality of data (QOD) monitoring by running the **cdr define qod** command.

```
cdr define qod -c server_1 --start
```

The command connects to **server_1**, defines **server_1** as a master server for monitoring data, and then turns on quality of data monitoring.
server_1 maintains a failed-transaction count for the servers in the replicate set. The failed-transaction count determines which replication server the Connection Managers send a client connection requests to.

9. Run the **oncmsm** utility on each Connection Manager host, to start each Connection Manager.
On the host of **connection_manager_1**:

```
oncmsm -c cm_1.cfg
```

On the host of **connection_manager_2**:

```
oncmsm -c cm_2.cfg
```

10. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Related reference:
Group information
Related information:
The oncmsm utility
HA_FOC_ORDER configuration parameter
INFORMIXSQLHOSTS environment variable
cdr define qod
cdr start qod

Example of configuring connection management for a high-availability replication system

You can use a Connection Manager to route client connections for the participants of a replicate set and to control failover for high-availability clusters that participate in Enterprise Replication.

For this example, you have a grid that consists of four nodes. One of the nodes is a primary server in a high-availability cluster that consists of a primary server, an SD secondary server, an HDR secondary server, and an RS secondary server:

- **server_1a** - ER Node 1, primary server
- **server_1b** - SD secondary server
- **server_1c** - HDR secondary server
- **server_1d** - RS secondary server
- **server_2** - ER Node 2
- **server_3** - ER Node 3
- **server_4** - ER Node 4

The grid supports reporting services, which can run on any of the ER nodes.

Your system has the following needs

- Client requests are directed to the ER node with the lowest transaction latency.
- The system can withstand the failure of a Connection Manager.
- The system can withstand a network-interface card (NIC) failure on each host.
- The Connection Managers control failover for the cluster.
- If failover occurs, it the SD secondary server takes priority over the HDR secondary server. The HDR secondary server takes priority over the RS secondary server.
- If the primary server fails, the Connection Managers can still connect to the cluster after restarting.

To configure connection management:

1. Install at least two network interface cards on each host. This prevents the failure of a network interface card from causing Connection Manager or database server failure.
2. Install two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on. This installation strategy prevents a Connection Manager from becoming a single point of failure, and prevents the simultaneous failure of database servers and Connection Managers if a host fails.
3. On each Connection Manager host, set the INFORMIXDIR environment to the directory the Connection Manager was installed into. Run the following command:

```
setenv INFORMIXDIR path
```

4. Create a configuration file in each Connection Manager installation's $INFORMIXDIR/etc directory.
The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

```
NAME connection_manger_1
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm1_log.log
LOCAL_IP 192.0.0.2,192.0.2.1

REPLSET replicate_set_1
{
        INFORMIXSERVER  g_server_1,g_server_2,g_server_3,g_server_4
        SLA report_1    DBSERVERS=ANY \
             POLICY=LATENCY
}

CLUSTER cluster_1
```

```
{
   INFORMIXSERVER g_server_1
   FOC ORDER=ENABLED \
       PRIORITY=1
   CMALARMPROGRAM $INFORMIXDIR/etc/CMALARMPROGRAM.sh
}
```

The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

```
NAME connection_manger_2
LOG 1
LOGFILE $INFORMIXDIR/tmp/my_cm2_log_.log
LOCAL_IP 192.0.2.2,192.0.2.3

REPLSET replicate_set_1
{
        INFORMIXSERVER  g_server_1,g_server_2,g_server_3,g_server_4
        SLA report_2    DBSERVERS=ANY \
             POLICY=LATENCY
}
CLUSTER cluster_1
{
   INFORMIXSERVER g_server_1
   FOC ORDER=ENABLED \
       PRIORITY=2
   CMALARMPROGRAM $INFORMIXDIR/etc/CMALARMPROGRAM.sh
}
```

The configuration file specifies the following information and behavior:

- Logging is enabled, and the log files are $INFORMIXDIR/tmp/my_cm1_log.log and $INFORMIXDIR/tmp/my_cm2_log.log.
- **connection_manager_1** monitors 192.0.2.0 and 192.0.2.1 and **connection_manager_2** monitors 192.0.2.2 and 192.0.2.3 for network failure.
- When the Connection Managers start, they each search their sqlhosts files for **g_server_1**, **g_server_2**, **g_server_3**, and **g_server_4** entries, and then connect to the servers **server_1a**, **server_1b**, **server_1c**, **server_1d**,**server_2**, **server_3**, and **server_4** that are in those groups.
- `CONNECT TO @report_1` and `CONNECT TO @report_2` connection requests are directed to the replication server that has the lowest transaction latency.
- The connection between **connection_manager_1** and the primary server is prioritized over the connection between **connection_manager_2** and the primary server. Failover that would break the connectivity between **connection_manager_1** and the primary server is blocked.
- If failover processing fails after eight attempts, $INFORMIXDIR/etc/CMALARMPROGRAM.sh is called.

Certain parameters and attributes are not included in this configuration file, so the Connection Manager has the following default behavior:

- The EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for primary-server events before failover processing begins. The SECONDARY_EVENT_TIMEOUT parameter is not set, so the Connection Managers wait 60 seconds for secondary-server events before the Connection Manager disconnects from the secondary server.
- The HOST, NETTYPE, SERVICE, and SQLHOSTSOPT attributes of the SLA parameters are not set, so each Connection Manager uses connection information in local and remote sqlhosts files.
- The SQLHOSTS parameter is not set, so each Connection Manager first searches its local sqlhosts file, and then remote database server sqlhosts files for connectivity information related to **server_1**, **server_2**, **server_3**, and **server_4**.
- The WORKERS attributes of the SLA parameters are not set, so four worker threads are allocated to each of the SLAs.

5. Set the onconfig file DRAUTO configuration parameter on **server_1a**, **server_1b**, **server_1c**, and **server_1d** to `3`

   ```
   DRAUTO 3
   ```

   This setting specifies that a Connection Manager controls failover arbitration.
6. Set the onconfig file HA_FOC_ORDER configuration parameter on **server_1a** to `SDS,HDR,RSS`

   ```
   HA_FOC_ORDER SDS,HDR,RSS
   ```

   After the Connection Managers start, and connect to **server_1a**, the HA_FOC_ORDER value replaces the value of the `ORDER` attributes in each Connection Manager's configuration file.

   If **server_1a** fails, the Connection Managers attempt failover to the SD secondary server. If the SD secondary server is also unavailable, the Connection Managers attempt failover to the HDR secondary server. If the HDR secondary server is also unavailable, the Connection Managers attempt failover to the RS secondary server.

7. Add entries to thesqlhosts files on the hosts of each database server and Connection Manager.

   ```
   #dbservername  nettype   hostname  servicename  options
    g_server_1    group     -         -            i=1,c=1,e=server_1d
    server_1a     onsoctcp  host_1    port_1       g=g_server_1
    server_1b     onsoctcp  host_1    port_2       g=g_server_1
    server_1c     onsoctcp  host_2    port_3       g=g_server_1
    server_1d     onsoctcp  host_3    port_4       g=g_server_1

    g_server_2    group     -         -            i=2,e=server_2
    server_2      onsoctcp  host_4    port_5       g=g_server_2

   g_server_3    group     -         -            i=3,e=server_3
    server_3      onsoctcp  host_5    port_6       g=g_server_3

    g_server_4    group     -         -            i=4,e=server_4
    server_4      onsoctcp  host_6    port_7       g=g_server_4
   ```

8. Create a sqlhosts file on each client host.

   ```
   #dbservername  nettype    hostname   servicename   options
    report        group      -          -             c=1,e=report_2
    report_1      onsoctcp   cm_host_1  cm_port_3     g=report
    report_2      onsoctcp   cm_host_2  cm_port_4     g=report
   ```

   If a Connection Manager fails, client applications can still connect to the other Connection Manager because the **report** group is defined.

   `CONNECT TO @report` connection requests are directed through one of the Connection Managers to the replication server that has the lowest transaction latency.

9. Set each INFORMIXSQLHOSTS environment variable to the sqlhosts file location by running the **setenv** command on each Connection Manager and client host.

   ```
   setenv INFORMIXSQLHOSTS path_and_file_name
   ```

10. Turn on quality of data (QOD) monitoring by running the **cdr define qod** command.

    ```
    cdr define qod -c server_1a --start
    ```

    The command connects to **server_1a**, defines **server_1a** as a master server for monitoring data, and then turns on quality of data monitoring. **server_1a** monitors transaction latency for the replication servers in the grid.

11. Run the **oncmsm** utility on each Connection Manager host, to start each Connection Manager.
    On the host of **connection_manager_1**:

    ```
    oncmsm -c cm_1.cfg
    ```

    On the host of **connection_manager_2**:

    ```
    oncmsm -c cm_2.cfg
    ```

12. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Related reference:
Group information
Related information:
The oncmsm utility
HA_FOC_ORDER configuration parameter
INFORMIXSQLHOSTS environment variable
cdr define qod
cdr start qod

Example: Configuring connection management for untrusted networks

This example shows steps that are required to configure connection management for an untrusted network.

For this example, you have a high-availability cluster on an untrusted network. All hosts use UNIX operating systems. The cluster consists of four servers:

- A primary server (**server_1**)
- A shared-disk secondary server (**server_2**)
- An HDR secondary server (**server_3**)
- An RS secondary server (**server_4**)

To configure connection management:

1. Install at least two network interface cards on each host.
2. Install at least two Connection Managers. Install each Connection Manager onto a different host, and do not install the Connection Managers onto the hosts that database servers are installed on.
3. On each host Connection Manager host, set the INFORMIXDIR environment to the directory the Connection Manager was installed into. Run the following command:

   ```
   setenv INFORMIXDIR path
   ```

4. Create a configuration file in each Connection Manager installation's $INFORMIXDIR/etc directory.
   The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

   ```
   NAME connection_manger_1
   LOG 1
   LOGFILE $INFORMIXDIR/tmp/my_cm1_log.log
   LOCAL_IP 192.0.2.0,192.0.2.1

   CLUSTER cluster_1
   {
      INFORMIXSERVER cluster_1
      SLA oltp_1    DBSERVERS=primary
      SLA payroll_1 DBSERVERS=(PRI,HDR) \
                    POLICY=WORKLOAD
      SLA report_1  DBSERVERS=(SDS,HDR,RSS) \
                    POLICY=WORKLOAD
      FOC ORDER=ENABLED \
          PRIORITY=1
      CMALARMPROGRAM $INFORMIXDIR/etc/CMALARMPROGRAM.sh
   }
   ```

   The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

   ```
   NAME connection_manger_2
   LOG 1
   LOGFILE $INFORMIXDIR/tmp/my_cm2_log.log
   LOCAL_IP 192.0.2.2,192.0.2.3

   CLUSTER cluster_1
   {
      INFORMIXSERVER cluster_1
      SLA oltp_2    DBSERVERS=primary
      SLA payroll_2 DBSERVERS=(PRI,HDR)\
   ```

```
                POLICY=WORKLOAD
   SLA report_2  DBSERVERS=(SDS,HDR,RSS) \
                POLICY=WORKLOAD
   FOC ORDER=ENABLED \
       PRIORITY=2
   CMALARMPROGRAM $INFORMIXDIR/etc/CMALARMPROGRAM.sh
}
```

5. Set the onconfig file DRAUTO configuration parameter on all database servers to `3`, to specify that Connection Managers control failover arbitration.

```
DRAUTO 3
```

6. Set the onconfig file HA_FOC_ORDER configuration parameter on **server_1** to `SDS,HDR,RSS`

```
HA_FOC_ORDER SDS,HDR,RSS
```

7. Optional: Configure the cmalarmprogram script on each Connection Manager host.
8. Add entries to thesqlhosts files on **server_1** and **server_2**'s host, **server_3**'s host, and **server_4**'s host.

```
#dbservername   nettype    hostname   servicename   options
 server_1       onsoctcp   host_1     port_1        s=6
 a_server_1     onsoctcp   host_1     port_2

 server_2       onsoctcp   host_1     port_3        s=6
 a_server_2     onsoctcp   host_1     port_4

 server_3       onsoctcp   host_2     port_5        s=6
 a_server_3     onsoctcp   host_2     port_6

 server_4       onsoctcp   host_3     port_7        s=6
 a_server_4     onsoctcp   host_3     port_8
```

9. Create a sqlhosts file on each Connection Manager's host.

```
#dbservername   nettype    hostname   servicename   options
 cluster_1      group      -          -             c=1,e=a_server_4
 server_1       onsoctcp   host_1     port_1        s=6,g=cluster_1
 a_server_1     onsoctcp   host_1     port_2        g=cluster_1
 server_2       onsoctcp   host_1     port_3        s=6,g=cluster_1
 a_server_2     onsoctcp   host_1     port_4        g=cluster_1
 server_3       onsoctcp   host_2     port_5        s=6,g=cluster_1
 a_server_3     onsoctcp   host_2     port_6        g=cluster_1
 server_4       onsoctcp   host_3     port_7        s=6,g=cluster_1
 a_server_4     onsoctcp   host_3     port_8        g=cluster_1
```

10. In each database server's onconfig file, set the DBSERVERALIASES parameter to that database server's alias.
    The onconfig file entry for **server_1**:

```
DBSERVERALIASES a_server_1
```

    The onconfig file entry for **server_2**:

```
DBSERVERALIASES a_server_2
```

    The onconfig file entry for **server_3**:

```
DBSERVERALIASES a_server_3
```

    The onconfig file entry for **server_4**:

```
DBSERVERALIASES a_server_4
```

11. On one of the Connection Manager hosts, use a text editor to create an ASCII-text password file that contains security information. Save the file to the $INFORMIXDIR/tmp directory. For example, my_passwords.txt has the following entries:

```
cluster_1   a_server_1  user_1  password_1
cluster_1   a_server_2  user_2  password_2
cluster_1   a_server_3  user_3  password_3
cluster_1   a_server_4  user_4  password_4

server_1    a_server_1  user_1  password_1
server_2    a_server_2  user_2  password_2
server_3    a_server_3  user_3  password_3
server_4    a_server_4  user_4  password_4

a_server_1  a_server_1  user_1  password_1
a_server_2  a_server_2  user_2  password_2
a_server_3  a_server_3  user_3  password_3
a_server_4  a_server_4  user_4  password_4
```

12. On the host where the password file is saved, run the **onpassword** utility with a specified encryption key to encrypt the password and create passwd_file in the $INFORMIXDIR/etc directory. For example, run the following command, specifying **my_secret_encryption_key_456** as your encryption key:

```
onpassword -k my_secret_encryption_key_456 -e my_passwords.txt
```

13. Store the original text file and encryption key in a safe place.
14. Distribute $INFORMIXDIR/etc/passwd_file to all the database servers that Connection Managers connect to, and to all Connection Managers. For systems that use Enterprise Replication, also distribute $INFORMIXDIR/etc/passwd_file to all the database servers that the **cdr** utility connects to.
15. Create a sqlhosts file on each client host.

```
#dbservername  nettype    hostname   servicename   options
 oltp          group      -          -             c=1,e=oltp_2
 oltp_1        onsoctcp   cm_host_1  cm_port_1     g=oltp
 oltp_2        onsoctcp   cm_host_2  cm_port_2     g=oltp
```

```
report          group       -           -             c=1,e=report_2
report_1        onsoctcp    cm_host_1   cm_port_3     g=report
report_2        onsoctcp    cm_host_2   cm_port_4     g=report

payroll         group       -           -             c=1,e=payroll_2
payroll_1       onsoctcp    cm_host_1   cm_port_5     g=payroll
payroll_2       onsoctcp    cm_host_2   cm_port_6     g=payroll
```

16. Set each INFORMIXSQLHOSTS environment variable to the sqlhosts file location by running the **setenv** command on each Connection Manager and client host.

    ```
    setenv INFORMIXSQLHOSTS path_and_file_name
    ```

17. Run the **oncmsm** utility on each Connection Manager host, to start each Connection Manager.
    On the host of **connection_manager_1**:

    ```
    oncmsm -c cm_1.cfg
    ```

    On the host of **connection_manager_2**:

    ```
    oncmsm -c cm_2.cfg
    ```

18. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

Related tasks:
Creating a password file for connecting to database servers on untrusted networks
Related reference:
Group information
Related information:
The onpassword utility
DBSERVERALIASES configuration parameter

Example of configuring connection management for prioritizing connections and network monitoring

You can install Connection Managers on the hosts of application servers, and then prioritize the connections between specific application servers and the primary server of a high-availability cluster. This configuration allows the highest priority application server to maintain its connection to the cluster's primary server if a portion of the network fails.

You can configure failover for the following conditions:

- When the primary server becomes inoperative.
- When an application server loses network connectivity with the primary server.

If network monitoring and failover priority are enabled and a network failure occurs, an application server that loses connectivity to the primary server but maintains connectivity to a secondary server can, through a shared-host Connection Manager, initiate failover to the secondary server. If, however, failover would cause an application server with more priority to lose connectivity to the primary server, the Connection Managers block failover.

Network monitoring and failover priority are enabled by setting the following parameters and attributes in each Connection Manager's configuration file:

- The Connection Manager's LOCAL_IP parameter
- A CLUSTER connection-unit's SLA parameters
- A CLUSTER connection-unit's FOC parameter
- The FOC parameter's ORDER attribute
- The FOC parameter's PRIORITY attribute

1. Install at least two network interface cards on each host. This method prevents the failure of a network interface card from causing client, Connection Manager, or database server connectivity failure.
2. Install and configure Connection Managers on each application server's host.
   a. Set the LOCAL_IP parameter in each Connection Manager configuration file to the IP addresses of the host's NIC cards. For example:
      The first Connection Manager's configuration file is named **cm_1.cfg** and has the following entries:

      ```
      NAME connection_manager_1
      LOCAL_IP 192.0.2.0,192.0.2.1
      ```

      The second Connection Manager's configuration file is named **cm_2.cfg** and has the following entries:

      ```
      NAME connection_manager_2
      LOCAL_IP 192.0.2.2,192.0.2.3
      ```

   b. Create service-level agreements for each Connection Manager. For example:
      cm_1.cfg now has the following entries:

      ```
      NAME connection_manager_1
      LOCAL_IP 192.0.2.0,192.0.2.1

      CLUSTER my_cluster

      {
         INFORMIXSERVER my_servers
         SLA sla_1 DBSERVERS=PRI
         SLA sla_2 DBSERVERS=SDS,HDR,RSS
      }
      ```

cm_2.cfg now has the following entries:

```
NAME connection_manager_1
LOCAL_IP 192.0.2.2,192.0.2.3

CLUSTER my_cluster

{
   INFORMIXSERVER my_servers
   SLA sla_3 DBSERVERS=PRI
   SLA sla_4 DBSERVERS=SDS,HDR,RSS
}
```

c. Specify each application server's priority by setting the FOC parameter's PRIORITY attribute for each shared-host Connection Manager. A PRIORITY value must be a positive integer and unique among all the Connection Managers that are configured to manage a specific cluster. If you specify a PRIORITY value in a connection-unit definition, you must set the ORDER attribute to ENABLED, and specify the failover order in the primary server's HA_FOC_ORDER configuration parameter. For example:
The primary server has the following onconfig file entry:

```
HA_FOC_ORDER SDS,HDR,RSS
```

cm_1.cfg now has the following entries:

```
NAME connection_manager_1
LOCAL_IP 192.0.2.0,192.0.2.1

CLUSTER my_cluster

{
   INFORMIXSERVER my_servers
   SLA sla_1 DBSERVERS=PRI
   SLA sla_2 DBSERVERS=SDS,HDR,RSS
   FOC ORDER=ENABLED PRIORITY=1
}
```

cm_2.cfg now has the following entries:

```
NAME connection_manager_2
LOCAL_IP 192.0.2.2,192.0.2.3

CLUSTER my_cluster

{
   INFORMIXSERVER my_servers
   SLA sla_3 DBSERVERS=PRI
   SLA sla_4 DBSERVERS=SDS,HDR,RSS
   FOC ORDER=ENABLED PRIORITY=2
}
```

3. Create sqlhosts files that contain network connectivity information for each Connection Manager and database server host.
4. Optional: Create a password file if you configure secure ports for database servers.
5. Run the **oncmsm** utility on each Connection Manager host, to start each Connection Manager.
   On the host of **connection_manager_1**:

   ```
   oncmsm -c cm_1.cfg
   ```

   On the host of **connection_manager_2**:

   ```
   oncmsm -c cm_2.cfg
   ```

6. Check each Connection Manager's log file to verify that the Connection Manager started correctly.

The Connection Managers now initiate failover if a network failure occurs, and failover would not cause a higher-priority application server to lose its connectivity to the cluster's primary server.
Related reference:
Group information
FOC Connection Manager configuration parameter
LOCAL_IP Connection Manager configuration parameter
Related information:
The oncmsm utility
HA_FOC_ORDER configuration parameter

Example of configuring for an SSL connection

This example shows the steps to configure CM to listen for SSL connections from database clients.

The steps to perform depend on the encryption library used by the CM: GSKit or OpenSSL. This in turn depends on whether the CM is installed in the same directory as the database server. If co-located with the database server, then the CM uses the same encryption library as the server, i.e. GSKit. Otherwise, the CM uses OpenSSL.
Note: You may determine which encryption library is used by the CM by running the utility **onkstash** with command line parameter version: **onkstash -version**

- Example: Using the OpenSSL encryption library
  This example shows the steps to configure CM to listen for SSL connection using the OpenSSL encryption library.
- Example: Using the GSKit encryption library
  This example shows the steps to configure CM to listen for SSL connection using the GSKit encryption library.

- Configuring a CM to connect to Oninit using SSL
  To configure a CM to connect to oninit using SSL, follow the steps in Configuring a client for SSL connections at Configuring a client for SSL connections.
- Configuring a client to connect to a CM using SSL
  A client can be configured to connect to a CM via SSL by following the same steps used to configure a client to connect to an Informix server via SSL. In this case the CM acts like a server. Follow the steps in Configuring a client for SSL connections at Configuring a client for SSL connections.

Example: Using the OpenSSL encryption library

This example shows the steps to configure CM to listen for SSL connection using the OpenSSL encryption library.

Use the openssl utility of your OpenSSL installation.

1. Create a private key for the self-signed certificate:

   ```
   $ openssl genrsa -out cm1key.pem
   ```

   Create the self-signed certificate using the private key

   ```
   $ openssl req -new -x509 -key cm1key.pem -subj "CN=`hostname`" -days 3650 -out cm1cert.pem
   ```

   Put the private key and the self-signed certificate into a single PEM file

   ```
   $ cat cm1key.pem cm1cert.pem > filewithcertificatetoimport.pem
   ```

2. Create the keystore file to contain the private key and certificate that are contained in a PEM file:

   ```
   $ openssl pkcs12 -export -in filewithcertificatetoimport.pem -name cm1ListeningCert -passout pass:test -out cm1.p12
   ```

3. Create the stash file to contain the encrypted keystore password:

   ```
   onkstash cm1.p12 test
   ```

4. In cm1's config file set "SSL_LABEL" to the certificate's label:

   ```
   SSL_LABEL cm1ListeningCert
   ```


Example: Using the GSKit encryption library

This example shows the steps to configure CM to listen for SSL connection using the GSKit encryption library.

Use the gsk8capicmd utility of your GSKit installation.

1. To configure **cm1** to listen for an SSL connection, create a keystore file named **cm1.p12** in the CM's $INFORMIXDIR/ssl directory.

   ```
   $ gsk8capicmd -keydb -create -db cm1.p12 -pw test -type pkcs12 -expire 3650 -stash
   ```

   Note: Ensure that this file is owned by the user running oncmsm (usually informix) and has 600 permissions.
2. Obtain the certificate, either creating it or importing it from a PEM file. Note its label.

   ```
    Example command for creating a certificate in the keystore:
   $ gsk8capicmd -cert -create -db cm1.p12 -pw test -dn "CN=`hostname`" -size 2048 -label cm1ListeningCert -ca true -sigalg
   SHA256WithRSA

   Example command for importing a certificate in a PEM file into the keystore:
   $ gsk8capicmd -cert -add -db cm1.p12 -pw test -file filewithcertificatetoimport.pem -label cm1ListeningCert -format ascii
   -trust enable
   ```

3. In cm1's config file set "SSL_LABEL" to the certificate's label:

   ```
   SSL_LABEL cm1ListeningCert
   ```


Configuring a CM to connect to Oninit using SSL

To configure a CM to connect to oninit using SSL, follow the steps in Configuring a client for SSL connections at Configuring a client for SSL connections.

Configuring a client to connect to a CM using SSL

A client can be configured to connect to a CM via SSL by following the same steps used to configure a client to connect to an Informix server via SSL. In this case the CM acts like a server. Follow the steps in Configuring a client for SSL connections at Configuring a client for SSL connections.

Cluster failover, redirection, and restoration

To maintain availability, you must plan for the failover of primary servers, redirecting client connections from unavailable servers, and restoring the cluster to its original configuration after a failure.

- Failover configuration for high-availability clusters
  A failure in a high-availability cluster means that one of the servers is no longer available. If a primary server fails, you must promote a secondary server to the role of primary server. Connection Managers can be configured to perform automatic failover.
- Redirection and connectivity for data-replication clients
  When any server in a high-availability cluster becomes unavailable, any client connections to it should be redirected to an available server. The best way to handle connection redirection is to use Connection Manager to automatically reconnect client connections to the servers that you specify in service level agreements. Alternatively, you can control connection redirection with environment variables or connection information.
- Recover HDR and RS clusters after failure
  When you restore the HDR or RS cluster back to the original configuration, you might need to restore critical media, as well as restart and configure servers in the cluster.
- Recovering a shared-disk cluster after data is damaged
  If a shared-disk cluster fails, you must perform a restore of affected dbspaces. The type of restore that you need to perform depends on whether critical data is damaged.
- Recovering an SD cluster after the secondary server became the primary server
  If a secondary server in an SD cluster became the primary server after the original primary server failed, you can use a script to reestablish the original primary and convert the current primary server back to a secondary server.

Failover configuration for high-availability clusters

A failure in a high-availability cluster means that one of the servers is no longer available. If a primary server fails, you must promote a secondary server to the role of primary server. Connection Managers can be configured to perform automatic failover.

You must set the DRAUTO configuration parameter to specify how failover is performed, and then use one of the following failover options:

- Connection Manager
- ISV cluster management software
- Manual switchover

- Failover with ISV cluster management software
  You can use independent software vendor (ISV) cluster management software instead of the Connection Manager to manage failover processing in high-availability cluster environments.
- I/O fencing for shared file systems
  You can configure I/O fencing to protect shared resources in a high-availability cluster environment.
- Cluster failures
  A high-availability cluster failure is a loss of connection between the database servers in a cluster that can be caused by several different situations.

Related concepts:
Connection management through the Connection Manager
Related information:
DRAUTO configuration parameter

Failover with ISV cluster management software

You can use independent software vendor (ISV) cluster management software instead of the Connection Manager to manage failover processing in high-availability cluster environments.

If the primary server in a high-availability cluster encounters a problem that requires a secondary server to assume the role of the primary, it is important that, before performing the actual failover, disk I/O is prohibited on the failed primary server and is allowed on the new primary server. In addition, network access to the failed primary server must be prevented. This is especially true for SD secondary servers, where disk corruption can occur if these steps are not done correctly.

The mechanism for enabling disk I/O operations from a server in a high-availability cluster environment is known as *I/O Fencing*. I/O Fencing is configured using a callback script. When a failure of the primary server occurs, the failover process executes a callback script on the secondary server before the secondary server assumes the role of the primary server. The script calls any I/O specific commands to enable or disable disk access. The script enables write access to the shared disk on the server that is to become the primary server, and disables write access to the shared disk on the failed server.

Use the FAILOVER_CALLBACK configuration parameter to specify the name of the script to run when a database server transitions from a secondary server to a primary server, or from a secondary server to a standard server. A template script named **ifx_failover_callback.sh** (UNIX) or **ifx_failover_callback.bat** (Windows) is provided in the $INFORMIXDIR/etc directory. When configured, the script specified by FAILOVER_CALLBACK is executed before the secondary server is switched to a primary or standard server.

You can test the failover script by performing one of the following actions, depending on your type of high-availability cluster:

- Converting an SD secondary server into a primary server.
- If the DRAUTO configuration parameter is set to 0, shutting down the primary server and convert the HDR secondary server to standard mode.
- If the DRAUTO configuration parameter is set to 1, shutting down the primary server in an HDR pair.
- Shutting down the primary server in a remote stand-alone cluster and converting the RS secondary server to standard mode.

An `Invoking Failover Callback` message is in the online.log listing the path and file name of the failover script after it is run.

See the information about the FAILOVER_CALLBACK configuration parameter in the *IBM® Informix® Administrator's Reference*.

If the script specified by FAILOVER_CALLBACK fails (that is, if it returns a non-zero exit code), the failover of the secondary to the primary (or standard) server also fails. In this case, the DBA must manually perform the failover.

I/O fencing for shared file systems

You can configure I/O fencing to protect shared resources in a high-availability cluster environment.

A software or hardware malfunction might cause unresolved operations to be written to shared storage devices. I/O fencing can be used to isolate a server and prevent it from accessing shared storage. I/O fencing must be used if maintenance or testing is being performed on a server in a high-availability cluster. If a problem is detected on a server or within an application, the cluster manager can detect the problem and prevent the server from connecting to the shared data.

You can configure a script to run when a primary server fails. Fencing commands are called by setting the FAILOVER_CALLBACK configuration parameter, which runs a script when a failover is initiated.

Although I/O fencing is not required in order to use Informix® database software, configuring I/O fencing must be used to protect shared-disk systems from inadvertent loss

Types of I/O fencing

Several types of I/O fencing are available, including:

- Power fencing - Powers off nodes if a problem is detected.
- Fibre Channel Switch Fencing (requires SCSI-3 persistent group reservation) - Blocks a port on the fibre channel device by removing the problem node's reservation.

Perhaps the most common method of implementing I/O fencing is to use fibre channel fencing. The fibre channel switch supports the industry-standard SCSI-3 persistent group reservation (PR) technology. PR technology allows a set of systems to have temporary registrations with the disk and to coordinate a write-exclusive reservation with the disk containing the data.

In most cases, cluster manager software must be installed. The cluster manager software provides the drivers and utilities required to issue commands to the fibre channel switch. For example, the Linux Cluster Suite provides a script named **fence_scsi**. Sun Cluster provides a command named **scdidadm**.

Other fencing methods are also available depending on the cluster manger software used and different hardware capabilities.

The IBM® General Parallel File System (GPFS™) is a high performance shared disk clustered file system developed by IBM. The file system is available for use with AIX®, Linux, and Windows platforms. The file system can be used with the IBM HACMP™ Cluster Management Software. GPFS uses the SCSI PR fencing mechanism to support I/O fencing. Fencing issues must be resolved by the IBM GPFS support team.

Implementing I/O fencing

I/O fencing can be configured on several platforms, including:

- Linux
- Solaris
- AIX
- Windows

see the documentation provided by the manufacturer of your equipment for specific information about configuring I/O fencing.

Related information:
FAILOVER_CALLBACK configuration parameter

Cluster failures

A high-availability cluster failure is a loss of connection between the database servers in a cluster that can be caused by several different situations.

Any of the following situations might cause a cluster failure:

- Equipment failure or destruction
- A network failure
- An excessive processing delay on one of the database servers

The database server interprets either of the following conditions as a cluster failure:

- The DRTIMEOUT configuration parameter value was exceeded without confirmation of communication with other cluster servers.
- A database server in the cluster does not respond to the periodic messaging attempts over the network. Cluster servers ping each other even if the primary server does not send records to the secondary database servers.
  A cluster server pings other cluster servers at the interval specified by its DRTIMEOUT configuration parameter.

After a database server detects a cluster failure, it writes a message to its message log (for example, `DR: receive error`) and turns off data replication. If a cluster failure occurs, the connection between the two database servers is dropped and the secondary database server remains in read-only mode.

You can configure automatic switchover for HDR replication pairs by setting the DRAUTO configuration parameter to 1 or 2.

You can configure automatic failover for a high-availability cluster by configuring Connection Managers. Connection Managers have many advantages over automatic switchover, and can manage failover to SD and RS secondary servers, as well.

- Automatic switchover
  If the primary server in a HDR-availability system fails, the HDR secondary server can be automatically converted to either a standard or primary database server.
- Automatic switchover without a reliable network
- Manual switchover
  Manual switchover means that the administrator of the secondary database server changes the type of the secondary database server to standard.
- Connecting offline servers to the new primary server
  After failover, you must reconnect any offline secondary servers to the new primary server.

Related concepts:
Replication of primary-server data to secondary servers

Automatic switchover

If the primary server in a HDR-availability system fails, the HDR secondary server can be automatically converted to either a standard or primary database server.

If automatic switchover occurs, the HDR secondary server to rolls back open transactions, and then switches to online mode as either a primary or standard database server. Automatic switchover does not redirect client applications the new primary or standard database server.

Automatic switchover can be a better option than manual failover, but using Connection Managers to control failover for a high-availability cluster has even more advantages.

Automatic switchover requires a very stable network, and works only with primary servers and HDR secondary servers. Connection Managers can be configured to be tolerant of network instability, and can perform failover to HDR, RS, and SD secondary servers. Connection Managers can also prioritize connections between application servers and the primary server if a network failure occurs.

Automatic switchover does not redirect client connections from the old primary server to the new standard or primary server. Connection Managers can redirect clients to whichever secondary server becomes the primary server.

If you use automatic switchover, you must ensure that logical-log files are backed up, and that the HDR server has enough logical-log disk space to allow processing to continue without backing up logical-log files.

To configure automatic switchover or Connection Manager failover, you must set the DRAUTO configuration parameter on all database servers of a high-availability cluster.

Related concepts:
Redirection and connectivity for data-replication clients
Automatic switchover without a reliable network
Related reference:
Restart if the primary server fails

Automatic switchover without a reliable network

Although automatic switchover might seem to be the best solution, it is not appropriate for all environments.

Consider what might happen if the primary database server does not actually fail, but the secondary database server registers that the primary has failed. For example, if the secondary database server does not receive responses when it signals (pings) the primary database server because of a slow or unstable network, the secondary server assumes that the primary database server failed and switches automatically to the standard type. If the primary database server also does not receive responses when it signals the secondary database server, it assumes that the secondary database server failed and turns off data replication but remains in online mode. Now the primary database server and the secondary database server (switched to the standard type) are both in online mode.

If clients can update the data on both database servers independently, the database servers in the pair reach a state in which each database server has the logical-log records that are required by the other. In this situation, you must start again and perform initial data replication with a level-0 dbspace backup of one entire database server, as described in Starting HDR for the First Time. Therefore, if your network is not entirely stable, you might not want to use automatic switchover. HDR cannot be reinstated without the risk of losing transactions on the previous secondary server.

Related concepts:
Automatic switchover

Manual switchover

Manual switchover means that the administrator of the secondary database server changes the type of the secondary database server to standard.

The secondary database server rolls back any open transactions and then comes into online mode as a standard database server, so that it can accept updates from client applications.

Related concepts:
Changing the database server type

Connecting offline servers to the new primary server

After failover, you must reconnect any offline secondary servers to the new primary server.

If the primary server in a high-availability cluster fails, all online secondary servers are notified of the failure. The secondary servers connect to the new primary server and continue to operate. However, RS secondary servers and HDR secondary servers that are not online at the time of the failover do not receive the failover notification, and attempts to connect to the original (failed) primary server when the servers are back online. In this case, the primary server must be manually reset on those secondary servers.

Run the following commands on secondary servers that were offline when the failover occurred.

- For HDR secondary servers:

  ```
  oninit -PHY
  onmode -d secondary new_primary
  ```

  new_primary indicates the name of the current primary server.
- For RS secondary servers:

  ```
  oninit -PHY
  onmode -d RSS new_primary
  ```

  new_primary indicates the name of the current primary server.

Redirection and connectivity for data-replication clients

When any server in a high-availability cluster becomes unavailable, any client connections to it should be redirected to an available server. The best way to handle connection redirection is to use Connection Manager to automatically reconnect client connections to the servers that you specify in service level agreements. Alternatively, you can control connection redirection with environment variables or connection information.

If you do not use Connection Manager, you can automatically redirect clients to different database servers in a cluster by configuring applications to connect to the server group to which the servers belong. When you create a connection to a server group, by default the connection is made to the current primary server in the group. If replication is down because one of the servers failed, the connection is made to the server which is online (in Standard mode or Primary mode without a secondary server). You can also automate this action from within the application. Some of the client connectivity drivers included in the have specific mechanisms for automating redirection. For details, see the documentation.

When you design client applications, you must make some decisions on redirection strategies. Specifically, you must decide whether to handle redirection within the application and which redirection mechanism to use. The three different redirection mechanisms are as follows:

- Automatic redirection with the DBPATH environment variable
- Administrator-controlled redirection with the connectivity information
- User-controlled redirection with the INFORMIXSERVER environment variable

The mechanism that you employ determines which CONNECT syntax you can use in your application.

- Redirecting clients automatically with the DBPATH environment variable
  You can use the DBPATH environment variable in applications to redirect connections.
- Redirecting clients with the connectivity information
  You can redirect client connections to the new primary server by using sqlhosts information.
- Redirecting clients with the INFORMIXSERVER environment variable
  The INFORMIXSERVER environment variable redirection method can be used when an application does not explicitly specify a database server in the CONNECT statement, so that the client connects to the database server that the INFORMIXSERVER environment variable specifies.
- Redirecting clients with application code
  If you use the DBPATH environment variable or connectivity information to redirect connections, you can include in your clients a routine that handles errors when clients encounter a cluster failure. The routine can call another function that contains a loop that tries repeatedly to connect to the other database server in the cluster. This routine redirects clients without requiring the user to exit the application and restart it.
- Comparison of redirection methods
  The different redirection methods have different requirements.

Related concepts:
Connection management through the Connection Manager
Automatic switchover

Redirecting clients automatically with the DBPATH environment variable

You can use the DBPATH environment variable in applications to redirect connections.

What the administrator must do

Administrators take no action to redirect clients, but they might be required to attend to the type of the database server.

What the user must do

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities.

If your applications do not include such code, users who are running clients must quit and restart all applications.

- How the DBPATH redirection method works

How the DBPATH redirection method works

When an application does not explicitly specify a database server in the CONNECT statement, and the database server that the INFORMIXSERVER environment variable specifies is unavailable, the client uses the DBPATH environment variable to locate the database (and database server).

If one of the database servers in a replication pair is unusable, applications that use that database server are not required to reset their INFORMIXSERVER environment variable if their DBPATH environment variable is set to the other database server in the pair. Their INFORMIXSERVER environment variable must always contain the name of the database server that they use regularly, and their DBPATH environment variable must always contain the name of the alternative database server in the pair.

For example, if applications normally use a database server called **cliff_ol**, and the database server paired with **cliff_ol** in a replication pair is called **beach_ol**, the environment variables for those applications would be as follows:

```
INFORMIXSERVER  cliff_ol
DBPATH          //beach_ol
```

Because the DBPATH environment variable is read only (if required) when an application issues a CONNECT statement, applications must restart in order for redirection to occur.

An application can contain code that tests whether a connection has failed and, if so, attempts to reconnect. If an application has this code, you are not required to restart it.

You can use the CONNECT TO *database* statement with this method of redirection. For this method to work, you cannot use any of the following statements:

- CONNECT TO DEFAULT
- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

The reason for this restriction is that an application does not use DBPATH if a CONNECT statement specifies a database server. For more information about DBPATH, see the *IBM® Informix® Guide to SQL: Reference*.

Redirecting clients with the connectivity information

You can redirect client connections to the new primary server by using sqlhosts information.

The connectivity information-redirection method relies on the fact that when an application connects to a database server, it uses the connectivity information to find that database server.

If one of the database servers in a replication pair is unusable, an administrator can change the definition of the unavailable database server in the connectivity information. As described in Changing client connectivity information, the fields of the unavailable database server (except for the **dbservername** field) are changed to point to the remaining database server in the replication pair.

Because the connectivity information is read when a CONNECT statement is issued, applications might be required to restart for redirection to occur. Applications can contain code that tests whether a connection failed and that issues a reconnect statement, if necessary. If a connection failed, redirection is automatic, and you are not required to restart applications for redirection to occur.

Applications can use the following connectivity statements to support this method of redirection:

- CONNECT TO *database@dbserver*
- CONNECT TO *@dbserver*

Applications can also use the following connectivity statements, provided that the INFORMIXSERVER environment variable always remains set to the same database server name and the DBPATH environment variable is not set:

- CONNECT TO DEFAULT

- CONNECT TO *database*

On UNIX, the INFORMIXSQLHOSTS environment variable specifies the full path name and file name of the connection information in $INFORMIXDIR/etc/sqlhosts. For more information about INFORMIXSQLHOSTS, see the *IBM Informix Guide to SQL: Reference*.

On Windows, the connectivity information is in a key in the Windows registry.

- Changing client connectivity information
  To use the connectivity information to redirect clients, you must change the connectivity information for the clients and change other connectivity files, if necessary.
- Connecting to the database server
  After the administrator changes the connectivity information and other connectivity files (if required), clients connect to the database server to which the administrator redirects them when they issue their next CONNECT statement.
- Automatic redirection with server groups

Related concepts:
The sqlhosts file and the SQLHOSTS registry key
Trusted-host information
Trusted-user information

Changing client connectivity information

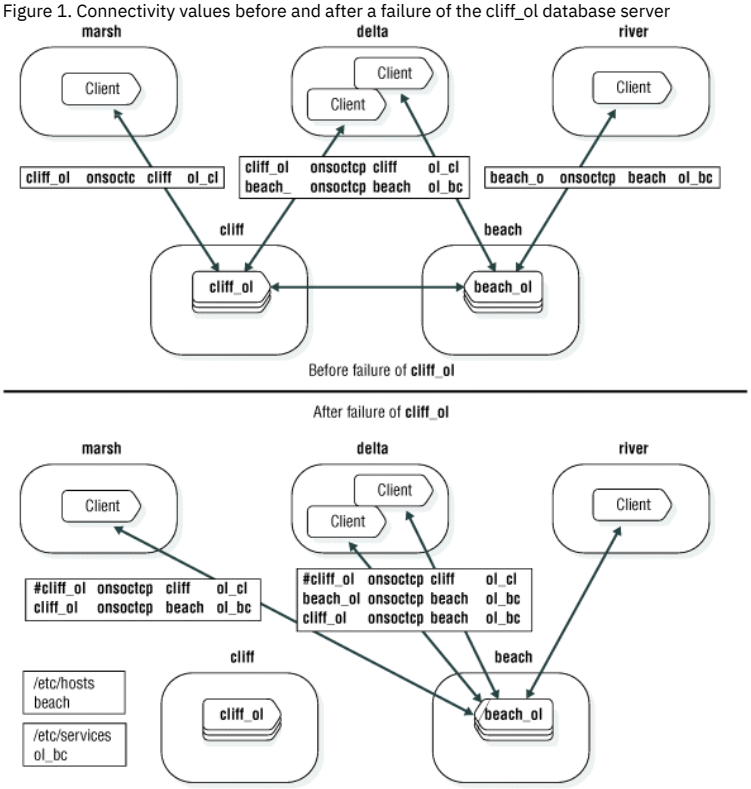
To use the connectivity information to redirect clients, you must change the connectivity information for the clients and change other connectivity files, if necessary.

To change the connectivity information about the client computer:

1. In the sqlhosts file or registry, comment out the entry for the failed database server.
2. Add an entry that specifies the dbservername of the failed database server in the **servername** field and information for the database server to which you are redirecting clients in the **nettype**, **hostname**, and **servicename** fields.
3. Use the following options in the sqlhosts file or registry to redirect applications to another database server if a failure occurs:
   a. Connection-redirection option
   b. End-of-group option
   c. Group option
4. Edit the /etc/hosts file on UNIX or hosts file on Windows to add an entry, if necessary, for the **hostname** of the computer that is running the database server to which you are redirecting clients.
5. Edit the /etc/services file on UNIX or services file on Windows to add an entry, if necessary, for the **servicename** of the database server to which you are redirecting clients.

The following figure shows how connectivity values might be modified to redirect clients.

You are not required to change entries in the connectivity information on either of the computers that is running the database servers.
Figure 1. Connectivity values before and after a failure of the cliff_ol database server



Related concepts:
Client/server communication
Related tasks:
Configuring secure connections for high-availability clusters

Related reference:
sqlhosts connectivity information

Connecting to the database server

After the administrator changes the connectivity information and other connectivity files (if required), clients connect to the database server to which the administrator redirects them when they issue their next CONNECT statement.

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities. If your applications do not include such code, users who are running clients must quit and restart all applications.

Automatic redirection with server groups

You can use the group option in the sqlhosts file to specify a server group to which applications connect instead of an individual database server. To make connection redirection automatic, add a database server definition for both the primary and secondary servers to the server group definition. By default, when a connection request is made to an HDR server group, the connection is routed to the primary server. If the primary server is unavailable, then the connection request is routed to the secondary server that gets promoted to the primary server after failover processing.

For example, the following sqlhosts entries represent an HDR server group, **g_hdr**, with a primary server definition, **hdr_prim**, and a secondary server definition, **hdr_sec**.

```
#dbservername   nettype    hostname      servicename   options
g_hdr           group      -             -             i=1
hdr_prim        ontlitcp   machine1pri   port1         g=g_hdr
hdr_sec         ontlitcp   machine1sec   port1         g=g_hdr
```

Applications can use the following connectivity statements to support this method of redirection:

- CONNECT TO *database@dbserver_group*
- CONNECT TO *@dbserver_group*

If your applications contain code that tests if a connection has failed and issues a reconnect statement if necessary, redirection is handled automatically. The user has no responsibilities. If your applications do not include such code, users who are running clients must quit and restart all applications.

Redirecting clients with the INFORMIXSERVER environment variable

The INFORMIXSERVER environment variable redirection method can be used when an application does not explicitly specify a database server in the CONNECT statement, so that the client connects to the database server that the INFORMIXSERVER environment variable specifies.

If one of the database servers in a cluster is unusable, applications that use that database server can reset their INFORMIXSERVER environment variable to the another database server in the cluster to access the same data.

Applications read the value of the INFORMIXSERVER environment variable only when they start. Therefore, applications must be restarted to recognize a change in the environment variable.

To support this method of redirection, you can use the following connectivity statements:

- CONNECT TO DEFAULT
- CONNECT TO *database*

You cannot use the CONNECT TO *database@dbserver* or CONNECT TO *@dbserver* statements for this method. When a database server is explicitly named, the CONNECT statement does not use the INFORMIXSERVER environment variable to find a database server.

Administrators take no action to redirect the clients, but they might be required to change the type of the database server.

Users who are running client applications must perform the following three steps when they decide to redirect clients with the INFORMIXSERVER environment variable.

To redirect clients with the INFORMIXSERVER environment variable:

1. Quit their applications.
2. Change their INFORMIXSERVER environment variable to hold the name of the other database server in the replication pair.
3. Restart their applications.

Redirecting clients with application code

If you use the DBPATH environment variable or connectivity information to redirect connections, you can include in your clients a routine that handles errors when clients encounter a cluster failure. The routine can call another function that contains a loop that tries repeatedly to connect to the other database server in the cluster. This routine redirects clients without requiring the user to exit the application and restart it.

The following example shows an example of a function in a client application using the DBPATH redirection mechanism that loops as it attempts to reconnect. After it establishes a connection, it also tests the type of the database server to make sure it is not a secondary database server. If the database server is still a secondary type, it calls another function to alert the user (or database server administrator) that the database server cannot accept updates.

```
/* The routine assumes that the INFORMIXSERVER environment
 * variable is set to the database server that the client
 * normally uses and that the DBPATH environment variable
 * is set to the other database server in the pair.
 */

#define SLEEPTIME 15
#define MAXTRIES 10

main()
{
   int connected = 0;
   int tries;
   for (tries = 0;tries < MAXTRIES && connected == 0;tries++)
   {
      EXEC SQL CONNECT TO "superstores";
      if (strcmp(SQLSTATE,"00000"))
      {
         if (sqlca.sqlwarn.sqlwarn6 != 'W')
         {
            notify_admin();
            if (tries < MAXTRIES - 1)
               sleep(SLEEPTIME);
            }
         else connected =1;
         }
      }
   return ((tries == MAXTRIES)? -1:0);
   }
```

This example assumes the DBPATH redirection mechanism and uses a form of the CONNECT statement that supports the DBPATH redirection method. If you used the connectivity information to redirect, you might have a different connection statement, as follows:

```
EXEC SQL CONNECT TO "superstores@cliff_ol";
```

In this example, `superstores@cliff_ol` is a database on a database server that the client computer recognizes. For redirection to occur, the administrator must change the connectivity information to make that name refer to a different database server. You might be required to adjust the amount of time that the client waits before it tries to connect or the number of tries that the function makes. Provide enough time for an administrative action on the database server (to change the connectivity information or change the type of the secondary database server to standard).

Comparison of redirection methods

The different redirection methods have different requirements.

The following tables summarize the differences among redirection mechanisms:

Table 1. Redirection methods for DBPATH connectivity

DBPATH	Automatic redirection	User redirection
When is a client redirected?	When the client next tries to connect with a specified database	When the client next tries to connect with a specified database
Do clients require being restarted to be redirected?	No	Yes
What is the scope of the redirection?	Individual clients redirected	Individual clients redirected
Are changes to environment variables required?	No	No

Table 2. Redirection methods for Connectivity information

Connectivity information	Automatic redirection	User redirection
When is a client redirected?	After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server	After the administrator changes the connectivity information, when the client next tries to establish a connection with a database server
Do clients require being restarted to be redirected?	No	Yes
What is the scope of the redirection?	All clients that use a given database server redirected	Individual clients redirected
Are changes to environment variables required?	No	No

Table 3. Redirection methods for INFORMIXDIR connectivity

INFORMIXDIR	Automatic redirection
When is a client redirected?	When the client restarts and reads a new value for the INFORMIXSERVER environment variable
Do clients require being restarted to be redirected?	Yes
What is the scope of the redirection?	Individual clients redirected
Are changes to environment variables required?	Yes

Table 4. Redirection methods for Connection Manager connectivity

Connection Manager	Automatic redirection
When is a client redirected?	When the configured service level agreement (SLA) is attained.
Do clients require being restarted to be redirected?	No
What is the scope of the redirection?	Individual clients redirected
Are changes to environment variables required?	No

Recover HDR and RS clusters after failure

When you restore the HDR or RS cluster back to the original configuration, you might need to restore critical media, as well as restart and configure servers in the cluster.

The result of a disk failure depends on whether the disk failure occurs on the primary or the secondary database server, whether the chunks on the disk contain critical media (the root dbspace, a logical-log file, or the physical log), and whether the chunks are mirrored.

If chunks are mirrored, you can perform recovery just as you would for a standard database server that used mirroring.

If chunks are not mirrored, the procedure for restoring a primary database server depends on whether the disk that failed contains critical media:

- If the disk contains critical media, the primary database server fails. You must perform a full restore using the primary dbspace backups (or the secondary dbspace backups if a secondary database server was switched to standard mode and activity redirected).
- If the disk does not contain critical media, you can restore the affected dbspaces individually with a warm restore. A warm restore consists of two parts: first a restore of the failed dbspace from a backup and next a logical restore of all logical-log records written since that dbspace backup. You must back up all logical-log files before you perform the warm restore.

If chunks are not mirrored, a secondary database server fails if the disk contains critical media but remains online if the disk does not contain critical media. In both cases, you must perform a full restore using the dbspace backups on the primary database server. In the second case, you cannot restore selected dbspaces from the secondary dbspace backup because they might now deviate from the corresponding dbspaces on the primary database server. You must perform a full restore.

- Recovering a cluster after critical data is damaged
  If one of the database servers in a high-availability cluster experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as if it has no data on the disks as is being started for the first time. Use the functioning database server with the intact disks as the database server with the data.
- Restarting HDR or RS clusters after a network failure
  After a network failure, the HDR or RS cluster might need to be restarted. After a network failure, the primary database server is in online mode, and the secondary database server is in read-only mode. Replication is turned off on both database servers (state = off).
- Restarting HDR or RS clusters if the secondary server fails
  If you must restart HDR or an RS cluster after a failure of a secondary server, in addition to starting the secondary server, you might need to perform a logical log restore on the secondary server.
- Recovering an HDR cluster after the secondary server became the primary server
  If a secondary server in an HDR cluster became the primary server after the original primary server failed, you can use a script to reestablish the original primary and convert the current primary server back to a secondary server.
- Restart if the primary server fails
  The process for restarting an HDR or RS cluster after the primary server fails depends on whether a secondary server became the primary server, and the method by which the secondary server became the primary server.

Related reference:
Recovering a cluster after critical data is damaged

Recovering a cluster after critical data is damaged

If one of the database servers in a high-availability cluster experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as if it has no data on the disks as is being started for the first time. Use the functioning database server with the intact disks as the database server with the data.

Primary server failure

For the following steps, assume that the configuration consists of a primary server named **srv_A** and an HDR secondary server named **srv_B**. The steps for restarting an RS cluster are the similar.

To restart HDR after a critical media failure:

1. The DRAUTO configuration parameter on **srv_B** affects what you do next

- If it is set to `0`, then you must convert the server to the primary server by running the **onmode -d make primary** command.
- If it is set to `1`, then convert the server to the primary server by running the **onmode -d make primary** command.
- If it is set to `2`, the secondary database server becomes a primary database server as soon as the connection ends when the old primary server fails.

2. Restore **srv_A** (the primary database server) from the last dbspace backup.
3. Use the **onmode -d** command to set **srv_A** to an HDR secondary database server and to start HDR.
   The **onmode -d** command starts a logical recovery from the logical-log files on **srv_B**. If logical recovery cannot complete because you backed up and freed logical-log files on **srv_B**, HDR does not start until you perform the next step.
4. Apply the logical-log files from **srv_B** (the new primary database server), which were backed up to tape. The HDR pair is now operational; however the roles of **srv_A** and **srv_B** are swapped. To swap **srv_A** and **srv_B** back to their original roles, follow the instructions: Recovering an HDR cluster after the secondary server became the primary server.

Table 1. Steps for restarting HDR after a critical media failure on the primary database server

Step	On the primary database server (svr_A)	On the secondary database server (svr_B)
1.		**onmode** command **onmode -d make primary srv_A**
2.	**ontape** command **ontape -p** ON-Bar command **onbar -r -p**	
3.	**onmode** command **onmode -d secondary srv_B**	
4.	**ontape** command **ontape -l** ON-Bar command **onbar -r -l**	

Secondary server failure

If the secondary database server suffers a critical media failure, recover the cluster by following the steps for starting a cluster for the first time.

Primary and secondary server failure

In the unfortunate event that both of the computers that are running database servers in a replication pair experience a failure that damages the root dbspace, the dbspaces that contain logical-log files or the physical log, you must restart the cluster.

To restart a high-availability cluster after a critical media failure on both database servers:

1. Restore the primary database server from the storage space and logical-log backup.
2. After you restore the primary database server, treat the other failed database server as if it had no data on the disks and you were starting the high-availability cluster for the first time.

Related concepts:
Recover HDR and RS clusters after failure
Related tasks:
Starting HDR for the First Time

Restarting HDR or RS clusters after a network failure

After a network failure, the HDR or RS cluster might need to be restarted. After a network failure, the primary database server is in online mode, and the secondary database server is in read-only mode. Replication is turned off on both database servers (state = off).

Restarting the cluster might not be necessary because the primary database server attempts to reconnect every 10 seconds and displays a message regarding the inability to connect every 2 minutes.

If the cluster does not restart automatically, when the connection is reestablished, you can restart the cluster by running the following commands on the secondary server:

```
onmode -d secondary primary_name
```


Restarting HDR or RS clusters if the secondary server fails

If you must restart HDR or an RS cluster after a failure of a secondary server, in addition to starting the secondary server, you might need to perform a logical log restore on the secondary server.

The steps assume that you have been backing up logical-log files on the primary database server as necessary since the failure of the secondary database server.

Table 1. Steps in restarting after a failure on the secondary database server

Step	On the primary	On the secondary
1.	The primary database server must be in online mode.	**oninit** If you receive the following message in the message log, continue with step 2: `DR: Start Failure recovery from tape`
2.		**ontape** command **ontape -l** ON-Bar command **onbar -r -l**

Recovering an HDR cluster after the secondary server became the primary server

If a secondary server in an HDR cluster became the primary server after the original primary server failed, you can use a script to reestablish the original primary and convert the current primary server back to a secondary server.

Suppose the primary server, named **srv_pri**, has encountered an error that has caused it to fail over to an HDR secondary server named **srv_hdr_sec**. At this point, the primary server is **srv_hdr_sec**, and any other secondary servers in the cluster are now pointing to **srv_hdr_sec**.

To restore the cluster to the way it was before **srv_pri** failed over, follow these steps:

1. Initialize **srv_pri** as the HDR secondary server by running the appropriate command:
   UNIX systems:

   ```
   $INFORMIXDIR/bin/hdrmksec.sh srv_hdr_sec
   ```

   Windows systems:

   ```
   hdrmksec.bat srv_hdr_sec
   ```

2. Change **srv_pri** to the primary server by running:

   ```
   onmode -d make primary srv_pri
   ```

   This command makes **srv_pri** the primary server, and redirects any other secondary servers in the cluster to point to the new primary server. The command also shuts down the old HDR primary (**srv_hdr_sec**) because only a single primary server can exist in a high-availability environment.
3. Initialize **srv_hdr_sec** as the HDR secondary server by running the following command:
   On UNIX systems:

   ```
   $INFORMIXDIR/bin/hdrmksec.sh srv_pri
   ```

   On Windows systems:

   ```
   hdrmksec.bat srv_pri
   ```


Restart if the primary server fails

The process for restarting an HDR or RS cluster after the primary server fails depends on whether a secondary server became the primary server, and the method by which the secondary server became the primary server.

The secondary database server was not changed to a standard database server

If you must restart an HDR or RS cluster after a failure of the primary database server if the secondary database server is not changed to standard, start the primary database server by using the **oninit** command.

The secondary database server was changed to a standard database server manually

If you must restart an HDR or RS cluster after a failure of the primary database server, and you have manually changed the secondary database server to be a standard database server, complete the steps in the following table.

Table 1. Steps to restart if you changed the secondary database server to standard

Step	On the primary database server	On the secondary database server
1.		**onmode -s** This step takes the secondary database server (now standard) to quiescent mode. All clients that are connected to this database server must disconnect. Applications that perform updates must be redirected to the primary.

Step	On the primary database server	On the secondary database server
2.		**onmode -d secondary** *prim_name*
3.	**oninit** If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the **oninit** command. If you have backed up and freed the logical-log files on the secondary, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 4). For **ontape** users: If you want to read the logical-log records over the network, set the logical-log tape device to a device on the computer that is running the secondary database server.	
4.	If you are prompted to recover logical-log records from tape, perform this step. **ontape** command **ontape -l** ON-Bar command **onbar -r -l**	

The secondary database server was changed to a standard database server automatically

If you must restart an HDR or RS cluster after a failure of the primary database server, and the secondary database server was automatically changed to a standard database server, complete the steps shown in the following table.

Table 2. Steps to restart if you changed the secondary database server to standard automatically

Step	On the primary database server	On the secondary database server
1.	**% oninit** If DRAUTO = `1`, the type of this database server is set to primary. If DRAUTO = `2`, the type of this database server is set to secondary when it is restarted. If all the logical-log records that were written to the secondary database server are still on the secondary database server disk, the primary database server recovers these records from that disk when you issue the **oninit** command. If logical-log files that you have backed up and freed are on the secondary database server, the records in these files are no longer on disk. In this case, you are prompted to recover these logical-log files from tape (step 2). For **ontape** users: • Set the logical-log tape device to a device on the computer running the secondary database server.	If DRAUTO = `1`, the secondary database server automatically goes through graceful shutdown when you bring the primary back up. This ensures that all clients are disconnected. The type is then switched back to secondary. Any applications that perform updates must be redirected back to the primary database server. If DRAUTO = `2`, the secondary database server switches automatically to primary. The old primary database server becomes a secondary database server after it restarts and connects to the other server and determines that it is now a primary database server.
2.	If you are prompted to recover logical-log records from tape, perform this step. **ontape** command **% ontape -l** ON-Bar command **onbar -r -l**	

Related concepts:
Automatic switchover

Recovering a shared-disk cluster after data is damaged

If a shared-disk cluster fails, you must perform a restore of affected dbspaces. The type of restore that you need to perform depends on whether critical data is damaged.

Related concepts:
Backup and restore with high-availability clusters

Critical data is damaged

If the primary server experiences a failure that damages the root dbspace, the dbspace that contains logical-log files, or the dbspace that contains the physical log, you must treat the failed database server as if it has no data on the disks. You must perform a full restore of the primary server. In this situation, the primary server and the SD secondary servers are offline.
To recover a shared-disk cluster after critical media failure:

1. Perform a full restore of the primary server. Run one of the following commands, depending on whether the backup was performed with ON-Bar or the **ontape** utility:
   - **onbar -r**
   - **ontape -r**

   The primary server restarts after the restore is complete.
2. Restart the SD secondary servers.

Alternatively, you can perform a cold restore of the critical dbspaces on the primary server, restart the SD secondary servers, and then perform a warm restore of non-critical dbspaces.

Critical data is not damaged

If a disk that does not contain critical media fails, you can restore the affected dbspaces with a warm restore. In this situation the primary server and the SD secondary servers are online.
To recover non-critical data in a shared-disk cluster:

1. Shut down and restart the SD secondary servers.
2. Perform a warm restore of the affected dbspaces. Run one of the following commands, depending on whether the backup was performed with ON-Bar or the **ontape** utility:
   - **onbar -r** with the names of the dbspaces to restore
   - **ontape -r -D** with the names of the dbspaces to restore

Recovering an SD cluster after the secondary server became the primary server

If a secondary server in an SD cluster became the primary server after the original primary server failed, you can use a script to reestablish the original primary and convert the current primary server back to a secondary server.

In this example, the primary server, named **srv_pri**, has failed over to an SD secondary server named **srv_sds_sec**. At this point, the primary server is **srv_sds_sec**, and any other secondary servers in the cluster are now pointing to **srv_sds_sec**. To restore the cluster to the way it was before **srv_pri** failed over, follow these steps:

1. If necessary, set the following parameters in the onconfig file of **srv_pri**:

   ```
   SDS_ENABLE 1
   SDS_PAGING <path 1>,<path 2>
   SDS_TEMPDBS <dbsname>,<dbspath>,<pagesize>,<offset>,<size>
   ```

   The *dbsname* value must be unique. In addition, the *dbsname* must be unique among all existing dbspaces, blobspaces, and sbspaces, including those (possibly disabled) temporary spaces that are inherited from a primary server. If you have multiple SD secondary servers, the *dbsname* value must be unique for each server and not shared with any other SD secondary server or the primary server. See Setting up a shared disk secondary server for more information about setting these parameters.

2. Initialize **srv_pri** as an SD secondary server by running the **oninit** command on **srv_pri**.
3. Perform a manual failover of **srv_pri** to make it the primary server:

   ```
   onmode -d make primary srv_pri
   ```

   The previous command removes **srv_sds_sec** from the cluster and makes **srv_pri** the primary server.
4. Restore **srv_sds_sec** as an SD secondary server by running the **oninit** command on **srv_sds_sec**.

Distributed data

Multiphase commit protocols

A *two-phase commit protocol* ensures that transactions are uniformly committed or rolled back across multiple database servers. You can use IBM® Informix® database servers with IBM Informix Enterprise Gateway products or transaction managers to manipulate data in non-Informix databases. Distributed queries across IBM Informix database servers support two-phase commit.

The *heterogeneous commit protocol* ensures that updates to one or more IBM Informix databases and one non-Informix database in a single transaction are uniformly committed or rolled back.

These topics contain information about the use of the two-phase commit protocol. For information about recovering manually from a failed two-phase commit transaction, see Manually recovering from failed two-phase commit.

These topics also contain information about using transaction support for XA-compliant, external data sources, which can participate in two-phase commit transactions. See Informix transaction support for XA-compliant, external data sources.

Transaction managers

Transaction managers support two-phase commit and roll back. For example, if your database is IBM® Informix®, your accounting system is Oracle, and your remittance system is Sybase, you can use a transaction manager to communicate between the different databases. You also can use transaction managers to ensure data consistency between IBM Informix or non-Informix databases by using distributed transactions instead of Enterprise Replication or High-Availability Data Replication.

TP/XA Library with a transaction manager

A *global transaction* is a distributed query where more than one database server is involved in the query. A global transaction environment has the following parts:

- The client application
- The resource manager (IBM® Informix® database server)
- The transaction manager (vendor software)

TP/XA is a library of functions that lets the database server act as a resource manager in the X/Open DTP environment. Install the TP/XA library as part of IBM Informix ESQL/C to enable communication between a third-party transaction manager and the database server. The X/Open environment supports large-scale, high-performance OLTP applications.

Use TP/XA when your database has the following characteristics:

- Data is distributed across multivendor databases
- Transactions include IBM Informix and non-Informix data

Related concepts:
XA in high-availability clusters

Microsoft Transaction Server (MTS/XA)

The database server supports the Microsoft Transaction Server (MTS/XA) as a transaction manager in the XA environment. To use MTS/XA, install IBM® Informix® Client Software Development Kit, the latest version of IBM Informix ODBC Driver, and MTS/XA. MTS/XA works on Windows. For more information, contact IBM Informix Technical Support, and see the *Informix Client Products Installation Guide* and the MTS/XA documentation.

Related concepts:
XA in high-availability clusters

Informix transaction support for XA-compliant, external data sources

The IBM® Informix® Transaction Manager, which is an integral part of , not a separate module, recognizes XA-compliant, external data sources. These data sources can participate in two-phase commit transactions.

The transaction manager runs support routines for each XA-compliant, external data source that participates in a distributed transaction at a particular transactional event, such as prepare, commit, or rollback. This interaction conforms to X/Open XA interface standards.

Transaction support for XA-compliant, external data sources, which are also called *resource managers*, enables you to:

- Create XA-compliant, external data source types and instances of XA-compliant, external data sources.

- Create or modify a user-defined routine (UDR), virtual table interface, or virtual index interface to enable XA-compliant data sources to provide data access mechanisms for external data from XA-compliant data sources.
  The MQ extension is an example of a set of UDRs that provide this type of external data access.

- Register XA-compliant, external data sources with .
- Unregister XA-compliant, external data sources.
- Use multiple XA-compliant, external data sources within the same global transaction.

The transaction coordination with an XA-compliant, external data source is supported only in IBM Informix logged databases and ANSI-compliant databases, since these databases support transactions. Transaction coordination with an XA-compliant, external data source is not supported in non-logged databases.

You can use the following DDL statements, which are extensions to SQL statements to manage XA data source types and data sources:

Statement	Description
CREATE XADATASOURCE TYPE	Creates a type of XA-compliant, external data source
CREATE XADATASOURCE	Creates an instance of an XA-compliant, external data source
DROP XADATASOURCE	Deletes an instance of an XA-compliant, external data source
DROP XADATASOURCE TYPE	Deletes a type of XA-compliant, external data source

For more information about these statements, see the *IBM Informix Guide to SQL: Syntax*.

The interaction between IBM Informix and an XA-compliant, external data source occurs through a set of user-defined XA-support routines, such as **xa_open**, **xa_end**, **xa_commit**, and **xa_prepare**. You create these support routines before using the CREATE XADATASOURCE TYPE statement. For more information, see the *IBM Informix DataBlade API Programmer's Guide*.

After you create an external XA-compliant data source, you can register the data source to a current transaction and you can unregister the data source using the **mi_xa_register_xadatasource()** or **ax_reg()** and **mi_xa_unregister_xadatasource()** or **ax_unreg()** functions. In a distributed environment, you must register a data source at the local, coordinator server. Registration is transient, lasting only for the duration of the transaction. For more information about using these functions, see the *IBM Informix DataBlade API Function Reference* and the *IBM Informix DataBlade API Programmer's Guide*.

Use the following **onstat** options to display information about transactions involving XA-compliant data sources:

The onstat option	What XA-compliant data source information this command displays
onstat -x	Displays information about XA participants in a transaction.
onstat -G	Displays information about XA participants in a global transaction.
onstat -g ses *session id*	Displays session information, including information about XA data sources participating in a transaction.

The IBM Informix MQ extension provides external data access mechanisms for XA data sources.

Related concepts:
XA in high-availability clusters
Related information:
MQ Messaging

XA in high-availability clusters

The X/Open Distributed Transaction Processing (DTP) Model allows an updatable secondary server in a high-availability cluster to serve as a resource manager in a distributed transaction.

There are three types of participants in any XA global transaction:

- Application Program (AP): Defines transaction boundaries and specifies actions that constitute the transaction branch.
- Resource Manager (RM): Provides access to the resources, such as databases.
- Transaction Manager (TM): Assigns identifier (XID) to transaction branches, monitors transaction progress, coordinates the transaction branches into completion and failure recovery.

In a high-availability cluster, sessions are able to create or attach to a transaction branch from any server in the cluster. For example, a transaction branch detached from server_1 can be attached from server_2. The application can connect to the cluster using the Connection Manager without tracking the server on which the transaction began. The transaction manager can also connect to the cluster using the Connection Manager to complete an XA transaction by committing the transaction, rolling it back, or forgetting it, for both loosely and tightly coupled transactions (see Loosely-coupled and tightly-coupled modes).

All global transaction branches started from a secondary server are redirected to the primary server using the existing proxy interface. The primary server starts and maintains all transaction branches and performs all requested work associated with the branches.

When an XA transaction is started on an updatable secondary server, a corresponding XA transaction is started on the primary server. The XA transaction on the primary server executes the full life cycle of an XA transaction (start, end, prepare, and commit or roll back). XA transactions on secondary servers are used to support queries that are not redirected to the primary server. When a call to the **xa_end()** function is issued, the XA transaction is freed and the user session is detached from the XA transaction. All XA transaction requests and all write operations issued within the XA transaction are redirected to the primary server.

The following features are specific to the Informix® XA implementation:

- All XA interface requests are available on updatable secondary servers (see Database updates on secondary servers.
- Starting, preparing, and committing or rolling back XA transactions from an updatable secondary server is supported.
- The **xa_recover()** function, which obtains a list of prepared transaction branches from a resource manager, is supported.
- XA transaction branch migration among high-availability cluster servers is supported. Any server in a cluster can attach to an XA transaction branch irrespective of whether the transaction branch was originated from it.

- XA clients and the Transaction Manager can connect to any high-availability cluster server using the Connection Manager (see Connection management through the Connection Manager).
- Redirection of XA requests from secondary servers to the primary server is supported.
- Transaction survival is supported for XA transactions, with the exception of transaction completion after failover (see Transaction completion during cluster failover).
- If a secondary server on which a redirected XA transaction is running fails, the transaction is rolled back.
- Support is provided for SQL transactions that run within the XA environment but which are outside the XA transaction

The following restrictions exist for XA transactions running on secondary servers:

- Resuming a suspended global transaction branch from a different user session (on the same or different secondary server) is not supported.
- A user session cannot attach to a global transaction branch that is associated with a different user session on another secondary server.
- XA transactions have the same restrictions as other data on secondary servers. See Database updates on secondary servers.
- XA transactions cannot be started on read-only secondary servers. If an application attempts to create a new XA transaction on a read-only secondary server, it receives XA error code XAER_RMERR. In addition, running xa_prepare(), xa_commit(), or xa_rollback() on a read-only secondary server returns error code XA_NOTA (-4).
- The following XA APIs are supported on read-only secondary servers:
  - xa_open()
  - xa_close()

Important: If you are using the .NET Framework with the Microsoft Transaction Server to manage XA transactions on a high availability cluster, you must use the TransactionScope class instead of the ServiceConfig class. The TransactionScope class is available in .NET Framework 3.5.

Related concepts:
High availability and scalability
Database updates on secondary servers

Related reference:
TP/XA Library with a transaction manager
Microsoft Transaction Server (MTS/XA)
Informix transaction support for XA-compliant, external data sources
Loosely-coupled and tightly-coupled modes

Loosely-coupled and tightly-coupled modes

The database server supports XA global transactions in loosely coupled and tightly coupled modes:

- *Loosely coupled mode* means that the different database servers coordinate transactions, but do not share resources. The records from all branches of the transactions display as separate transactions in the logical log.
- *Tightly coupled mode* means that the different database servers coordinate transactions and share resources such as locking and logging. The records from all branches of the transactions display as a single transaction in the logical log.

The Tuxedo Transaction Manager, provided by BEA systems, supports loosely coupled mode. Tuxedo operates on both UNIX and Windows.

Windows only: The MTS/XA Transaction Manager, which operates only on Windows, supports the tightly coupled mode. MTS tightly coupled transaction support on the database server includes:

- Support for application programs with two tiers (a business-logic layer and a data-access layer).
- Connection pooling and session pooling.

MTS tightly coupled transaction support does not affect existing loosely coupled-transaction support. The same database server can use both loosely coupled and tightly coupled transaction support at the same time.

MTS tightly coupled transaction support has the following restrictions:

- Temporary tables are limited to one transaction branch. Different transaction branches within one global transaction cannot share a temporary table.
- Different transaction branches within one global transaction cannot share cursors.
- Different transaction branches within one global transaction cannot share an isolation level or lock-wait mode. The isolation level and lock-wait mode of each transaction branch must be set individually or set to the default level. If you want the same isolation level for all transaction branches, you must use SQL to specify this information for each transaction branch.

For a complete list of supported transaction managers, contact your marketing representative.

Related concepts:
XA in high-availability clusters

Two-phase commit protocol

The two-phase commit protocol provides an automatic recovery mechanism in case a system or media failure occurs during execution of the transaction. The two-phase commit protocol ensures that all participating database servers receive and implement the same action (either to commit or to roll back a transaction), regardless of local or network failure.

If any database server is unable to commit its portion of the transaction, all database servers participating in the transaction must be prevented from committing their work.

- When the two-phase commit protocol is used

When the two-phase commit protocol is used

A database server automatically uses the two-phase commit protocol for any transaction that modifies data on multiple database servers.

For example, suppose three database servers that have the names **australia**, **italy**, and **france**, are connected, as shown in the following figure.
Figure 1. Connected database servers

If you run the commands shown in the following example, the result is one update and two inserts at three different database servers.

```
CONNECT TO stores_demo@italy
BEGIN WORK
   UPDATE stores_demo:manufact SET manu_code = 'SHM' WHERE manu_name = 'Shimara'
   INSERT INTO stores_demo@france:manufact VALUES ('SHM', 'Shimara', '30')
   INSERT INTO stores_demo@australia:manufact VALUES ('SHM', 'Shimara', '30')
COMMIT WORK
```


Two-phase commit concepts

Every global transaction has a *coordinator* and one or more *participants*, defined as follows:

- The coordinator directs the resolution of the global transaction. It decides whether the global transaction must be committed or stopped.
  The two-phase commit protocol always assigns the role of coordinator to the current database server. The role of coordinator cannot change during a single transaction. In the sample transaction in When the two-phase commit protocol is used, the coordinator is **italy**. If you change the first line in this example to the following statement, the two-phase commit protocol assigns the role of coordinator to **france**:

  ```
  CONNECT TO stores_demo@france
  ```

  Use the **onstat -x** option to display the coordinator for a distributed transaction. For more information, see Monitor a global transaction.

- Each participant directs the execution of one *transaction branch*, which is the part of the global transaction involving a single local database. A global transaction includes several transaction branches when:
  - An application uses multiple processes to work for a global transaction
  - Multiple remote applications work for the same global transaction

  In When the two-phase commit protocol is used, the participants are **france** and **australia**. The coordinator database server, **italy**, also functions as a participant because it is also doing an update.

The two-phase commit protocol relies on two kinds of communication, *messages* and *logical-log records*:

- Messages pass between the coordinator and each participant. Messages from the coordinator include a transaction identification number and instructions (such as `prepare to commit`, `commit`, or `roll back`). Messages from each participant include the transaction status and reports of action taken (such as `can commit` or `cannot commit`, `committed`, or `rolled back`).
- Logical-log records of the transaction are kept on disk or tape to ensure data integrity and consistency, even if a failure occurs at a participating database server (participant or coordinator).
  For more details, see Two-phase commit and logical-log records.

Phases of the two-phase commit protocol

In a two-phase commit transaction, the coordinator sends all the data modification instructions (for example, inserts) to all the participants. Then, the coordinator starts the two-phase commit protocol. The two-phase commit protocol has two parts, the *precommit phase* and the *postdecision phase*.

Precommit phase

During the precommit phase, the coordinator and participants perform the following dialog:

Coordinator
: The coordinator directs each participant database server to prepare to commit the transaction.

Participants
: Every participant notifies the coordinator whether it can commit its transaction branch.

Coordinator
: The coordinator, based on the response from each participant, decides whether to commit or roll back the transaction. It decides to commit only if all participants indicate that they can commit their transaction branches. If any participant indicates that it is not ready to commit its transaction branch (or if it does not respond), the coordinator decides to end the global transaction.

Postdecision phase

During the postdecision phase, the coordinator and participants perform the following dialog:

Coordinator
: The coordinator writes the commit record or rollback record to the coordinator's logical log and then directs each participant database server to either commit or roll back the transaction.

Participants
: If the coordinator issued a commit message, the participants commit the transaction by writing the commit record to the logical log and then sending a message to the coordinator acknowledging that the transaction was committed. If the coordinator issued a rollback message, the participants roll back the transaction but do not send an acknowledgment to the coordinator.

Coordinator
: If the coordinator issued a message to commit the transaction, it waits to receive acknowledgment from each participant before it ends the global transaction. If the coordinator issued a message to roll back the transaction, it does not wait for acknowledgments from the participants.

How the two-phase commit protocol handles failures

The two-phase commit protocol is designed to handle system and media failures in such a way that data integrity is preserved across all the participating database servers. The two-phase commit protocol performs an automatic recovery if a failure occurs.

- Types of failures that automatic recovery handles
- Administrator's role in automatic recovery
- Automatic-recovery mechanisms for coordinator failure
- Automatic-recovery mechanisms for participant failure

Types of failures that automatic recovery handles

The following events can cause the coordinating thread or the participant thread to terminate or hang, thereby requiring automatic recovery:

- System failure of the coordinator
- System failure of a participant
- Network failure
- Termination of the coordinating thread by the administrator
- Termination of the participant thread by the administrator

Administrator's role in automatic recovery

The only role of the administrator in automatic recovery is to bring the coordinator or participant (or both) back online after a system or network failure.
Important: A slow network cannot trigger automatic recovery. None of the recovery mechanisms described here go into effect unless a coordinator system fails, a network fails, or the administrator terminates the coordinating thread.

Automatic-recovery mechanisms for coordinator failure

If the coordinating thread fails, each participant database server must decide whether to initiate automatic recovery before it commits or rolls back the transaction or after it rolls back a transaction. This responsibility is part of the presumed-end optimization. (See Presumed-end optimization.)

Automatic-recovery mechanisms for participant failure

Participant recovery occurs whenever a participant thread precommits an item of work that is terminated before the two-phase commit protocol can be completed. The goal of participant recovery is to complete the two-phase commit protocol according to the decision reached by the coordinator.

Participant recovery is driven by either the coordinator or the participant, depending on whether the coordinator decided to commit or to roll back the global transaction.

Important: To support automatic recovery after a subordinate server is shut down or restarted while a cross-server transaction is open, the sqlhosts file must include an entry for every database server from which distributed operations might be initiated. During automatic recovery, the name of the coordinator is recovered from the logical logs, and the subordinate server reconnects with the coordinator to complete the transaction. Because the coordinator always identifies itself to the participants using the name that is in the DBSERVERNAME configuration parameter in its own onconfig file, the DBSERVERNAME setting of the coordinator must be an Internet protocol connection name known to the participants, but you can also define at least one DBSERVERALIASES setting with the correct connection protocol for connectivity between the coordinator and the subordinate servers. The subordinate server must be able to connect to the coordinator using either the DBSERVERNAME setting or a DBSERVERALIASES setting of the coordinator.

Presumed-end optimization

Presumed-end optimization is a term that describes how the two-phase commit protocol handles the rollback of a transaction.

Rollback is handled in the following manner. When the coordinator determines that the transaction must be rolled back, it sends a message to all the participants to roll back their piece of work. The coordinator does not wait for an acknowledgment of this message, but proceeds to close the transaction and remove it from shared memory. If a participant tries to determine the status of this transaction—that is, find out whether the transaction was committed or rolled back (during participant recovery, for example)—it does not find any transaction status in shared memory. The participant must interpret this as meaning that the transaction was rolled back.

Independent actions

An independent action in the context of two-phase commit is an action that occurs independently of the two-phase commit protocol. Independent actions might or might not be in opposition to the actions that the two-phase commit protocol specifies. If the action is in opposition to the two-phase commit protocol, the action results in an error or a *heuristic decision*. Heuristic decisions can result in an inconsistent database and require manual two-phase commit recovery. Manual recovery is an extremely complicated administrative procedure that you must try to avoid. (For an explanation of the manual-recovery process, see Manually recovering from failed two-phase commit.)

- Situations that initiate independent action
- Possible results of independent action
- The heuristic rollback scenario
- The heuristic end-transaction scenario
- Monitor a global transaction

Situations that initiate independent action

Independent action during a two-phase commit protocol is rare, but it can occur in the following situations:

- The participant's piece of work develops into a long-transaction error and is rolled back.
- An administrator stops a participant thread during the postdecision phase of the protocol with **onmode -z**.
- An administrator ends a participant transaction (piece of work) during the postdecision phase of the protocol with **onmode -Z**.
- An administrator ends a global transaction at the coordinator database server with **onmode -z** or **onmode -Z** after the coordinator issued a commit decision and became aware of a participant failure. This action always results in an error, specifically error -716.

Possible results of independent action

As mentioned earlier, not all independent actions are in opposition to the two-phase commit protocol. Independent actions can yield the following three possible results:

- Successful completion of the two-phase commit protocol
- An error condition
- A heuristic decision

If the action is not in opposition to the two-phase protocol, the transaction either commits or rolls back normally. If the action ends the global transaction prematurely, an error condition results. Ending the global transaction at the coordinator is not considered a heuristic decision. If the action is in opposition to the two-phase commit protocol, a heuristic decision results. All these situations are explained in the sections that follow.

- Independent actions that allow transactions to complete successfully
- Independent actions that result in an error condition
- Independent actions that result in heuristic decisions

Independent actions that allow transactions to complete successfully

Independent actions are not necessarily in opposition to the two-phase commit protocol. For example, if a piece of work at a participant database server is rolled back because it developed into a long transaction, and the coordinator issues a decision to roll back the global transaction, the database remains consistent.

Independent actions that result in an error condition

If you, as administrator at the coordinator database server, run either **onmode -z** (stop the coordinator thread) or **onmode -Z** (stop the global transaction) after the coordinator issues its final *commit* decision, you are removing all knowledge of the transaction from shared memory at the coordinator database server.

This action is not considered a heuristic decision because it does not interfere with the two-phase protocol; it is either acceptable, or it interferes with participant recovery and causes an error.

The action is acceptable any time that all participants are able to commit the transaction without difficulty. In this case, your action to end the transaction forcibly is superfluous. The indication that you ran **onmode -Z** reaches the coordinator only when the coordinator is preparing to terminate the transaction.

In practice, however, you would probably consider running **onmode -z** or **onmode -Z** at the coordinator database server only if you were attempting to hasten the conclusion of a global transaction that has remained open for an unusually long period. In this scenario, the source of the problem is probably a failure at some participant database server. The coordinator has not received acknowledgment that the participant committed its piece of work, and the coordinator is attempting to establish communication with the participant to investigate.

If you run either **onmode -z** or **onmode -Z** while the coordinator is actively trying to reestablish communication, the coordinating thread obeys your instruction to die, but not before it writes error -716 into the database server message log. The action is considered an error because the two-phase commit protocol was forcibly broken, preventing the coordinator from determining whether the database is consistent.

Stopping a global transaction at a coordinator database server is not considered a heuristic decision, but it can result in an inconsistent database. For example, if the participant eventually comes back online and does not find the global transaction in the coordinator shared memory, it rolls back its piece of work, thereby causing a database inconsistency.

Independent actions that result in heuristic decisions

Some independent actions can develop into heuristic decisions when *both* of the following conditions are true:

- The participant database server already sent a `can commit` message to the coordinator and then rolls back.
- The coordinator's decision is to commit the transaction.

When both conditions are true, the net result is a global transaction that is inconsistently implemented (committed by one or more database servers and rolled back by another). The database becomes inconsistent.

The following two heuristic decisions are possible:

- Heuristic rollback (described in The heuristic rollback scenario)
- Heuristic end transaction (described in The heuristic end-transaction scenario)

After a heuristic rollback or end transaction occurs, you might be required to perform manual recovery, a complex and time-consuming process. you must understand heuristic decisions fully in order to avoid them. Always be wary of running **onmode -z** or **onmode -Z** within the context of two-phase commit.

The heuristic rollback scenario

In a *heuristic rollback*, either the database server or the administrator rolls back a piece of work that has already sent a `can commit` message.

- Conditions that result in a heuristic rollback
- Results of a heuristic rollback

Conditions that result in a heuristic rollback

The following two conditions can cause a heuristic rollback:

- The logical log fills to the point defined by the LTXEHWM configuration parameter. (See the topics about configuration parameters in the *IBM® Informix® Administrator's Reference*.) The source of the long-transaction condition is a piece of work being performed on behalf of a global transaction.
- An administrator executes **onmode -z *session_id*** to stop a database server thread that is executing a piece of work being performed on behalf of a global transaction.

In either case, if the piece of work has already sent a `can commit` message to its coordinator, the action is considered a heuristic decision.

- Condition 1: Logical log fills to a high-watermark
- Condition 2: System administrator executes onmode -z

Condition 1: Logical log fills to a high-watermark

Under two-phase commit, a participant database server that is waiting for instructions from the coordinator is blocked from completing its transaction. Because the transaction remains open, the logical-log files that contain records associated with this transaction cannot be freed. The result is that the logical log continues to fill because of the activity of concurrent users.

If the logical log fills to the value of the long-transaction high-watermark (LTXHWM) while the participant is waiting, the database server directs all database server threads that own long transactions to begin rolling them back. If a piece of work that is precommitted is the offending long transaction, the database server has initiated a heuristic rollback. That is, this database server is rolling back a precommitted piece of work without the instruction or knowledge of the coordinator.

Under two-phase commit, the logical-log files that contain records associated with the piece of work are considered open until an ENDTRANS logical-log record is written. This type of transaction differs from a transaction involving a single database server where a rollback actually closes the transaction.

The logical log might continue to fill until the exclusive high-watermark is reached (LTXEHWM). If this happens, all user threads are suspended except those that are currently rolling back or currently committing. In the two-phase commit scenario, the open transaction prevents you from backing up the logical-log files and freeing space in the logical log. Under these specific circumstances, the logical log can fill completely. If this happens, the participant database server shuts down, and you must perform a data restore.

Condition 2: System administrator executes onmode -z

You, as administrator, can decide to initiate a heuristic rollback of a precommitted piece of work by running **onmode -z**. You might make this decision because you want to free the resources that are held by the piece of work. (If you stop the participant thread by running **onmode -z**, you free all locks and shared-memory resources that are held by the participant thread even though you do not end the transaction.)

Results of a heuristic rollback

These topics describe what happens at both the coordinator and participant when a heuristic rollback occurs and how this process can result in an inconsistent database:

1. At the participant database server where the rollback occurred, a record is placed in the database server logical log (type HEURTX). Locks and resources held by the transaction are freed. The participant thread writes the following message in the database server message log, indicating that a long-transaction condition and rollback occurred:

   ```
   Transaction Completed Abnormally (rollback):

   tx=address flags=0xnn
   ```

2. The coordinator issues postdecision phase instructions to commit the transaction.
   The participant thread at the database server where the heuristic rollback occurred returns error message -699 to the coordinator as follows:

   ```
   -699 Transaction heuristically rolled back.
   ```

   This error message is not returned to the application at this point; it is an internal notification to the coordinator. The coordinator waits until all participants respond to the commit instruction. The coordinator does not determine database consistency until all participants report.

3. The next steps depend on the actions that occur at the other participants. Two situations are possible.

- Situation 1: Coordinator issues a commit and all participants report heuristic rollbacks
- Situation 2: Coordinator issued a commit; one participant commits and one reports a heuristic rollback

Situation 1: Coordinator issues a commit and all participants report heuristic rollbacks

The coordinator gathers all responses from participants. If every participant reports a heuristic rollback, the following events occur as a consequence:

1. The coordinator writes the following message to its own database-server message log:

   ```
   Transaction heuristically rolled back.
   ```

2. The coordinator sends a message to all participants to end the transaction.
3. Each participant writes an ENDTRANS record in its logical-log buffer. (The transaction entry is removed from the transaction table.)
4. The coordinator returns error -699 to the application, as follows:

   ```
   -699 Transaction heuristically rolled back.
   ```

5. In this situation, all databases remain consistent.

Situation 2: Coordinator issued a commit; one participant commits and one reports a heuristic rollback

The coordinator gathers all responses from participants. If at least one participant reports a heuristic rollback and at least one reports an acknowledgment of a commit, the result is called a *mixed-transaction result*. The following events occur as a consequence:

1. The coordinator writes the following message to its own database server message log:

   ```
   Mixed transaction result. (pid=nn user=userid)
   ```

   The `pid` value is the user-process identification number of the coordinator process. The `user` value is the user ID associated with the coordinator process. Associated with this message are additional messages that list each of the participant database servers that reported a heuristic rollback. The additional messages take the following form:

   ```
   Participant database server dbservername heuristically rolled back.
   ```

2. The coordinator sends a message to each participant that heuristically rolled back its piece of work, directing each one to end the transaction.
3. Each participant writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the transaction table.)
4. The coordinator writes an ENDTRANS message in its logical-log buffer. (The transaction entry is removed from the shared-memory transaction table.)
5. The coordinator returns error -698 to the application, as follows:

   ```
   -698 Inconsistent transaction. Number and names of servers rolled back.
   ```

6. Associated with this error message is the list of participant database servers that reported a heuristic rollback. If many database servers rolled back the transaction, this list might be truncated. The complete list is always included in the message log for the coordinator database server.

In this situation, examine the logical log at each participant database server site and determine whether your database system is consistent. (See Determine if a transaction was implemented inconsistently.)

The heuristic end-transaction scenario

A *heuristic end transaction* is an independent action taken by the administrator to roll back a piece of work and remove all information about the transaction from the transaction table. The heuristic end-transaction process is initiated when the administrator executes the **onmode -Z *address*** command.

Whenever you initiate a heuristic end transaction by running **onmode -Z**, you remove critical information required by the database server to support the two-phase commit protocol and its automatic-recovery features. If you run **onmode -Z**, it becomes your responsibility to determine whether your networked database system is consistent.

- When to perform a heuristic end transaction
- How to use onmode -Z
- Action when the transaction is ended heuristically

When to perform a heuristic end transaction

You must run the **onmode -Z** option to initiate a heuristic end transaction in only one, rare, situation. This situation occurs when a piece of work that has been heuristically rolled back remains open, preventing your logical-log files from becoming free. As a result, the logical log is dangerously close to full.

In general, the coordinator issues its commit-or-rollback decision within a reasonable period of time. However, if the coordinator fails and does not return online to end a transaction that was heuristically rolled back at your participant database server, you might face a serious problem.

The problem scenario begins in this way:

1. The participant thread that is executing a piece of work on behalf of a global transaction has sent a `can commit` response to the coordinator.
2. The piece of work waits for instructions from the coordinator.
3. While the piece of work is waiting, the logical log fills past the long-transaction high-watermark.
4. The piece of work that is waiting for instructions is the source of the long transaction. The participant database server directs the executing thread to roll back the piece of work. This action is a heuristic rollback.
5. The participant continues to wait for the coordinator to direct it to end the transaction. The transaction remains open. The logical log continues to fill.

If the coordinator contacts the participant and directs it to end the transaction in a reasonable period of time, no problem develops. The serious problem arises if the heuristic rollback occurs at a participant database server and subsequently the coordinator fails, preventing the coordinator from directing the participant to end the transaction.

As a consequence, the transaction remains open. The open transaction prevents you from backing up logical-log files and freeing space in the logical log. As the logical log continues to fill, it might reach the point specified by the exclusive-access, long-transaction high-watermark (LTXEHWM). If this point is reached, normal processing is suspended. At some point after the high-watermark is reached, you must decide if the open transaction is endangering your logical log. The danger is that if the logical log fills completely, the database server shuts down, and you must perform a data restore.

You must decide whether to end the transaction and protect your system against the possibility of filling the logical log, despite all the problems associated with running **onmode -Z**, or to wait and see if communication with the coordinator can be reestablished in time to end the transaction before the logical log fills.

How to use onmode -Z

The **onmode -Z *address*** command is intended for use only if communication between the coordinator and the participant is broken. To ensure that communication is really broken, the **onmode -Z** command does not run unless the thread that was executing the piece of work has been dead for the amount of time specified by TXTIMEOUT. For more information about this option, see the *IBM® Informix® Administrator's Reference*.

The *address* parameter is obtained from **onstat -x** output. For more information about the **onstat -x** option, see the *IBM Informix Administrator's Reference*.

Action when the transaction is ended heuristically

When you run **onmode -Z**, you direct the **onmode** utility to remove the participant transaction entry, which is located at the specified address, from the transaction table.

Two records are written in the logical log to document the action. The records are type ROLLBACK and ENDTRANS, or if the transaction was already heuristically rolled back, ENDTRANS only. The following message is written to the participant database server message log:

```
(time_stamp) Transaction Completed Abnormally (endtx): tx=address flags:0xnn user username tty ttyid
```

The coordinator receives an error message from the participant where the **onmode -Z** occurred, in response to its COMMIT instruction. The coordinator queries the participant database server, which no longer has information about the transaction. The lack of a transaction-table entry at the participant database server indicates that the transaction committed. The coordinator assumes that the acknowledgment message was sent from the participant, but somehow it was not received. Because the coordinator does not know that this participant's piece of work did not commit, it does not generate messages indicating that the global transaction was inconsistently implemented. Only the administrator who ran the **onmode -Z** command is aware of the inconsistent implementation.

Monitor a global transaction

Use the **onstat -x** command to track open transactions and determine whether they have been heuristically rolled back.

For example, in the output, an `H` flag in the **flags** field identifies a heuristic rollback, the `G` flag identifies a global transaction, the `L` flag indicates loosely coupled mode, and the `T` flag indicates tightly coupled mode.

The **curlog** and **logposit** fields provide the exact position of a logical-log record. If a transaction is not rolling back, **curlog** and **logposit** describe the position of the most recently written log record. When a transaction is rolling back, these fields describe the position of the most recently "undone" log record. As the transaction rolls back, the **curlog** and **logposit** values decrease. In a long transaction, the rate at which the **logposit** and **beginlg** values converge can help you estimate how much longer the rollback is going to take.

For more information about and an example of **onstat -x** output, see the *IBM® Informix® Administrator's Reference*.

You also can use the **onstat -u** and **onstat -k** commands to track transactions and the locks that they hold. For details, see the monitoring transactions topics in your *IBM Informix Performance Guide*. For a description of the fields that **onstat -x** displays, see the *IBM Informix Administrator's Reference*.

On a secondary server, when transaction completion after failover is enabled (by setting the FAILOVER_TX_TIMEOUT configuration parameter), it is possible that two global transactions might have the same global transaction identifier: one is a local temporary global transaction, and the other is the global transaction that belongs to the recovery thread. A quick way to tell the real global transaction from the temporary transaction is that the real transaction has a B flag if the transaction has performed any operations. You can also check the owner of the transaction by using the **onstat -g ath** command. The temporary global transaction on the secondary server is deleted after the xa_end() function is called.

The following onstat utility example output illustrates XA transaction support on both primary and secondary servers in a high-availability cluster environment. The **onstat -x**, **onstat -G**, and **onstat -ath** commands are separately documented, but output from the combined **onstat -xG** command is of special interest for global transactions.

The examples show each state of a redirected transaction.

In the examples, the global transaction shown running on the secondary server is a temporary transaction. The temporary transaction is used to support the SQL statements performed on the secondary server (not transactions redirected to the primary server). The temporary transaction is only shown when a user thread is actively associated with the global transaction branch.

The following example shows output from the **onstat -xG** command run on a secondary server after an xa_start() function:

```
Transactions
                                                                                                  est.
address           flags userthread       locks  begin_logpos      current logpos    isol    rb_time  retrys coord
7000000104d4190   AT--G 7000000104a7b68  0      -                 -                 LC      -        0
7000000104d8bd0   ALB-G 7000000104a5aa8  1      180:0x0           180:0x4eb018      DIRTY   0:00     0

Global Transaction Identifiers
address            flags  isol    timeout  fID        gtl  bql  data
7000000104d4190    AT--G  COMMIT  0        5067085    15   4    000102030405060708090A0B0C0D0E0F000000
7000000104d8bd0    ALB-G  DIRTY   0        5067085    15   4    000102030405060708090A0B0C0D0E0F000000
```

Output from **onstat -g ath** run on the secondary server:

```
Threads:
 tid     tcb               rstcb             prty status               vp-class       name
 317     7000001500902c8   7000000104a7b68   1    cond wait  netnorm      1cpu         sqlexec
 84      7000001403a7dc0   7000000104a5aa8   3    sleeping secs: 1        5cpu         xchg_2.0
```

Output from **onstat -xG** run on the primary server:

```
Transactions
                                                                                                  est.
address           flags userthread       locks  begin_logpos      current logpos    isol    rb_time  retrys coord
7000000104d9e60   ATB-M 7000000104a8bc8  2      180:0x4ea018      180:0x4eb018      COMMIT  0:00     0

Global Transaction Identifiers
address            flags  isol    timeout  fID        gtl  bql  data
7000000104d9e60    AT--M  COMMIT  0        5067085    15   4    000102030405060708090A0B0C0D0E0F000000
```

The M flag in the previous example indicates that the global transaction was started from a secondary server. Global transactions started on the primary server display a G flag. The M flag is displayed only on the primary server.

Output from the **onstat -g ath|grep 7000000104a8bc8** command:

```
 196     70000013012d3a8   7000000104a8bc8  1    sleeping secs: 1        4cpu         proxyTh
```

The following example shows output from the **onstat -xG** command run on secondary server after the xa_end() function:

```
Transactions
                                                                                                  est.
address           flags userthread       locks  begin_logpos      current logpos    isol    rb_time  retrys coord
7000000104d8bd0   ALB-G 7000000104a5aa8  1      180:0x0           180:0x4ee018      DIRTY   0:00     0

Global Transaction Identifiers
address            flags  isol    timeout  fID        gtl  bql  data
7000000104d8bd0    ALB-G  DIRTY   0        5067085    15   4    000102030405060708090A0B0C0D0E0F000000
```

Output from the **onstat -xG** command run on the primary server:

```
Transactions
                                                                                                  est.
address           flags userthread       locks  begin_logpos      current logpos    isol    rb_time  retrys coord
7000000104d9e60   -TB-M 0                2      180:0x4ea018      180:0x4ee018      COMMIT  0:00     0


Global Transaction Identifiers
address            flags  isol    timeout  fID        gtl  bql  data
7000000104d9e60    -T--M  COMMIT  -1       5067085    15   4    000102030405060708090A0B0C0D0E0F000000
```

Output from the **onstat -xG** command run on the secondary server after running the xa_prepare() function:

```
Transactions
                                                                                                  est.
address           flags userthread       locks  begin_logpos      current logpos    isol    rb_time  retrys coord
7000000104d8bd0   ALX-G 7000000104a5aa8  1      180:0x0           180:0x4ef018      DIRTY   0:00     0

Global Transaction Identifiers
address            flags  isol    timeout  fID        gtl  bql  data
7000000104d8bd0    ALX-G  DIRTY   0        5067085    15   4    000102030405060708090A0B0C0D0E0F000000
```

Output from the **onstat -xG** command run on the primary server:

```
Transactions
                                                                                                  est.
address           flags userthread       locks  begin_logpos      current logpos    isol    rb_time  retrys coord
7000000104d9e60   -TX-M 0                2      180:0x4ea018      180:0x4ef018      COMMIT  0:00     0

Global Transaction Identifiers
address            flags  isol    timeout  fID        gtl  bql  data
7000000104d9e60    -TX-M  COMMIT  -1       5067085    15   4    0
```

Two-phase commit protocol errors

The following two-phase commit protocol errors require special attention from the administrator.

Error number
Description

-698
If you receive error -698, a heuristic rollback has occurred and has caused an inconsistently implemented transaction. The circumstances leading up to this event are described in Results of a heuristic rollback. For an explanation of how the inconsistent transaction developed and to learn the options available to you, see this information.

-699
If you receive error -699, a heuristic rollback has occurred. The circumstances leading up to this event are described in Results of a heuristic rollback. For an explanation of how the inconsistent transaction developed, see this information.

-716
If you receive error -716, the coordinating thread has been terminated by administrator action after it issued its final decision. This scenario is described under Independent actions that result in an error condition.

Two-phase commit and logical-log records

The database server uses logical-log records to implement the two-phase commit protocol. You can use these logical-log records to detect heuristic decisions and, if necessary, to help you perform a manual recovery. (See Manually recovering from failed two-phase commit.)

The following logical-log records are involved in distributed transactions:

- BEGPREP
- PREPARE
- TABLOCKS
- HEURTX
- ENDTRANS

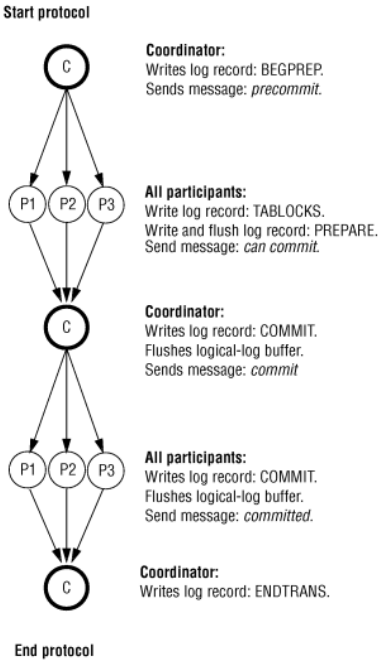
For information about these logical-log records, see the chapter on interpreting the logical log in the *IBM® Informix® Administrator's Reference*.

This section examines the sequence of logical-log records that are written during the following database server scenarios:

- Logical-log records when the transaction commits
- Logical-log records written during a heuristic rollback
- Logical-log records written after a heuristic end transaction

Logical-log records when the transaction commits

The following figure illustrates the writing sequence of the logical-log records during a successful two-phase commit protocol that results in a committed transaction.
Figure 1. Logical-log records written during a committed transaction



Some of the logical-log records must be flushed from the logical-log buffer immediately; for others, flushing is not critical.

The coordinator's commit-work record (COMMIT record) contains all information required to initiate the two-phase commit protocol. It also serves as the starting point for automatic recovery in the event of a failure on the coordinator's host computer. Because this record is critical to recovery, it is not allowed to remain in the logical-log buffer. The coordinator must immediately flush the COMMIT logical-log record.

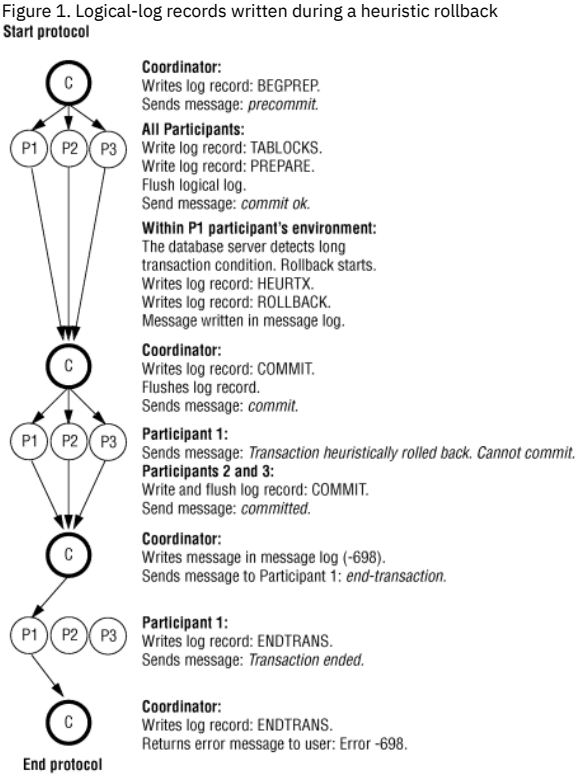
The participants in the preceding figure must immediately flush both the PREPARE and the COMMIT logical-log records. Flushing the PREPARE record ensures that, if the participant's host computer fails, fast recovery is able to determine that this participant is part of a global transaction. As part of recovery, the participant might query the coordinator to learn the final disposition of this transaction.

Flushing the participant's COMMIT record ensures that, if the participant's host computer fails, the participant has a record of what action it took regarding the transaction. To understand why this information is crucial, consider the situation in which a participant crashes after the PREPARE record is written but before the COMMIT record flushes. After fast recovery, the PREPARE record is restored, but the COMMIT record is lost (because it was in the logical-log buffer at the time of the failure). The existence of the PREPARE record would initiate a query to the coordinator about the transaction. However, the coordinator would know nothing of the transaction, because it ended the transaction after it received the participant's acknowledgment that the commit occurred. In this situation, the participant would interpret the lack of information as a final direction to roll back the transaction. The two-phase commit protocol requires the participant's COMMIT record to be flushed immediately to prevent this kind of misunderstanding.

Logical-log records written during a heuristic rollback

The following figure illustrates the sequence in which the database server writes the logical-log records during a heuristic rollback. Because a heuristic rollback only occurs after the participant sends a message that it can commit and the coordinator sends a message to commit, the first phase of this protocol is the same as that shown in Figure 1. When a heuristic rollback occurs, the rollback is assumed to be the consequence of a long-transaction condition that occurs at the Participant 1 (P1) database server. The end result is a transaction that is inconsistently implemented. See The heuristic rollback scenario.

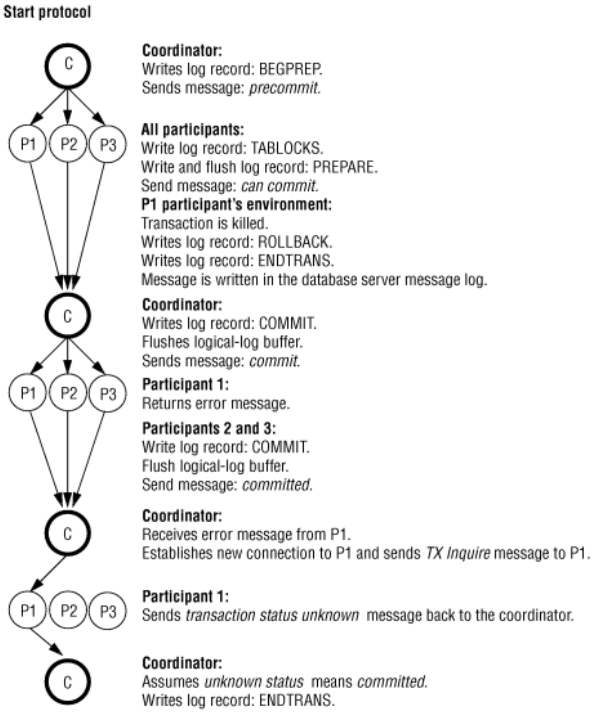
Figure 1. Logical-log records written during a heuristic rollback



Logical-log records written after a heuristic end transaction

The following figure illustrates the writing sequence of the logical-log records during a heuristic end transaction. The event is always the result of a database server administrator ending a transaction (see information about the **onmode** utility in the *IBM® Informix® Administrator's Reference*) at a participant database server after the participant has sent a `can commit` message. In the following figure, the heuristic end transaction is assumed to have occurred at the Participant 1 (P1) database server. The result is an inconsistently implemented transaction. See The heuristic end-transaction scenario.

Figure 1. Logical-log records written during a heuristic end transaction



Configuration parameters used in two-phase commits

The following two configuration-file parameters are specific to distributed environments:

- DEADLOCK_TIMEOUT
- TXTIMEOUT

Although both parameters specify timeout periods, the two are independent. For more information about these configuration parameters, see the *IBM® Informix® Administrator's Reference*.

- Function of the DEADLOCK_TIMEOUT parameter
- Function of the TXTIMEOUT parameter

Function of the DEADLOCK_TIMEOUT parameter

If a distributed transaction is forced to wait longer than the number of seconds specified by DEADLOCK_TIMEOUT for a shared-memory resource, the thread that owns the transaction assumes that a multiserver deadlock exists. The following error message is returned:

```
-154 ISAM error: deadlock timeout expired - Possible deadlock.
```

The default value of DEADLOCK_TIMEOUT is 60 seconds. Adjust this value carefully. If you set it too low, individual database servers end transactions that are not deadlocks. If you set it too high, multiserver deadlocks might reduce concurrency.

Function of the TXTIMEOUT parameter

The TXTIMEOUT configuration parameter is specific to the two-phase commit protocol. It is used only if communication between a transaction coordinator and participant has been interrupted and must be reestablished.

The TXTIMEOUT parameter specifies a period of time that a participant database server waits to receive a commit instruction from a coordinator database server during a distributed transaction. If the period of time specified by TXTIMEOUT elapses, the participant database server checks the status of the transaction to determine if the participant must initiate automatic participant recovery.

TXTIMEOUT is specified in seconds. The default value is `300` (five minutes). The optimal value for this parameter varies, depending on your specific environment and application. Before you modify this parameter, read the explanation How the two-phase commit protocol handles failures.

Heterogeneous commit protocol

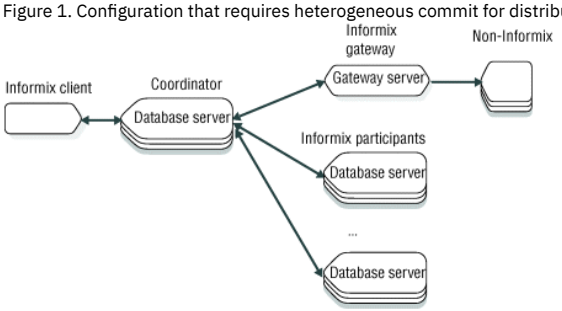
Used in the context of IBM® Informix® database servers, the term heterogeneous environment is a group of database servers in which at least one of the database servers is a different database server. Heterogeneous commit ensures the all-or-nothing basis of distributed transactions in a heterogeneous environment.

Unlike the two-phase commit protocol, the heterogeneous commit protocol supports the participation of a participant that is not Informix. The other participant, called a gateway participant, must communicate with the coordinator through the IBM Informix gateway.

The database server uses heterogeneous commit protocol when the following criteria are met:

- Heterogeneous commit is enabled. (That is, the HETERO_COMMIT configuration parameter is set to `1`.)
- The coordinator of the commit is a version 7.2 or later IBM Informix.
- The gateway participant communicates with the IBM Informix database server through the IBM Informix gateway.
- At most, one gateway participant performs an update within a single transaction.

The following figure illustrates this scenario.
Figure 1. Configuration that requires heterogeneous commit for distributed transactions



- Gateways that can participate in a heterogeneous commit transaction
- Enable and disable of heterogeneous commit
- How heterogeneous commit works
- Implications of a failed heterogeneous commit

Gateways that can participate in a heterogeneous commit transaction

A gateway acts as a bridge between the IBM® Informix® application (in this case, a database server) and a different database server. You can use a gateway to use IBM Informix applications to access and modify data that is stored in a database that is not IBM Informix.

The following table lists the gateways and corresponding database servers that can participate in a transaction in which the database server uses the heterogeneous commit protocol.

Table 1. Gateways and corresponding database servers/heterogeneous commit transaction

Gateway	Database servers
IBM Informix Enterprise Gateway with DRDA	IBM DB2®, OS/400®, SQL/DS
IBM Informix Enterprise Gateway for EDA/SQL	EDA/SQL
IBM Informix Enterprise Gateway Manager	Any database server with ODBC connectivity

Enable and disable of heterogeneous commit

Use a text editor to change the HETERO_COMMIT configuration parameter which enables or disables heterogeneous commit: The change takes effect when you shut down and restart the database server.

When you set HETERO_COMMIT to `1`, the transaction coordinator checks for distributed transactions that require the use of heterogeneous commit. When the coordinator detects such a transaction, it automatically executes the heterogeneous commit protocol.

If you set HETERO_COMMIT to `0` or any number other than `1`, the transaction coordinator disables the heterogeneous commit protocol. The following table summarizes which protocol the transaction coordinator uses, heterogeneous commit or two-phase commit, to ensure the integrity of a distributed transaction.

HETERO_COMMIT setting	Gateway participant updated	Database server protocol
Disabled	No	Two-phase commit
Disabled	Yes	Two-phase commit

HETERO_COMMIT setting	Gateway participant updated	Database server protocol
Enabled	No	Two-phase commit
Enabled	Yes	Heterogeneous commit

How heterogeneous commit works

The heterogeneous commit protocol is a modified version of the standard two-phase commit protocol. The postdecision phase in the heterogeneous commit protocol is identical to the postdecision phases in the two-phase commit protocol. The precommit phase contains a minor modification, and a new phase, called the gateway commit phase, is added to the heterogeneous commit protocol.

The following topics explain the modification to the precommit phase and the gateway commit phase. For a detailed explanation of the postdecision phases, see Postdecision phase.

- Precommit phase
- Gateway commit phase
- Heterogeneous commit optimization

Precommit phase

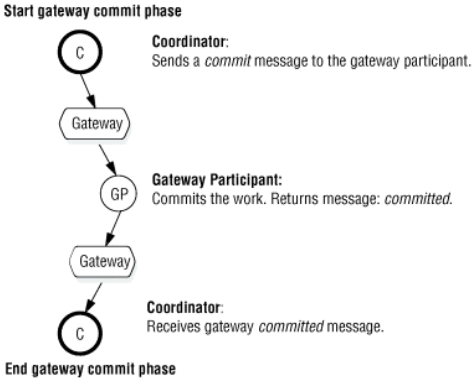
The coordinator directs each update participant (except the gateway participant) to prepare to commit the transaction.

If the updates satisfy all deferred constraints, all participants (except the gateway participant) return messages to the coordinator indicating that they can commit their piece of work.

Gateway commit phase

If all participants successfully return a message indicating that they are prepared to commit, the coordinator sends a commit message to the gateway. The gateway in turn sends a response to the coordinator indicating whether the gateway committed its piece of the transaction. If the gateway commits the transaction, the coordinator decides to commit the entire transaction. The following figure illustrates this process.

Figure 1. Heterogeneous commit phase that results in a committed transaction



If the gateway fails to commit the transaction, the coordinator rolls back the entire transaction, as the previous figure illustrates.

Heterogeneous commit optimization

The database server optimizes the heterogeneous commit protocol when the only participant that receives an update is a non-Informix® database. In this case, the coordinator sends a single commit message to all participants without invoking the heterogeneous commit protocol.

Implications of a failed heterogeneous commit

At any time during a distributed transaction that the database server processes using heterogeneous commit, the coordinator or any number of participants can fail. The database server handles these failures in the same way as in the two-phase commit protocol, except in certain instances. The following topics examine these special instances in detail.

- Database server coordinator failure
- Participant failure
- Interpretation of heterogeneous commit error messages

Database server coordinator failure

The consistency of data after a coordinator failure depends on the point in the heterogeneous commit process at which the coordinator fails. If the coordinator fails before sending the commit message to the gateway, the entire transaction is stopped upon recovery, as is the case with two-phase commit.

If the coordinator fails after it writes the commit log record, the entire transaction is committed successfully upon recovery, as is the case with two-phase commit.

If the coordinator fails after it sends the commit message to the gateway but before it writes the commit log record, the remote IBM® Informix® database server sites in the transaction are stopped upon recovery. This can result in inconsistencies if the gateway received the commit message and committed the transaction.

The following table summarizes these scenarios.

Point of database server coordinator failure	Expected result
After the coordinator writes the PREPARE log record and before the gateway commit phase	Data consistency is maintained.
After the coordinator sends a commit message to the gateway but before it receives a reply	Data is probably inconsistent. No indication of probable data inconsistency from the coordinator.
After gateway commit phase but before the coordinator writes a COMMIT record to the logical log	Data consistency is lost. No indication of data inconsistency from the coordinator.

Participant failure

Whenever a participant in a distributed transaction that uses the heterogeneous protocol fails, the coordinator sends the following error message:

```
-441    Possible inconsistent data at the target DBMS name due to an
aborted commit.
```

In addition, the database server sends the following message to the message log:

```
Data source accessed using gateway name might be in an inconsistent state.
```

A participant failure is not limited to the failure of a database server or gateway. In addition, a failure of the communication link between the coordinator and the gateway is considered a gateway failure. The gateway terminates if a link failure occurs. The gateway must terminate because it does not maintain a transaction log and therefore cannot reestablish a connection with the coordinator and resume the transaction. Because of this restriction, some scenarios exist in which a gateway failure might leave data in an inconsistent state. The following table summarizes these scenarios.

Point of participant failure	Expected result
After participant receives `commit transaction` message from coordinator, but before participant performs commit	Data consistency is maintained.
After participant receives `commit transaction` message from coordinator and commits the transaction, but before the participant replies to coordinator	Data is inconsistent.
After participant commits the transaction and sends a reply to coordinator	If the communications link fails before the coordinator receives the reply, then data is inconsistent. If the coordinator receives the reply, then data is consistent (provided the coordinator does not fail before writing the COMMIT record).

The recovery procedure that the database server follows when a participant fails is identical to the procedure that is followed in two-phase commit. For more information about this procedure, see How the two-phase commit protocol handles failures.

Interpretation of heterogeneous commit error messages

When the database server fails to process a distributed transaction using heterogeneous commit, it returns one of the two error messages that are explained in the following topics.

- Application attempts to update multiple gateway participants
- Failed attempt to commit distributed transaction using heterogeneous commit

Application attempts to update multiple gateway participants

If your client application attempts to update data at more than one gateway participant when HETERO_COMMIT is set to `1`, the coordinator returns the following error message:

```
-440 Cannot update more than one non-Informix DBMS within a transaction.
```

If you receive this error message, rewrite the offending application so that it updates at most one gateway participant in a single distributed transaction.

Failed attempt to commit distributed transaction using heterogeneous commit

The database server can fail to commit a distributed transaction while it is using the heterogeneous protocol for one or more of the following reasons:

- Communication error
- Site failure
- Gateway failure
- Other unknown error

When such a failure occurs, the coordinator returns the following message:

```
-441    Possible inconsistent data at the target DBMS name due to
an aborted commit.
```

After the database server sends this message, it rolls back all update sites that are participating in the transaction, with the possible exception of the work done at the site of the gateway participant. The gateway participant might have committed its updates if the failure occurred after the gateway participant processed the commit message. If the gateway participant committed the updates, you must manually roll back these updates.

Manually recovering from failed two-phase commit

Distributed transactions follow the two-phase commit protocol. Certain actions occur independently of the two-phase commit protocol and cause the transaction to be inconsistently implemented. (See Independent actions.) In these situations, it might be necessary to recover manually from the transaction.

- Determine if manual recovery is required
- Example of manual recovery

Determine if manual recovery is required

The following topics outline the steps in the procedure to determine database consistency and to correct the situation if required.

Each of these steps is described in the following topics.

- Determine if a transaction was implemented inconsistently
- Determine if the distributed database contains inconsistent data
- Decide if action is needed to correct the situation

Determine if a transaction was implemented inconsistently

Your first task is to determine whether the transaction was implemented inconsistently as a result of an independent action.

- Global transaction ended prematurely
- Heuristic end transaction
- Heuristic rollback

Global transaction ended prematurely

If you ran an **onmode -z** command to end the global transaction on the coordinator, the transaction might be inconsistently implemented. (For an explanation of how this situation can arise, see Independent actions that result in an error condition.) You can check for an inconsistent transaction by first examining the database server message log for the coordinator. Look for the following error message:

```
-716 Possible inconsistent transaction.
Unknown servers are server-name-list.
```

This message lists all the database servers that were participants. Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic end transaction

If you ran an **onmode -Z *address*** command to end a piece of work performed by a participant, and the coordinator decided to commit the transaction, the transaction is implemented inconsistently. (For a description of this scenario, see The heuristic end-transaction scenario.) Examine the logical log of each participant. If at least one participant performed a commit and one performed a rollback, the transaction was inconsistently implemented.

Heuristic rollback

You can determine the specific database server participants affected by a heuristic decision to roll back a transaction in the following ways:

- Examine the return code from the COMMIT WORK statement in the application.
  The following message indicates that one of the participants performed a heuristic rollback:

  ```
  -698 Inconsistent transaction. Number and names of servers rolled back.
  ```

- Examine the messages in the database server message-log file.
  If a database inconsistency is possible because of a heuristic decision at a participating database server, the following message is in the database server message-log file of the coordinator:

  ```
  Mixed transaction result. (pid=nn user=user_id)
  ```

  This message is written whenever error -698 is returned. Associated with this message is a list of the participant database servers where the transaction was rolled back. This is the complete list. The list that is created with the -698 error message might be truncated if many participants rolled back the transaction.

- Examine the logical log for each participant.
  If at least one participant rolls back its piece of work and one participant commits its piece of work, the transaction is implemented incorrectly.

Determine if the distributed database contains inconsistent data

If you determine that a transaction was inconsistently implemented, you must determine what this situation means for your distributed database system. Specifically, you must determine if data integrity has been affected.

A transaction that is inconsistently implemented causes problems whenever the piece of work rolled back by one participant is dependent on a piece of work that was updated by another participant. It is impossible to define these dependencies with SQL because distributed transactions do not support constraints that reference data at multiple database servers. The pieces of work are independent (no dependencies exist) only if the data could have been updated in two independent transactions. Otherwise, the pieces of work are considered to be dependent.

Before you proceed, consider the transaction that caused the error. Are the pieces of data that were updated and rolled back dependent on one another? Multiple updates might be included in a single transaction for reasons other than maintaining data integrity. For example, three possible reasons are as follows:

- Reduced transaction overhead
- Simplified coding
- Programmer preference

Verify also that every participant database server that is assumed to have committed the transaction actually modified data. A read-only database server might be listed as a participant that committed a transaction.

If an inconsistent transaction does not lead to a violation of data integrity, you can quit the procedure at this point.

- Obtaining information from the logical log
- Obtain the global transaction identifier

Obtaining information from the logical log

To determine if data integrity has been affected by an inconsistently implemented global transaction, you must reconstruct the global transaction and determine which parts of the transaction were committed and which were rolled back. Use the **onlog** utility to obtain the necessary information. The procedure is as follows:

1. Reconstruct the transaction at the participant that contains the HEURTX record.
    a. A participant database server logical log is the starting point for your information search. Each record in the log has a local transaction identification number (**xid**). Obtain the **xid** of the HEURTX record.
    b. Use the local **xid** to locate all associated log records that rolled back as part of this piece of work.
2. Determine which database server acted as coordinator for the global transaction.
    a. Look for the PREPARE record on the participant that contains the same local **xid**. The PREPARE record marks the start of the two-phase commit protocol for the participant.
    b. Use the **onlog -l** option to obtain the long output of the PREPARE record. This record contains the global transaction identifier (GTRID) and the name of the coordinating database server. For information about GTRID, see Obtain the global transaction identifier.
3. Obtain a list of the other participants from the coordinator log.
    a. Examine the log records on the coordinator database server. Find the BEGPREP record.
    b. Examine the long output for the BEGPREP record. If the first 32 bytes of the GTRID in this record match the GTRID of the participant, the BEGPREP record is part of the same global transaction. Note the participants displayed in the ASCII part of the BEGPREP long output.
4. Reconstruct the transaction at each participant.
    a. At each participant database server, read the logical log to find the PREPARE record that contains the GTRID associated with this transaction and obtain the local **xid** for the piece of work performed by this participant.
    b. At each participant database server, use the local **xid** to locate all logical-log records associated with this transaction (committed or rolled back).

After you follow this procedure, you know what all the participants for the transaction were, which pieces of work were assigned to each participant, and whether each piece of work was rolled back or committed. From this information, you can determine if the independent action affected data integrity.

Obtain the global transaction identifier

When a global transaction starts, it receives a unique identification number called a global transaction identifier (GTRID). The GTRID includes the name of the coordinator. The GTRID is written to the BEGPREP logical-log record of the coordinator and the PREPARE logical-log record of each participant.

To see the GTRID, use the **onlog -l** option. The GTRID is offset 20 bytes into the data portion of the record and is 144 bytes long. The following example shows the **onlog -l** output for a BEGPREP record. The coordinator is **chrisw**.

```
4a064     188  BEGPREP  4         0  4a038       0    1
          000000bc 00000043 00000004 0004a038 .......C .......8
          00087ef0 00000002 63687269 73770000 ..~..... chrisw..
          00000000 00000000 00000000 00087eeb ........ ......~.
          00006b16 00000000 00000000 00000000 ..k..... ........
          00000000 00000000 00000000 00000000
          00000000 00000000 00000000 00000000
          00000000 00000000 00000000 00000000
          00000000 00000000 00000000 00000000
          00000000 00000000 00000000 00000000
          00000000 00000000 00000000 00000000
          00000000 00000001 6a756469 74685f73 ........ judith_s
          6f630000 736f6374 63700000          oc..soct cp..
```

The first 32 bytes of the GTRID are identical for the BEGPREP record on the coordinator and the PREPARE records on participants, which are part of the same global transaction. For example, compare the GTRID for the PREPARE record in the following example with that of the BEGPREP record in the previous example.

```
c7064     184  PREPARE  4         0  c7038    chrisw
          000000b8 00000044 00000004 000c7038 .......D ......p8
          00005cd6 00000002 63687269 73770000 ......  chrisw..
          00000000 00000000 00000069 00087eeb ........ ...i..~.
          00006b16 00000000 00000010 00ba5a10 ..k..... ......Z.
          00000002 00ba3a0c 00000006 00000000 ......:. ........
          00ba5a10 00ba5a1c 00000000 00000000 ..Z...Z. ........
          00ba3a0e 00254554 00ba2090 00000001 ..:..%ET .. .....
          00000000 00ab8148 0005fd70 00ab8148 .......H ...p...H
          0005fe34 0000003c 00000000 00000000 ...4...< ........
          00000000 00ab80cc 00000000 00ab80c4 ........ ........
          00ba002f 63687269 73770000 00120018 .../chrisw......
          00120018 00ba0000                   ........
```


Decide if action is needed to correct the situation

If an inconsistent transaction creates an inconsistent database, the following three options are available to you:

- Leave the networked database in its inconsistent state.
- Remove the effects of the transaction wherever it was committed, thereby rolling back the entire transaction.
- Reapply the effects of the transaction wherever it was rolled back, thereby committing the transaction.

You can leave the database in its inconsistent state if the transaction does not significantly affect database data. You might encounter this situation if the application that is performing the transaction can continue as it is, and you decide that the price (in time and effort) of returning the database to a consistent state by either removing the effects or reapplying the transaction is too high.

You are not required to make this decision immediately. You can use the methods described in the following paragraphs to determine what data the transaction was updating and which records are affected.

As you make your decision, consider that no automatic process or utility can perform a rollback of a committed transaction or can commit part of a transaction that has been rolled back. The following paragraphs describe how to look through the database server message log and the logical log to locate affected records. Without detailed knowledge of the application, messages are not enough to determine what has happened. Based on your knowledge of your application and your system, you must determine whether to roll back or to commit the transaction. You must also program the compensating transaction that performs the rollback or the commit.

Example of manual recovery

This example illustrates the kind of work that is involved in manual recovery. The following SQL statements were executed by user **nhowe**. Error -698 was returned.

```
dbaccess
CREATE DATABASE tmp WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
CREATE DATABASE tmp@apex WITH LOG;
CREATE TABLE t (a int);
CLOSE DATABASE;
DATABASE tmp;
BEGIN WORK;
INSERT INTO t VALUES (2);
INSERT INTO tmp@apex:t VALUES (2);
COMMIT WORK;
### return code -698
```

The following excerpt is taken from the logical log at the current database server:

```
addr      len   type      xid      id   link
.....
17018    16         CKPOINT  0        0     13018    0

18018    20         BEGIN    2        1     0        08/27/91 10:56:57
   3482     nhowe

1802c    32         HINSERT  2        0     18018    1000018  102
   4

1804c    40         CKPOINT  0        0     17018    1

   begin    xid    id addr     user

   1        2          1  1802c    nhowe

19018    72         BEGPREP  2        0     1802c   6d69    1

19060    16         COMMIT   2        0     19018   08/27/91 11:01:38

1a018    16         ENDTRANS 2        0     19060   580543
```

The following excerpt is taken from the logical log at the database server **apex**:

```
addr      len   type      xid      id   link
.....
16018    20         BEGIN    2            1   0        08/27/91
   10:57:07 3483     pault

1602c    32         HINSERT  2        0     16018    1000018  102
   4

1604c    68         PREPARE  2        0     1602c      eh

17018    16         HEURTX   2        0     1604c          1

17028    12         CLR      2        0     1602c

17034    16         ROLLBACK 2        0     17018    08/27/91 11:01:22

17044    40         CKPOINT  0        0     15018    1

      begin    xid      id addr     user
      1        2        1  17034    --------

18018    16         ENDTRANS 2        0     17034     8806c3
   ....
```

First, you would try to match the transactions in the current database server log with the transactions in the **apex** database server log. The BEGPREP and PREPARE log records each contain the GTRID. You can extract the GTRID by using **onlog -l** and looking at the data portion of the BEGPREP and PREPARE log records. The GTRID is offset 22 bytes into the data portion and is 68 bytes long. A more simple, though less precise, approach is to look at the time of the COMMIT or ROLLBACK records. The times must be close, although there is a slight delay because of the time taken to transmit the commit (or rollback) message from the coordinator to the participant. (This second approach lacks precision because concurrent transactions can commit at the same time although concurrent transactions from one coordinator would probably not commit at the same time.)

To correct this sample situation

1. Find all records that were updated.

2. Identify their type (insert, delete, update) using **onlog** and the table of record types.
3. Use the **onlog -l** output for each record to obtain the local **xid**, the tblspace number, and the rowid.
4. Map the tblspace number to a table name by comparing the tblspace number to the value in the **partnum** column of the **systables** system catalog table.
5. Using your knowledge of the application, determine what action is required to correct the situation.

In this example, the time stamps on the COMMIT and ROLLBACK records in the different logs are close. No other active transactions introduce the possibility of another concurrent commit or rollback. In this case, an insert (HINSERT) of assigned rowid 102 hex (258 decimal) was committed on the current database server. Therefore, the compensating transaction is as follows:

```
DELETE FROM t WHERE rowid = 258
```


Overview of automatic monitoring and corrective actions

You can use the SQL administration API, the Scheduler, and drill-down queries to manage automatic maintenance, monitoring, and administrative tasks.

These components of IBM® Informix® simplify the collection of information and maintenance of the server in complex systems.

SQL administration API
: The SQL administration API performs remote administration through SQL functions. Because SQL administration API operations occur entirely in SQL, these functions can be used in client tools to administer the database server.

Scheduler
: The Scheduler is a set of tasks that execute SQL statements at predefined times or as determined internally by the server. The SQL statements can either collect information or monitor and adjust the server.

Query Drill-Down
: Query drill-down provides statistical information about recently executed SQL statements to track the performance of individual SQL statements and analyze statement history.

You can use the SQL administration API and the Scheduler on the primary server of an HDR pair of servers.

Each of these tools requires additional disk space to store information.

- The Scheduler
  You can use the Scheduler to create jobs to run administrative tasks or collect information at predictable times. The Scheduler uses SQL statements instead of operating system job scheduling tools.
- Remote administration with the SQL administration API
  You can use the SQL administration API to perform remote administration tasks by using SQL statements.
- Query drill-down
  You can use query drill-down, or SQL tracing, to gather statistical information about each SQL statement that was run and to analyze statement history.

Related concepts:
The Scheduler
Query drill-down
Related tasks:
Viewing SQL administration API history
Related reference:
Remote administration with the SQL administration API

The Scheduler

You can use the Scheduler to create jobs to run administrative tasks or collect information at predictable times. The Scheduler uses SQL statements instead of operating system job scheduling tools.

The Scheduler is controlled by a set of tables in the **sysadmin** database.

The Scheduler has four different job types that you can choose from:

Task
: Runs an action at a specific time and frequency.

Sensor
: Runs an action at a specific time and frequency to collect data, create a results table, store the data in the results table, and purge old data after a specified time.

Startup task
: A task that runs only when the server moves from quiescent mode to online mode.

Startup sensor
: A sensor that runs only when the server moves from quiescent mode to online mode.

The action of a task or sensor can be one or more SQL statements, user-defined routines, or stored procedures.

In addition to defining an action for a task or sensor, you can also use the Scheduler to:

- Associate tasks and sensors into functional groups
- Track the execution time and return value each time a task or sensor is run
- Define alerts with varying severity
- Define thresholds to control when tasks or sensors are run

The Scheduler contains built-in tasks and sensors that run automatically. You can modify the built-in tasks and sensors and define your own tasks and sensors.

Disk space requirements

The Scheduler tables and sensor results tables can use significant amounts of disk space.
You can use the following formula to estimate the disk usage for one sensor:

*Number of rows collected * size of the row collected * the frequency of data collection per day * the retention period*

Repeat this estimate for all sensors and you can determine a close estimate of the space required.

You can reduce the amount of data stored by decreasing the frequency of data collection or shortening the retention period by updating the **ph_task** table.

You can move the **sysadmin** database to a different dbspace by using the SQL administration API, however, all existing data in the database will be lost.

For more information about the **sysadmin** database, see the *IBM® Informix® Administrator's Reference*.

- Scheduler tables
  The Scheduler tables are located in the **sysadmin** database and contain information about tasks and sensors.
- Built-in tasks and sensors
  The Scheduler contains built-in tasks and sensors that run automatically.
- Creating a task
  You can create a Scheduler task to perform a specific action at specific times.
- Creating a sensor
  You can create a Scheduler sensor to collect and store data about the database server.
- Actions for task and sensors
  The action for a task or sensor is an SQL statement or routine that performs one or more operations.
- Creating a group
  You can create a group to organize Scheduler tasks and sensors.
- Creating a threshold
  You can create a threshold to determine under what conditions a Scheduler task or sensor is run.
- Creating an alert
  You can create an alert as part of the action of a Scheduler task or sensor.
- Monitor the scheduler
  You can monitor Scheduler threads that are in progress with the **onstat -g dbc** command. You can view information about tasks and sensors that have completed in the **ph_run** table.
- Modifying the scheduler
  You can modify the properties of Scheduler tasks, sensors, alerts, thresholds, or groups. You can modify both built-in properties and properties that you added.

Related concepts:
Overview of automatic monitoring and corrective actions
Related tasks:
Scheduling data optimization
Related information:
The Scheduler tables

Scheduler tables

The Scheduler tables are located in the **sysadmin** database and contain information about tasks and sensors.

The **sysadmin** database contains the Scheduler tables listed in the following table. The **ph_task** table has a direct relationship with each of the other tables.

Table 1. Scheduler tables

Table	Description
ph_alert	Contains a list of errors, warnings, or information messages associated with tasks that must be monitored. The **ph_alert** table contains built-in alerts that the database server uses automatically. You can add your own alerts.
ph_group	Contains a list of group names. Each task and sensor is a member of a group. The **ph_group** table contains built-in groups that the database server uses. You can add your own groups.
ph_run	Contains information about how and when each task and sensor was run.
ph_task	Lists tasks and sensors and contains information about how and when the database server runs them. The **ph_task** table contains built-in tasks and sensors that the database server uses automatically. You can add your own tasks and sensors.
ph_threshold	Contains a list of thresholds that are associated with tasks or sensors. If a threshold is met, the associated task can perform an action, such as inserting an alert in the **ph_alert** table. The **ph_threshold** table contains built-in thresholds that the database server uses. You can add your own thresholds.
results	Multiple tables that contain historical data collected by sensors. The structure of these tables is determined by the CREATE TABLE statement in the sensor definition in the **ph_task** table.

For details about these tables, see the *IBM® Informix® Administrator's Reference*.

Related information:
The Scheduler tables

Built-in tasks and sensors

The Scheduler contains built-in tasks and sensors that run automatically.

The following table shows the built-in Scheduler tasks and sensors. Sensors have results tables to store the information they collect, and retention periods to determine how long that information is stored. You can change task and sensor properties, for example, the frequency, by updating the **ph_task** table. Some tasks are triggered by thresholds. You can change thresholds by updating the **ph_threshold** table. You can disable a task or sensor by changing the value of the **tk_enable** column in the **ph_task** table to `f`.

You can determine how long tasks take by querying the **run_duration** column in the **ph_task** table.

Table 1. Built-in tasks and sensors

Task or sensor	Description	Results table	Frequency	Retention
add_storage	This task add more storage space automatically when automatic space management is configured.		As needed	
Alert Cleanup	This task removes all alert entries from the **ph_alert** table that are older than the threshold of 15 days. The threshold is named ALERT HISTORY RETENTION in the **ph_threshold** table.		Once a day	
auto_compress	This task compresses tables that are configured for automatic compression.			
auto_crsd	This task compresses, shrinks, repacks, and defragments tables and fragments. By default, this task is disabled. You must enable it by updating the **ph_task** table. Each of the operations has 2 rows in the **ph_threshold** table: one to control whether it is enabled and one to control its threshold. For more information, see Scheduling data optimization.		Once a week	
autoreg exe	This task registers database extensions when they are first used.		As necessary	
autoreg migrate-console	Internal. This task checks every database with a logging option of log or buffered log, and, if necessary, migrates all built-in database extensions to the correct version for the database server. This task creates sub-tasks for individual databases, as necessary.		At server startup	
autoreg vp	This task creates a specialized virtual processor for a database extension as necessary.		As necessary	
auto_tune_cpu_vps	This task automatically adds CPU virtual processors if the number of allocated VPs is less than half the number of CPU processors on the computer.		At server startup	
Auto Update Statistics Evaluation	This task analyzes all the tables in all logged databases, identifies the tables whose distributions must be updated, and generates UPDATE STATISTICS statements for those tables, based on the current automatic update statistics (AUS) policies. The AUS policies are set by thresholds in the **ph_threshold** table: • AUS_AGE: statistics are updated after 30 days. • AUS_CHANGE: statistics are updated after 10 percent of the data is changed. • AUS_AUTO_RULES: guidelines are followed for updating statistics. • AUS_SMALL_TABLES: tables containing fewer than 100 rows always have their statistics updated. For more information, see Automatic statistics updating.		Once a day	

Task or sensor	Description	Results table	Frequency	Retention
Auto Update Statistics Refresh	This task runs the UPDATE STATISTICS statements generated by the Auto Update Statistics Evaluation task. The PDQ priority for updating statistics is set to 10 by the threshold named AUS_PDQ in the **ph_threshold** table.		Saturday and Sunday between 1:00 AM and 5:00 AM	
bad_index_alert	This task checks for corrupted indexes. If any corrupted indexes are found, a warning alert is added to the **ph_alert** table. For more information, see Validate indexes.		Once a day	
bar_act_log_rotate	This task rotates the ON-Bar activity log file that is specified in the BAR_ACT_LOG configuration parameter. When the ON-Bar activity log rotates, the server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted. The threshold for the maximum logs to rotate is specified in the **ph_threshold** table.		3 A.M. every 30 days (with a maximum of 12 log files)	
bar_debug_log_rotate	This task rotates the ON-Bar debug log file that is specified in the BAR_DEBUG_LOG configuration parameter. When the ON-Bar debug log rotates, the server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted. The threshold for the maximum logs to rotate is specified in the **ph_threshold** table.		3 A.M. every 30 days (with a maximum of 12 log files)	
check_backup	This task checks to ensure that backups have run since the time specified by thresholds in the **ph_threshold** table: • REQUIRED LEVEL BACKUP: maximum of 2 days between backups of any level • REQUIRED LEVEL 0 BACKUP: maximum of 2 days between level-0 backups If a backup has not occurred, a warning alert is added to the **ph_alert** table.		Once a day	
check_for_ipa	This task adds an entry in the **ph_alert** table for each table that has one or more outstanding in-place alter operations.		Once a week	
idle_user_timeout	This task terminates user sessions that have been idle for longer than 60 minutes. By default, this task is disabled. You must enable it by updating the **ph_task** table. For more information, see Automatically terminating idle connections.		Every 2 hours	
ifx_ha_monitor_log_replay_task	This task monitors the high-availability cluster replay position.		Not set	
ifx_TrickleFeed_load_*ID*	This task continuously refreshed the data in a data mart. The name of the data mart and accelerator are listed in the task description. This task appears in the Scheduler after trickle feed is enabled for a data mart. Each data mart for which trickle feed is enabled has a separate task. The *ID* in the task name is unique. For more information, see ifx_setupTrickleFeed() function.		Every number of seconds that are specified when trickle feed is enabled	
mon_checkpoint	This sensor saves information about checkpoints.	**mon_checkpoint**	Every hour	7 days

Task or sensor	Description	Results table	Frequency	Retention
mon_chunk	This sensor saves general information about chunk usage and I/O chunk performance.	**mon_chunk**	Every hour	30 days
mon_command_history	This task deletes rows from the **command_history** table that are older than the threshold of 30 days. The threshold is named COMMAND HISTORY RETENTION in the **ph_threshold** table.		Once a day	
mon_compression_estimates	This sensor saves information about how much space might be saved if the data is compressed.	**mon_compression_ estimates**	Once a week	30 days
mon_config	This sensor saves the most recent value for each configuration parameter in the onconfig file.	**mon_config**	Once a day	
mon_config_startup	This sensor saves the value for each configuration parameter in the onconfig file when the server starts.	**mon_config**	At server startup	99 days
mon_iohistory	This sensor saves performance information about chunk I/O. You can change the IO_SAMPLES_PER_HOUR parameter in the **ph_threshold** table to collect information more frequently.		Every hour	30 days
mon_low_storage	This task scans the list of dbspaces to find spaces that fall below the threshold specified by the SP_THRESHOLD configuration parameter. Then, the task expands the spaces by extending chunks or adding chunks using entries in the storage pool. For more information, see Automatic space management.	**mon_low_storage**	Every hour	7 days
mon_memory_system	This sensor collects information about the amount of memory the server uses.	mon_memory_system	Every hour	7 days
mon_page_usage	This sensor saves information about the pages that are used and free in storage spaces.	**mon_page_usage**	Once a day	7 days
mon_profile	This sensor saves server profile information.	**mon_prof**	Every 4 hours	30 days
mon_sysenv	This startup sensor saves information about the environment when the database server starts.	**mon_sysenv**	At server startup	60 days
mon_table_names	This sensor saves table names along with their creation time.	**mon_table_names**	Once a day	30 days
mon_table_profile	This sensor saves table profile information, including the total number of update, insert, and delete operations that occurred on this table.	**mon_table_profile**	Once a day	7 days
mon_users	This sensor saves profile information about each user.	**mon_users**	Every 4 hours	7 days
mon_vps	This sensor collects virtual processor information.	**mon_vps**	Every 4 hours	15 days
online_log_rotate	This task rotates the online message log file that is specified in the MSGPATH configuration parameter. When the online message log rotates, the server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted. The threshold for the maximum logs to rotate is specified in the **ph_threshold** table.		3 A.M. every 30 days (with a maximum of 12 log files)	
post_alarm_message	This task posts alerts.		Every hour	
purge_tables	This task identifies rolling window tables whose purge policies have been exceeded. It discards or detaches qualifying fragments, according to each purge policy, until that policy is satisfied, or until no more fragments can be removed.		Daily at 00:45	

Task or sensor	Description	Results table	Frequency	Retention
SET tk_enable	This task enables the tasks that rotate message log files.		3 A.M. every 30 days	
sync_registry	This task automatically converts the connection information between the sqlhosts file format and the Windows registry format.		Every 15 minutes	

Related concepts:
Automatic performance tuning
Related tasks:
Modifying the scheduler
Related information:
The Scheduler tables

Creating a task

You can create a Scheduler task to perform a specific action at specific times.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
To create a task, use an INSERT statement to add a row into the **ph_task** table:

1. Include values for the following columns:
   a. **tk_name**: Give the task a unique name.
   b. **tk_type**: Change the job type to TASK or STARTUP TASK.
   c. **tk_description**: Add a description of the action the task performs.
   d. **tk_execute**: Add the action that the task performs. The action can be a user-defined function, a single SQL statement, or a multiple-statement prepared object that was created using the PREPARE SQL to enable the assembly of one or more SQL statements at runtime. The length of the command is limited to 2048 bytes.
2. Optional: Change the default values for the following columns:
   - **tk_start_time**: The default start time is 8:00:00. For a startup task, set the start time to NULL.
   - **tk_stop_time**: The default stop time is 19:00:00. For a startup task, set the stop time to NULL.
   - **tk_frequency**: The default frequency once a day. For a startup task, set the frequency to NULL.
   - **tk_group**: The default group is MISC.
   - **tk_monday** through **tk_sunday**: The default is to run every day.

The task runs at the specified start time and subsequently at the time calculated from the frequency.

Example

The following task uses the SQL administration API to take a checkpoint every two minutes between the hours of 8:00 A.M. and 7:00 P.M. on Mondays, Wednesdays, and Fridays.

```
INSERT INTO ph_task
( tk_name,
tk_description,
tk_type,
tk_group,
tk_execute,
tk_start_time,
tk_stop_time,
tk_frequency,
tk_Monday,
tk_Tuesday,
tk_Wednesday,
tk_Thursday,
tk_Friday,
tk_Saturday,
tk_Sunday)
VALUES
( "Example Checkpoint",
"Example to do a checkpoint every 2 minutes.",
"TASK",
"EXAMPLES",
"EXECUTE FUNCTION admin('checkpoint')",
DATETIME(08:00:00) HOUR TO SECOND,
DATETIME(19:00:00) HOUR TO SECOND,
INTERVAL ( 2 ) MINUTE TO MINUTE,
't',
'f',
't',
'f',
't',
'f',
'f');
```

The following example shows the code for a task that runs once a day at 2:00 A.M. to ensure that the **command_history** table contains only recent data. In this example, the definition of recent data is stored in a **Command History Interval** column in the **ph_threshold** table.

```
INSERT INTO ph_task
(
```

```
tk_name,
tk_group,
tk_description,
tk_type,
tk_execute,
tk_start_time,
tk_frequency
)
VALUES
(
"mon_command_history",
"TABLES",
"Monitor how much data is kept in the command history table",
"TASK",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY INTERVAL' ) ",
"2:00:00",
"1 0:00:00"
);
```

Related concepts:
Actions for task and sensors
Related tasks:
Modifying the scheduler

Creating a sensor

You can create a Scheduler sensor to collect and store data about the database server.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
To create a sensor, use an INSERT statement to add a row into the **ph_task** table:

1. Include values for the following columns:
   - **tk_name**: Give the task a unique name.
   - **tk_description**: Add a description of the action the task performs.
   - **tk_result_table**: Add the name of the table that holds the data that the sensor gathers.
   - **tk_create**: Add a CREATE statement to create the results table. The results table must have an INTEGER column named ID to hold the sensor ID. You can add other columns to the table.
   - **tk_execute**: Add the action that the sensor performs. The action can be a user-defined function, a single SQL statement, or a multiple-statement prepared object that was created using the PREPARE SQL to enable the assembly of one or more SQL statements at runtime.
2. Optionally change the default values for the following columns:
   - **tk_type**: The default value is SENSOR. For a startup sensor, change the value to STARTUP SENSOR.
   - **tk_delete**: The default interval after which to delete sensor data is one day.
   - **tk_start_time**: The default start time is 8:00:00. For a startup sensor, set the start time to NULL.
   - **tk_stop_time**: The default stop time is 19:00:00. For a startup sensor, set the stop time to NULL.
   - **tk_frequency**: The default frequency once a day. For a startup sensor, set the frequency to NULL.
   - **tk_group**: The default group is MISC.
   - **tk_monday** through **tk_sunday**: The default is to run every day.
   -

The sensor runs at the specified start time and subsequently at the time calculated from the frequency.

Examples

The following example shows the code for a sensor that tracks the startup environment of the database server. The $DATA_SEQ_ID variable is the current execution of the sensor.

```
INSERT INTO ph_task
(
tk_name,
tk_type,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete
)
VALUES
(
"mon_sysenv",
"STARTUP SENSOR",
"SERVER",
"Tracks the database servers startup environment.",
"mon_sysenv",
"create table mon_sysenv (ID integer, name varchar(250), value lvarchar(1024))",
"insert into mon_sysenv select $DATA_SEQ_ID, env_name, env_value
FROM sysmaster:sysenv",
```

```
NULL,
NULL,
NULL,
"60 0:00:00"
);
```

The following example shows the code for a sensor that collects information about the amount of memory that is being used and stores the information in the **mon_memory_system** table. If that table does not exist, the task creates it. This task, which runs every 30 minutes, deletes any data in the **mon_memory_system** table that has existed for more than 30 days.

```
INSERT INTO ph_task
(
tk_name,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete
)
VALUES
("mon_memory_system",
"MEMORY",
"Server memory consumption",
"mon_memory_system",
"create table mon_memory_system (ID integer, class smallint, size int8,
     used int8, free int8 )",
"insert into mon_memory_system select $DATA_SEQ_ID, seg_class, seg_size,
     seg_blkused, seg_blkfree FROM sysmaster:sysseglst",
NULL,
NULL,
INTERVAL ( 30 ) MINUTE TO MINUTE,
INTERVAL ( 30 ) DAY TO DAY
);
```

Related concepts:
Actions for task and sensors
Related tasks:
Modifying the scheduler

Actions for task and sensors

The action for a task or sensor is an SQL statement or routine that performs one or more operations.

SQL statements are useful if the action consists of a single operation. A stored procedure or a user-defined routine written in C or Java™ is useful if the action consists of multiple operations. The action is stored in the **tk_execute** column of the **ph_task** table.

You have a great deal of flexibility when you create an action. Possible types of operations include:

- Perform a DML operation. You can use a sensor to insert or update data in a table. You can use a task to delete older data from a table.
- Perform an administrative operation. You can use a task to run an SQL administration API function to administer the database server. For example, you can create a task to take checkpoints every two minutes.
- Perform an operation based on a threshold. You can use a threshold from the **ph_threshold** table to determine if a task action must be run. For example, you can create a task that adds a shared memory segment if the amount of available shared memory falls below a threshold value.
- Create an alert to report an operation or warn of a potential problem. For example, you can create a task to terminate idle users that inserts a row into the **ph_alert** table when a user session is terminated. You can also create a task to monitor backups and insert a warning into the **ph_alert** table when a backup has not occurred.

Use the following variables in your task or sensor action:

- $DATA_TASK_ID: Use to indicate the current task or sensor. This variable corresponds to the value of the **tk_id** field in the **ph_task** table.
- $DATA_SEQ_ID: Use to indicate the current execution of the task or sensor. This variable corresponds to the value of the **tk_sequence** field in the **ph_task** table and the **run_task_sequence** field in the **ph_run** table.

Examples

The following action is an SQL statement that the built-in **mon_command_history** task uses to remove older rows from the **command_history** table.

```
DELETE FROM command_history
WHERE cmd_exec_time < (
SELECT CURRENT - value::INTERVAL DAY to SECOND
FROM ph_threshold
WHERE name = 'COMMAND HISTORY RETENTION' )
```

The following example is an SQL statement that the built-in **mon_vps** sensor uses to add data to the **mon_vps** result table:

```
INSERT INTO mon_vps
SELECT $DATA_SEQ_ID, vpid, num_ready,
class, usecs_user, usecs_sys
FROM sysmaster:sysvplst
```

The following example is a stored procedure that terminates user sessions that are idle for longer than a value set by a threshold, and then adds an alert to the **ph_alert** table.

```
/*
 ************************************************************
 *  Create a function that will find all users that have
 *   been idle for the specified time. Call the SQL admin API to
 *   terminate those users.  Create an alert to track which
 *   users have been terminated.
 ************************************************************
 */
CREATE FUNCTION idle_timeout( task_id INT, task_seq INT)
RETURNING INTEGER

DEFINE time_allowed INTEGER;
DEFINE sys_hostname CHAR(16);
DEFINE sys_username CHAR(257);
DEFINE sys_sid      INTEGER;
DEFINE rc           INTEGER;

    {*** Get the maximum amount of time to be idle ***}
    SELECT value::integer
          INTO time_allowed
          FROM ph_threshold
          WHERE name = "IDLE TIMEOUT";

    {*** Find all users who are idle longer than the threshold ***}
    FOREACH SELECT admin("onmode","z",A.sid), A.username, A.sid, hostname
           INTO rc, sys_username, sys_sid, sys_hostname
           FROM sysmaster:sysrstcb A , sysmaster:systcblst B,
                sysmaster:sysscblst C
           WHERE A.tid = B.tid
           AND C.sid = A.sid
           AND lower(name) in  ("sqlexec")
           AND CURRENT - DBINFO("utc_to_datetime",last_run_time) > time_allowed UNITS MINUTE
           AND lower(A.username) NOT IN( "informix", "root")

           {*** If a user is successfully terminated, log ***}
           {*** the information into the alert table.    ***}
           IF rc > 0 THEN
               INSERT INTO ph_alert
                  (
                   ID, alert_task_id,alert_task_seq,
                   alert_type, alert_color,
                   alert_state,
                   alert_object_type, alert_object_name,
                   alert_message,
                   alert_action
                  ) VALUES (
                   0,task_id, task_seq,
                   "INFO", "GREEN",
                   "ADDRESSED",
                   "USER","TIMEOUT",
                   "User "||TRIM(sys_username)||"@"||TRIM(sys_hostname)||
                            " sid("||sys_sid||")"||
                            " terminated due to idle timeout.",
                   NULL
                  );
          END IF

   END FOREACH;

   RETURN 0;

END FUNCTION;
```

Related tasks:
Creating a task
Creating a sensor
Creating a threshold
Creating an alert

Creating a group

You can create a group to organize Scheduler tasks and sensors.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
When you create a task or sensor, you can specify a group name from the **ph_group** table in the **tk_group** column of the **ph_task** table.

To create a group:
Use an INSERT statement to add a row into the **ph_group** table in the **sysadmin** database. You must include a name for the group for the **group_name** column and a description of the group for the **group_description** column. The database server generates an ID for the group in the **group_id** column.

Example

The following example adds a group named TABLES:

```
INSERT INTO ph_group
(
group_name,
group_description
)
VALUES
(
"TABLES",
"Tasks that trim history and results tables."
);
```


Creating a threshold

You can create a threshold to determine under what conditions a Scheduler task or sensor is run.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
A threshold specifies a value that can be compared to a current value to determine whether a task or sensor must be run.

To create a threshold:

1. Use an INSERT statement to add values for the following columns in the **ph_threshold** table:
   - **name**: the name of the threshold
   - **task_name**: the name of the task from the **ph_task** table
   - **value**: the value of the threshold
   - **value_type**: the data type of the threshold (STRING or NUMERIC)
   - **description**: a description of what the threshold does
2. Write the task or sensor action to use the threshold.

Example

The following example adds a threshold named IDLE TIMEOUT for a task named Idle_timeout:

```
INSERT INTO ph_threshold
(
name,
task_name,
value,
value_type,
description
)
VALUES
(
"IDLE TIMEOUT",
"Idle_timeout",
"60",
"NUMERIC",
"Maximum amount of time in minutes for non-informix users to be idle."
);
```

The task action subtracts the time of the last user action from the current time and compare that value with the value column in the **ph_threshold** table.
Related concepts:
Actions for task and sensors

Creating an alert

You can create an alert as part of the action of a Scheduler task or sensor.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
To create an alert:
Use an INSERT statement to add a row to the **ph_alert** table. Include values for the following columns:

- **ID**: System generated; use 0 for the value.
- **alert_task_id**: Must reference the job ID from the **ph_task** table.
- **alert_task_seq**: Must reference the job sequence number from the **ph_task** table.
- **alert_type**: Choose INFO, WARNING, or ERROR.
- **alert_color**: Choose GREEN, YELLOW, or RED.
- **alert_state**: Choose NEW, IGNORED, ACKNOWLEDGED, ADDRESSED.
- **alert_object_type**: The type of object the alert describes, for example, SERVER.
- **alert_object_name**: The name of the object.
- **alert_message**: The message describing the alert.
- **alert_action**: An SQL statement or function that performs a corrective action, or NULL.

Example

The following example adds an alert to warn that a backup has not been taken. This code snippet is part of a stored procedure that takes task_id and task_seq as its arguments.

```
INSERT INTO ph_alert
(
ID,
alert_task_id,
alert_task_seq,
alert_type,
alert_color,
alert_state,
alert_object_type,
alert_object_name,
alert_message,
alert_action
)
VALUES
(
0,
task_id,
task_seq,
"WARNING",
"RED",
"NEW",
"SERVER",
"dbspace_name",
"Dbspace ["||trim(dbspace_name)|| "] has never had a level-0 backup.
    Recommend taking a level-0 backup immediately.",
NULL
);
```

Related concepts:
Actions for task and sensors

Monitor the scheduler

You can monitor Scheduler threads that are in progress with the **onstat -g dbc** command. You can view information about tasks and sensors that have completed in the **ph_run** table.

The Scheduler uses two types of threads while it is running:

- **dbWorker:** These threads are running scheduled tasks and sensors.
- **dbScheduler:** This thread prepares the next task or sensor that is scheduled to be run.

To view information about currently running tasks and sensor, and the next task or sensor that is run, use the **onstat -g dbc** command.

To view information about tasks and sensor that have completed, query the **ph_run** table in the **sysadmin** database. You must be connected to the **sysadmin** database as user **informix** or another authorized user.

Examples

The following output from the **onstat -g dbc** command shows two **dbWorker** threads and the **dbScheduler** thread:

```
Worker Thread(0)    46fa6f10
====================================
Task:                47430c18
Task Name:           mon_config_startup
Task ID:             3
Task Type:           STARTUP SENSOR
Last Error
    Number               -310
    Message              Table (informix.mon_onconfig)
                           already exists in database.
    Time                 09/11/2007 11:41
    Task Name            mon_config_startup

Task Execution:      onconfig_save_diffs

WORKER PROFILE
    Total Jobs Executed           10
    Sensors Executed               8
    Tasks Executed                 2
    Purge Requests                 8
    Rows Purged                    0

Worker Thread(1)    46fa6f80
====================================
Task:                4729fc18
Task Name:           mon_sysenv
Task ID:             4
Task Type:           STARTUP SENSOR
Task Execution:      insert into mon_sysenv select 1, env_name,
                       env_value FROM sysmaster:sysenv


WORKER PROFILE
```

```
    Total Jobs Executed             3
    Sensors Executed                2
    Tasks Executed                  1
    Purge Requests                  2
    Rows Purged                     0


Scheduler Thread    46fa6f80
=====================================
Run Queue
    Empty
Run Queue Size         0
Next Task              7
Next Task Waittime     57
```

The following output shows the history of four Scheduler jobs from the **ph_run** table:

```
SELECT * FROM ph_run;

RUN_ID           1
RUN_TASK_ID      2
RUN_TASK_SEQ     1
RUN_RETCODE      0
RUN_TIME         2009-07-20 13:04:59
RUN_DURATION     0.131850300007433
RUN_ZTIME        1248109468
RUN_BTIME        1248109468
RUN_MTIME        1248109499

RUN_ID           2
RUN_TASK_ID      3
RUN_TASK_SEQ     1
RUN_RETCODE      0
RUN_TIME         2009-07-20 13:04:59
RUN_DURATION     0.120845244247991
RUN_ZTIME        1248109468
RUN_BTIME        1248109468
RUN_MTIME        1248109499

RUN_ID           3
RUN_TASK_ID      4
RUN_TASK_SEQ     1
RUN_RETCODE      0
RUN_TIME         2009-07-20 13:04:59
RUN_DURATION     0.00254887164461759
RUN_ZTIME        1248109468
RUN_BTIME        1248109468
RUN_MTIME        1248109499

RUN_ID           2087
RUN_TASK_ID      7
RUN_TASK_SEQ     742
RUN_RETCODE      0
RUN_TIME         2009-09-09 11:09:51
RUN_DURATION     0.00489335523104662
RUN_ZTIME        1248109468
RUN_BTIME        1248109468
RUN_MTIME        1252508991
```


Modifying the scheduler

You can modify the properties of Scheduler tasks, sensors, alerts, thresholds, or groups. You can modify both built-in properties and properties that you added.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
To modify Scheduler properties:
Use an UPDATE statement for the appropriate Scheduler table in the **sysadmin** database.

Examples

The following example stops a task named `task1` from running:

```
UPDATE ph_task
  SET tk_enable = "F"
    WHERE tk_name = "task1";
```

The following example changes the amount of time that data collected by the built-in sensor `mon_profile` to 99 days:

```
UPDATE ph_task
SET tk_delete = "INTERVAL ( 99 ) DAY TO DAY"
WHERE tk_name = "mon_profile";
```

The following example changes the threshold named COMMAND HISTORY RETENTION to 20 so that the **command_history** table retains information about SQL administration API commands for 20 days:

```
UPDATE ph_threshold SET value = "20 0:00:00"
WHERE name = "COMMAND HISTORY RETENTION";
```

Related tasks:
Creating a task
Creating a sensor
Related reference:
Built-in tasks and sensors

Remote administration with the SQL administration API

You can use the SQL administration API to perform remote administration tasks by using SQL statements.

The SQL administration API functions take one or more arguments that define the task. Many of the tasks are ones that you can also complete with command-line utilities. The advantage of using the SQL administration API functions is that you can run them remotely from other database servers. You must be directly connected to the database server when you run command-line utility commands.

You can perform the following types of administrative tasks with the SQL administration API:

- Control data compression
- Update configuration parameters
- Check data, partitions, and extents consistency, control the B-tree scanner, and force a checkpoint
- Set up and administer Enterprise Replication
- Set up and administer high-availability clusters
- Control logging and logical logs
- Control shared-memory and add buffer pools
- Control mirroring
- Control decision-support queries
- Change the server mode
- Add, drop, and configure storage spaces
- Control the SQL statement cache
- Control and configure SQL tracing
- Start and stop the listen control threads dynamically
- Perform other tasks, such as moving the **sysadmin** database, terminating a session, or adding a virtual processor

For more information about the SQL administration API, see the *IBM® Informix® Administrator's Reference*.

- SQL administration API admin() and task() functions
  The SQL administration API consists of two functions: **admin()** and **task()** that are defined in the **sysadmin** database.
- Viewing SQL administration API history
  You can view the history of all the SQL administration API functions that the were run in the previous 30 days in the **command_history** table in the **sysadmin** database.

Related concepts:
Overview of automatic monitoring and corrective actions
Related information:
SQL Administration API Overview

SQL administration API admin() and task() functions

The SQL administration API consists of two functions: **admin()** and **task()** that are defined in the **sysadmin** database.

These functions perform the same tasks, but return results in different formats. The **task()** function returns a string that describes the results of the command. The **admin()** function returns an integer.

By default, only user **informix**, can connect to the **sysadmin** database. If user **root** or a member of the DBSA group is granted privileges to connect to the **sysadmin** database, user root or a member of the DBSA group can also run the SQL administration API **task()** and **admin()** functions.

Run the **task()** or **admin()** function in a transaction that does not include any other statements.

You can use EXECUTE FUNCTION statement to execute the **admin()** and **task()** functions. For example, the following SQL statement, which is equivalent to the **oncheck -ce** command, instructs the database server to check the extents:

```
BEGIN WORK;
EXECUTE FUNCTION admin("check extents");
END WORK;
```

You use SQL administration API functions in your Scheduler task actions. For example, you can define a task that creates a dbspace by using the following statement in the task action:

```
EXECUTE FUNCTION admin("create dbspace","dbspace2","/work/dbspace2","20 MB");
```

Related information:
SQL Administration API Overview

Viewing SQL administration API history

You can view the history of all the SQL administration API functions that the were run in the previous 30 days in the **command_history** table in the **sysadmin** database.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
The **command_history** table shows if an administrative task was executed through an **admin()** or **task()** function and displays information about the user who ran the command, the time the command was run, the command, and the message returned when the database server completed running the command.

To display command history:
Use a SELECT statement to return the data from the **command_history** table.
The following example displays all command history for the past 30 days:

```
SELECT * FROM command_history;
```

The following table shows sample commands and the associated results in a sample **command_history** table. For a description of all information in the **command_history** table, see the *IBM® Informix® Administrator's Reference*.

Table 1. Example of some information in a command_history table

Command	Sample returned messages
set sql tracing on	`SQL tracing on with 1000 buffers of 2024 bytes.`
create dbspace	`Space 'space12' added.`
checkpoint	`Checkpoint completed.`
add log	`Added 3 logical logs to dbspace logdbs.`

- Controlling the size of the command_history table
  You can reduce the retention period or remove rows from the **command_history** table to prevent it from becoming too large.

Related concepts:
Overview of automatic monitoring and corrective actions
Related information:
The command_history table

Controlling the size of the command_history table

You can reduce the retention period or remove rows from the **command_history** table to prevent it from becoming too large.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
By default, rows in the **command_history** table are automatically removed after a 30 days. The retention period is controlled by the COMMAND HISTORY RETENTION row in the **ph_threshold** table.

To reduce the retention period:
Use an UPDATE statement to modify the value of the COMMAND HISTORY RETENTION row in the **ph_threshold** table.
The following example sets the retention period to 25 days:

```
UPDATE ph_threshold
SET value = "25"
WHERE name = "COMMAND HISTORY RETENTION";
```

You can use SQL commands like DELETE or TRUNCATE TABLE to manually remove data from this table. You can also create a task in the **ph_task** table to purge data from the **command_history** table.

The following example shows a task that monitors the amount of data in the **command_history** table and purges data when it becomes too old.

```
INSERT INTO ph_task
( tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency )
VALUES
("mon_command_history",
"TASK",
"TABLES",
"Monitor how much data is kept in the command history table",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY RETENTION' ) ",
DATETIME(02:00:00) HOUR TO SECOND,
NULL,
INTERVAL ( 1 ) DAY TO DAY);
```

Related information:
The command_history table

Query drill-down

You can use query drill-down, or SQL tracing, to gather statistical information about each SQL statement that was run and to analyze statement history.

SQL tracing helps you answer questions such as:

- How long do SQL statements take?
- How many resources are individual statements using?
- How long did statement execution take?
- How much time was involved waiting for each resource?

The statistical information is stored in a circular buffer, which is an in-memory pseudo table, called **syssqltrace**, that is stored in the **sysmaster** database. You can dynamically resize the circular buffer.

By default SQL tracing turned off, but you can turn it on for all users or for a specific set of users. When SQL tracing is enabled with its default configuration, the database server tracks the last 1000 SQL statements that ran, along with the profile statistics for those statements. You can also disable SQL tracing globally or for a particular user.

The memory required by SQL tracing is large if you plan to keep much historical information. The default amount of space required for SQL tracing is two megabytes. You can expand or reduce the amount of storage according to your requirements.

Information displayed includes:

- The user ID of the user who ran the command
- The database session ID
- The name of the database
- The type of SQL statement
- The duration of the SQL statement execution
- The time this statement completed
- The text of the SQL statement or a function call list (also called *stack trace*) with the statement type, for example:

  ```
  procedure1() calls procedure2() calls procedure3()
  ```

- Statistics including the:
  - Number of buffer reads and writes
  - Number of page reads and writes
  - Number of sorts and disk sorts
  - Number of lock requests and waits
  - Number of logical log records
  - Number of index buffer reads
  - Estimated number of rows
  - Optimizer estimated cost
  - Number of rows returned
- Database isolation level.

You can also specify escalating levels of information to include in the tracing, as follows:

- low-level tracing, which is enabled by default, captures the information shown in the example below. This information includes statement statistics, statement text, and statement iterators.
- Medium level tracing captures all of the information included in low-level tracing, plus the list of table names, database name and stored procedure stacks.
- high-level tracing captures all of the information included in medium-level tracing, plus host variables.

The amount of information traced affects the amount of memory required for this historical data.

You can enable and disable the tracing at any point in time, and you can change the number and size of the trace buffers while the database server is running. If you resize the trace buffer, the database server attempts to maintain the content of the buffer. If the parameters are increased, data is not truncated. However, if the number or the size of the buffers are reduced, the data in the trace buffers might be truncated or lost.

The number of buffers determines how many SQL statements are traced. Each buffer contains the information for a single SQL statement. By default, an individual trace buffer is a fixed size. If the text information stored in the buffer exceeds the size of the trace buffer, then the data is truncated.

The following example shows SQL tracing information:

```
select * from syssqltrace where sql_id = 5678;

sql_id             5678
sql_address        4489052648
sql_sid            55
sql_uid            2053
sql_stmttype       6
sql_stmtname       INSERT
sql_finishtime     1140477805
sql_begintxtime    1140477774
sql_runtime        30.86596333400
sql_pgreads        1285
sql_bfreads        19444
sql_rdcache        93.39127751491
sql_bfidxreads     5359
sql_pgwrites       810
sql_bfwrites       17046
sql_wrcache        95.24815205913
sql_lockreq        10603
sql_lockwaits      0
sql_lockwttime     0.00
sql_logspace       60400
sql_sorttotal      0
sql_sortdisk       0
sql_sortmem        0
sql_executions     1
sql_totaltime      30.86596333400
sql_avgtime        30.86596333400
sql_maxtime        30.86596333400
```

```
sql_numiowaits     2080
sql_avgiowaits     0.014054286131
sql_totaliowaits   29.23291515300
sql_rowspersec     169.8958799132
sql_estcost        102
sql_estrows        1376
sql_actualrows     5244
sql_sqlerror       0
sql_isamerror      0
sql_isollevel      2
sql_sqlmemory      32608
sql_numiterators   4
sql_database       db3
sql_numtables      3
sql_tablelist      t1
sql_statement      insert into t1 select {+ AVOID_FULL(sysindices) } 0, tabname
```

For an explanation of all table rows, see information about the **syssqltrace** table in the **sysmaster** database section of the *IBM® Informix® Administrator's Reference*.

- Specifying startup SQL tracing information by using the SQLTRACE configuration parameter
  Use the SQLTRACE configuration parameter to control the default tracing behavior when the database server starts. By default, this parameter is not set. The information you set includes the number of SQL statements to trace and the tracing mode.
- Disable SQL tracing globally or for a session
  Even if the mode specified in the SQLTRACE configuration parameter is `global` or `user`, you can disable SQL tracing if you want to completely turn off all user and global tracing and deallocate resources currently in use by SQL tracing. By default, SQL tracing is off for all users.
- Enable SQL tracing
  After you specify `user` as the mode in the SQLTRACE configuration parameter, you must run the SQL administration API **task()** or **admin()** function to turn SQL history tracing on for a particular user.
- Enable global SQL tracing for a session
  You can enable global SQL tracing for the current session by running the SQL administration API **task()** or **admin()** function.

Related concepts:
Overview of automatic monitoring and corrective actions

Specifying startup SQL tracing information by using the SQLTRACE configuration parameter

Use the SQLTRACE configuration parameter to control the default tracing behavior when the database server starts. By default, this parameter is not set. The information you set includes the number of SQL statements to trace and the tracing mode.

Any user who can modify the onconfig file can modify the value of the SQLTRACE configuration parameter and effect the startup configuration. However, only user **informix**, **root**, or a DBSA who has been granted connect privileges to the **sysadmin** database can use SQL administration API commands to modify the runtime status of the SQL tracing.

To specify SQL tracing information when the database server starts:

1. Set the SQLTRACE configuration parameter in the onconfig file.
2. Restart the database server.

Example

The following setting in the onconfig file specifies that the database server gathers low-level information about up to 2000 SQL statements executed by all users on the system and allocates approximately four megabytes of memory (2000 * two KB).

```
SQLTRACE level=LOW,ntraces=2000,size=2,mode=global
```

If you use only a percentage of the allocated buffer space (for example, 42 percent of the buffer space), the amount of memory that is allocated is still two KB.

If you do not want to set the SQLTRACE configuration parameter and restart the server, you can run the following SQL administration API command, which provides the same function as setting SQLTRACE for the current session:

```
EXECUTE FUNCTION task("set sql tracing on", 100,"1k","med","user");
```

After enabling the SQL tracing system in user mode, you can then enable tracing for each user. See Enable SQL tracing.

For more information about using **task()** and **admin()** functions, see the *IBM® Informix® Administrator's Reference*.

For more information about the SQLTRACE configuration parameter, including minimum and maximum values for some fields, see the *IBM Informix Administrator's Reference*.

Disable SQL tracing globally or for a session

Even if the mode specified in the SQLTRACE configuration parameter is `global` or `user`, you can disable SQL tracing if you want to completely turn off all user and global tracing and deallocate resources currently in use by SQL tracing. By default, SQL tracing is off for all users.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To disable global SQL tracing, run the SQL administration API **task()** or **admin()** function with the set sql tracing off argument.

To disable SQL tracing for a particular session, run the SQL administration API **task()** or **admin()** function with set sql tracing off as the first argument and the session identification number as the second argument.

Examples

The following example disables SQL tracing globally:

```
EXECUTE FUNCTION task('set sql tracing off');
(expression) SQL tracing off.

1 row(s) retrieved.
```

The following example disables SQL tracing for the session with an ID of 47:

```
EXECUTE FUNCTION task("set sql user tracing off",47);
```

For more information about using **task()** or **admin()** functions, see the *IBM® Informix® Guide to SQL: Syntax*.

Enable SQL tracing

After you specify `user` as the mode in the SQLTRACE configuration parameter, you must run the SQL administration API **task()** or **admin()** function to turn SQL history tracing on for a particular user.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

Global SQL tracing is not required to be enabled to allow SQL tracing for a particular user.

To enable SQL tracing for a particular user, run the SQL administration API **task()** or **admin()** function with set sql tracing on as the first argument and the user session ID as the second argument.

To enable user SQL tracing for all users except **root** or **informix**, you can run a **task()** or **admin()** function with the set sql tracing on argument and information that defines the users.

Example

The following example enables SQL tracing for the user with the session ID of 74:

```
EXECUTE FUNCTION task("set sql user tracing on", 74);
```

The following example enables the tracing of SQL statements of users who are currently connected to the system as long as they are not logged in as user **root** or **informix**.

```
dbaccess sysadmin -<<END
 execute function task("set sql tracing on", 1000, 1,"low","user");
 select task("set sql user tracing on", sid)
        FROM sysmaster:syssessions
  WHERE username not in ("root","informix");
END
```

For more information about the **task()** and **admin()** functions, see the *IBM® Informix® Administrator's Reference*.

Enable global SQL tracing for a session

You can enable global SQL tracing for the current session by running the SQL administration API **task()** or **admin()** function.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

By default, global SQL tracing is not enabled. You can set the SQLTRACE configuration parameter to permanently enable global tracing.

To enable global user SQL history tracing for the current database server session, run the SQL administration API **task()** or **admin()** function with the set sql tracing on argument.

Example

The following example enables global low-level SQL tracing for all users:

```
EXECUTE FUNCTION task("set sql tracing on", 1000, 1,"low","global");
```

If a new user logs on to the system after your statement runs, you can enable tracing for the new user. See Enable SQL tracing.

For more information about the **task()** and **admin()** functions, see the *IBM® Informix® Guide to SQL: Syntax*.

Administrator's Reference

These topics include comprehensive descriptions of IBM® Informix® configuration parameters, the system-monitoring interface (SMI) tables in the sysmaster database, the syntax of database server utilities such as **onmode** and **onstat**, logical-log records, disk structures, event alarms, and unnumbered error messages.

These topics are of interest to the following users:

- Database administrators
- System administrators
- Performance engineers

These topics are written with the assumption that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with database server administration, operating-system administration, or network administration

These topics are from the *IBM Informix Administrator's Reference*.

Configuring and monitoring Informix

Database configuration parameters

The Informix® database server uses a configuration file, which is called the onconfig file, during initialization. This file contains default configuration parameter values. You can modify the parameter values to improve performance and other characteristics of the instance or database.

The **ONCONFIG** environment variable identifies your onconfig file.

- AUTO_LLOG configuration parameter
  Use the AUTO_LLOG configuration parameter to automatically add logical logs in the specified dbspace to improve performance.
- AUTO_TUNE_SERVER_SIZE configuration parameter
  Use the AUTO_TUNE_SERVER_SIZE configuration parameter to set the sizes of memory and storage spaces to allocate based on the number of expected concurrent users.
- AUTO_LRU_TUNING configuration parameter
  Use the AUTO_LRU_TUNING configuration parameter to enable automatic LRU tuning, which automatically maintains enough clean pages for page replacement.
- AUTO_READAHEAD configuration parameter
  Use the AUTO_READAHEAD configuration parameter to change the automatic read-ahead mode or to disable automatic read-ahead operations for a query.
- AUTO_REPREPARE configuration parameter
  The AUTO_REPREPARE configuration parameter controls whether the database server automatically reoptimizes SPL routines and reprepares prepared objects after the schema of a table that is referenced by the SPL routine or by the prepared object was changed.
- AUTO_STAT_MODE configuration parameter
  Use the AUTO_STAT_MODE configuration parameter to enable or disable the mode for selectively updating only stale or missing data distributions in UPDATE STATISTICS operations instead of updating statistics for all data distributions.
- AUTO_TUNE configuration parameter
  Use the AUTO_TUNE configuration parameter to enable or disable all automatic tuning configuration parameters that have values that are not present in the onconfig file.
- AUTOLOCATE configuration parameter
  Use the AUTOLOCATE configuration parameter to enable the automatic location of databases, indexes, and tables, and the automatic fragmentation of tables.
- BATCHEDREAD_INDEX configuration parameter
  Use the BATCHEDREAD_INDEX configuration parameter to enable the optimizer to execute light scans for indexes. This reduces the number of times that a buffer is read, thus improving performance.
- BATCHEDREAD_TABLE configuration parameter
  Use the BATCHEDREAD_TABLE configuration parameter to enable or disable light scans on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data.
- BLOCKTIMEOUT configuration parameter
  Use the BLOCKTIMEOUT configuration parameter to specify the number of seconds that a thread or database server will hang. After the timeout, the thread or database server will either continue processing or fail.
- BTSCANNER Configuration Parameter
  Use the BTSCANNER configuration parameter to set the B-tree scanner. The B-tree scanner improves transaction processing for logged databases when rows are deleted from a table with indexes. The B-tree scanner threads remove deleted index entries and rebalance the index nodes. The B-tree scanner automatically determines which index items are to be deleted.
- BUFFERPOOL configuration parameter
  Use the BUFFERPOOL configuration parameter to configure how many data pages are cached in shared memory and how often those pages are flushed to disk between checkpoints. The default values of the BUFFERPOOL configuration parameter are adequate for many systems. However, you can change the values to tune the performance of your system.
- CHECKALLDOMAINSFORUSER configuration parameter
  Use the CHECKALLDOMAINSFORUSER configuration parameter to check all of the domains for all users.
- CKPTINTVL configuration parameter
  Use the CKPTINTVL configuration parameter to specify the frequency, expressed in seconds, at which the database server checks to determine whether a checkpoint is needed. When a checkpoint occurs, all pages in the shared-memory buffer pool are written to disk.
- CLEANERS configuration parameter
  Use the CLEANERS configuration parameter to specify the number of page-cleaner threads available during the database server operation. By default, the database server always runs one page-cleaner thread. A general guideline is one page cleaner per disk drive. The value specified has no effect on the size of shared memory.
- CLUSTER_TXN_SCOPE configuration parameter
  Set the CLUSTER_TXN_SCOPE configuration parameter to configure your high-availability cluster so that when a client session issues a commit, the server blocks the session until the transaction is applied in that session, on a secondary server, or across the cluster.
- CONSOLE configuration parameter
  Use the CONSOLE configuration parameter to specify the path and name for console-message file.
- CONVERSION_GUARD configuration parameter
  Use the CONVERSION_GUARD configuration parameter to specify whether IBM® Informix stops or continues an upgrade to a new version of the server if an error occurs during the upgrade process.
- DATASKIP Configuration Parameter
  Use the DATASKIP configuration parameter to control whether the database server skips a dbspace that is unavailable during the processing of a transaction.
- DBCREATE_PERMISSION configuration parameter
  Use the DBCREATE_PERMISSION configuration parameter to restrict the permission to create databases to the user that you specify.
- DB_LIBRARY_PATH configuration parameter
  Use the DB_LIBRARY_PATH configuration parameter to specify a comma-separated list of valid directory prefix locations from which the database server can load external modules, such as DataBlade modules. You can also include server environment variables, such as $INFORMIXDIR, in the list.
- DBSERVERALIASES configuration parameter
  Use the DBSERVERALIASES configuration parameter to specify an alias name, or a list of unique alias names for the database server. Each alias defined by the DBSERVERALIASES configuration parameter can be used in a different connection, as specified by entries in the sqlhosts information.
- DBSERVERNAME configuration parameter
  Use the DBSERVERNAME configuration parameter to specify a unique name that you want to associate with the database server. You specify this configuration parameter when you install the database server.
- DBSPACETEMP configuration parameter
  Use the DBSPACETEMP configuration parameter to specify a list of dbspaces that the database server uses to globally manage the storage of temporary tables.
- DD_HASHMAX configuration parameter
  Use the DD_HASHMAX configuration parameter to specify the maximum number of tables in each hash bucket in the data-dictionary cache.
- DD_HASHSIZE configuration parameter
  Use the DD_HASHSIZE configuration parameter to specify the number of hash buckets or lists that are in the data-dictionary cache.
- DEADLOCK_TIMEOUT configuration parameter
  Use the DEADLOCK_TIMEOUT configuration parameter to specify the maximum number of seconds that a database server thread can wait to acquire a lock.
- DEF_TABLE_LOCKMODE configuration parameter
  Use the DEF_TABLE_LOCKMODE configuration parameter to specify the lock mode at the page or row level for new tables.
- DEFAULTESCCHAR configuration parameter
  The DEFAULTESCCHAR configuration parameter specifies the default escape character that is used for LIKE and MATCHES conditions.
- DELAY_APPLY Configuration Parameter
  Use the DELAY_APPLY configuration parameter to configure RS secondary servers to wait for a specified period of time before applying logs.

- DIRECT_IO configuration parameter (UNIX)
  Use the DIRECT_IO configuration parameter to control the use of direct I/O for cooked files used for dbspace chunks.
- DIRECTIVES configuration parameter
  Use the DIRECTIVES configuration parameter to enable or disable the use of optimizer directives. These directives specify behavior for the query optimizer in developing query plans for SELECT, UPDATE, and DELETE statements.
- DISABLE_B162428_XA_FIX configuration parameter
  Use the DISABLE_B162428_XA_FIX configuration parameter to specify when transactions are freed.
- DISK_ENCRYPTION configuration parameter
  The DISK_ENCRYPTION configuration parameter controls the encryption of storage spaces.
- DRDA_COMMBUFFSIZE configuration parameter
  Use the DRDA_COMMBUFFSIZE configuration parameter to specify the size of the DRDA communications buffer.
- DRAUTO configuration parameter
  Set the DRAUTO configuration parameter to specify a HDR-failover method for HDR high-availability systems.
- DRIDXAUTO configuration parameter
  Use the DRIDXAUTO configuration parameter to specify whether the primary High-Availability Data Replication (HDR) server automatically starts index replication if the secondary HDR server detects a corrupted index.
- DRINTERVAL configuration parameter
  Use the DRINTERVAL configuration parameter to specify the maximum number of seconds between flushes of the data-replication buffer, whether to use HDR SYNC mode, or whether to use the synchronization mode that is specified by the HDR_TXN_SCOPE configuration parameter.
- DRLOSTFOUND configuration parameter
  Use the DRLOSTFOUND configuration parameter to specify the path name to the HDR lost-and-found file. This file indicates that some transactions were committed on the HDR primary database server before that were not committed on the secondary database server when the primary database server experienced a failure.
- DRTIMEOUT configuration parameter
  Use the DRTIMEOUT configuration parameter to specify the length of time, in seconds, that a database server in a high-availability data-replication pair waits for a transfer acknowledgment from the other database server in the pair. This parameter applies only to high-availability data-replication pairs.
- DS_HASHSIZE configuration parameter
  Use the DS_HASHSIZE configuration parameter to specify the number of hash buckets in the data-distribution cache and other caches. The database server stores and accesses column statistics that the UPDATE STATISTICS statement generates in the MEDIUM or HIGH mode in the data-distribution cache.
- DS_MAX_QUERIES configuration parameter
  Use the DS_MAX_QUERIES configuration parameter to specify the maximum number of parallel database queries (PDQ) that can run concurrently.
- DS_MAX_SCANS configuration parameter
  Use the DS_MAX_SCANS configuration parameter to limit the number of PDQ scan threads that the database server can execute concurrently.
- DS_NONPDQ_QUERY_MEM configuration parameter
  Use the DS_NONPDQ_QUERY_MEM configuration parameter to increase the amount of memory that is available for a query that is not a Parallel Database Query (PDQ). (You can only use this parameter if PDQ priority is set to zero.)
- DS_POOLSIZE configuration parameter
  Use the DS_POOLSIZE parameter to specify the maximum number of entries in the data-distribution cache and other caches. The database server stores and accesses column statistics that the UPDATE STATISTICS statement generates in the MEDIUM or HIGH mode in the data-distribution cache.
- DS_TOTAL_MEMORY configuration parameter
  Use the DS_TOTAL_MEMORY configuration parameter to specify the amount of memory available for PDQ queries. The amount should be smaller than the computer physical memory, minus fixed overhead such as operating-system size and buffer-pool size.
- DUMPCNT configuration parameter (UNIX)
  Use the DUMPCNT configuration parameter to specify the number of assertion failures in a thread for which a database server dumps shared memory or generates a core file by calling the **gcore** utility.
- DUMPCORE configuration parameter (UNIX)
  Use the DUMPCORE configuration parameter to control whether assertion failures cause a virtual processor to dump a core image. The core file is left in the directory from which the database server was last invoked. (The DUMPDIR parameter has no impact on the location of the core file.)
- DUMPDIR configuration parameter
  DUMPDIR specifies a directory in which the database server dumps shared memory, gcore files, or messages from a failed assertion.
- DUMPGCORE configuration parameter (UNIX)
  Use the DUMPGCORE configuration parameter to specify whether to dump the **gcore** core file. Use this configuration parameter with operating systems that support **gcore**.
- DUMPSHMEM configuration parameter (UNIX)
  Use the DUMPSHMEM configuration parameter to indicate whether a shared memory dump is created on an assertion failure. This configuration parameter also specifies how much memory is written to the shmem.*pid.cnt* file in the directory specified by the DUMPDIR configuration parameter.
- DYNAMIC_LOGS configuration parameter
  Use the DYNAMIC_LOGS configuration parameter to allow logical logs to be dynamically added when necessary to prevent transaction blocking.
- EILSEQ_COMPAT_MODE configuration parameter
  Use the EILSEQ_COMPAT_MODE configuration parameter to control if IBM Informix checks whether character data inserted by a client application contains code point sequences not recognized by the locale of the current database.
- ENABLE_SNAPSHOT_COPY configuration parameter
  Use the ENABLE_SNAPSHOT_COPY configuration parameter to enable or disable the ability to clone a server using the **ifxclone** utility.
- ENCRYPT_CIPHERS configuration parameter
  Use the ENCRYPT_CIPHERS configuration parameter to define all ciphers and modes that can be used by the current database session. ENCRYPT_CIPHERS is used for Enterprise Replication and High-Availability Data Replication only.
- ENCRYPT_HDR configuration parameter
  Use the ENCRYPT_HDR configuration parameter to enable or disable HDR encryption.
- ENCRYPT_MAC configuration parameter
  Use the ENCRYPT_MAC configuration parameter to control the level of message authentication code (MAC) generation. This configuration parameter is used only for Enterprise Replication and High-Availability Data Replication.
- ENCRYPT_MACFILE configuration parameter
  Use the ENCRYPT_MACFILE configuration parameter to specify a list of the full path names of MAC key files. This configuration parameter is used only for Enterprise Replication and High-Availability Data Replication.
- ENCRYPT_SMX configuration parameter
  Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high-availability configurations on secondary servers and between Enterprise Replication Servers.
- ENCRYPT_SWITCH configuration parameter
  Use the ENCRYPT_SWITCH configuration parameter to define the frequency at which ciphers or secret keys are renegotiated. This configuration parameter is used only for Enterprise Replication and High-Availability Data Replication.

- EXPLAIN_STAT configuration parameter
  Use the EXPLAIN_STAT configuration parameter to enable or disable the inclusion of a Query Statistics section in the explain output file.
- EXT_DIRECTIVES configuration parameter
  Use the EXT_DIRECTIVES configuration parameter to enable or disable the use of external query optimizer directives.
- EXTSHMADD configuration parameter
  Use the EXTSHMADD configuration parameter to specify the size of virtual-extension segments that are added when a user-defined routine or a DataBlade routine is run in a user-defined virtual processor.
- FAILOVER_CALLBACK configuration parameter
  Use the FAILOVER_CALLBACK configuration parameter to specify the script executed by the database server when a database server transitions from a secondary server to a primary or standard server.
- FAILOVER_TX_TIMEOUT configuration parameter
  In high-availability cluster environments, use the FAILOVER_TX_TIMEOUT configuration parameter to enable transactions to complete after failover of the primary server.
- FASTPOLL configuration parameter
  Use the FASTPOLL configuration parameter to enable or disable fast polling of your network. FASTPOLL is a platform-specific configuration parameter.
- FILLFACTOR configuration parameter
  Use the FILLFACTOR configuration parameter to specify the degree of index-page fullness. A low value provides room for growth in the index. A high value compacts the index.
- FULL_DISK_INIT configuration parameter
  Use the FULL_DISK_INIT configuration parameter to prevent an accidental disk reinitialization of an existing database server instance. This configuration parameter specifies whether or not the disk initialization command (**oninit -i**) can run on your IBM Informix instance when a page zero exists at the root path location, which is at the first page of the first chunk location.
- GSKIT_VERSION configuration parameter
  Use the GSKIT_VERSION configuration parameter to specify the major version of IBM Global Security Kit (GSKit) that the database server uses for encryption and SSL communication.
- HA_ALIAS configuration parameter
  The HA_ALIAS configuration parameter defines a network alias that is used for server-to-server communication in a high-availability cluster. The specified network alias is also used by Connection Managers, the **ifxclone** utility, and **onmode -d** commands.
- HA_FOC_ORDER configuration parameter
  Use the HA_FOC_ORDER configuration parameter to define a single connection-management failover rule for a high-availability cluster of servers.
- HDR_TXN_SCOPE configuration parameter
  The HDR_TXN_SCOPE configuration parameter is used with the DRINTERVAL configuration parameter to specify the synchronization mode for HDR replication in a high-availability cluster.
- HETERO_COMMIT configuration parameter
  Use the HETERO_COMMIT configuration parameter to control whether the database server participates in heterogeneous commit transactions.
- IFX_EXTEND_ROLE configuration parameter
  Your database system administrator (DBSA), by default user **informix**, can use the IFX_EXTEND_ROLE parameter to control which users are authorized to register DataBlade modules or external user-defined routines (UDRs).
- IFX_FOLDVIEW configuration parameter
  Use the IFX_FOLDVIEW configuration parameter to enable or disable view folding. For certain situations where a view is involved in a query, view folding can significantly improve the performance of the query. In these cases, views are folded into a parent query instead of the query results being put into a temporary table.
- IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter
  Use the IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter to enable the transaction manager to use the same XID to represent global transactions on different databases in the same database server instance.
- INFORMIXCONRETRY configuration parameter
  Use the INFORMIXCONRETRY configuration parameter to specify the maximum number of connection attempts that can be made to each database server after the initial connection attempt fails. These attempts are made within the time limit that the INFORMIXCONTIME configuration parameter specifies.
- INFORMIXCONTIME configuration parameter
  Use the INFORMIXCONTIME configuration parameter to specify the number of seconds that the CONNECT statement attempts to establish a connection to a database server.
- LIMITNUMSESSIONS configuration parameter
  Use the LIMITNUMSESSIONS configuration parameter to define the maximum number of sessions that you want connected to IBM Informix.
- LISTEN_TIMEOUT configuration parameter
  Use the LISTEN_TIMEOUT configuration parameter to specify the number of seconds in which the server waits for a connection.
- LOCKS configuration parameter
  The LOCKS configuration parameter specifies the initial size of the lock table.
- LOGBUFF configuration parameter
  Use the LOGBUFF configuration parameter to specify the size in kilobytes for the three logical-log buffers in shared memory.
- LOGBUF_INTVL configuration parameter
  Use the LOGBUF_INTVL configuration parameter to ensure the logical log buffer is flushed periodically when only buffered logging is used.
- LOGFILES configuration parameter
  Use the LOGFILES configuration parameter to specify the number of logical-log files that the database server creates during disk initialization.
- LOG_INDEX_BUILDS configuration parameter
  Use the LOG_INDEX_BUILDS configuration parameter to enable or disable index page logging.
- LOG_STAGING_DIR configuration parameter
  Use the LOG_STAGING_DIR configuration parameter to specify the location of log files received from the primary server when configuring delayed application of log files on RS secondary servers.
- LOGSIZE configuration parameter
  Use the LOGSIZE configuration parameter to specify the size that is used when logical-log files are created.
- LOW_MEMORY_MGR configuration parameter
  Use the LOW_MEMORY_MGR configuration parameter to enable automatic low memory management, which you can use to change the default behavior of a primary or standard server when it reaches its memory limit.
- LOW_MEMORY_RESERVE configuration parameter
  Use the LOW_MEMORY_RESERVE configuration parameter to reserve a specific amount of memory for use when critical activities are needed and the server has limited free memory.
- LTXEHWM configuration parameter
  Use the LTXEHWM configuration parameter to specify the *long-transaction, exclusive-access, high-watermark*. When the logical-log space reaches the LTXEHWM threshold, the long transaction currently being rolled back is given *exclusive* access to the logical log.

- LTXHWM configuration parameter
  Use the LTXHWM configuration parameter to specify the long-transaction high-watermark. The *long-transaction high-watermark* is the percentage of available log space that, when filled, triggers the database server to check for a long transaction.
- MAX_FILL_DATA_PAGES configuration parameter
  Use the MAX_FILL_DATA_PAGES configuration parameter to control inserting more rows to pages that have variable-length rows.
- MAX_INCOMPLETE_CONNECTIONS configuration parameter
  Use the MAX_INCOMPLETE_CONNECTIONS configuration parameter to specify the maximum number of incomplete connections in a session.
- MAX_PDQPRIORITY configuration parameter
  Use the MAX_PDQPRIORITY configuration parameter to limit the PDQ resources that the database server can allocate to any one DSS query.
- MIRROR configuration parameter
  Use the MIRROR configuration parameter to enable or disable mirroring for the database server.
- MIRROROFFSET configuration parameter
  In IBM Informix, MIRROROFFSET specifies the offset into the disk partition or into the device to reach the chunk that serves as the mirror for the initial chunk of the root dbspace.
- MIRRORPATH configuration parameter
  Use the MIRRORPATH configuration parameter to specify the full path name of the mirrored chunk for the initial chunk of the root dbspace.
- MSG_DATE configuration parameter
  Use the MSG_DATE configuration parameter to enable the insertion of a date in MM/DD/YY format at the beginning of each message printed to the online log.
- MSGPATH configuration parameter
  Use the MSGPATH configuration parameter to specify the full path name of the message-log file. The database server writes status messages and diagnostic messages to this file during operation.
- MULTIPROCESSOR configuration parameter
  Use the MULTIPROCESSOR configuration parameter to specify whether the database server performs locking in a manner that is suitable for a single-processor computer or a multiprocessor computer.
- NET_IO_TIMEOUT_ALARM configuration parameter
  Use the NET_IO_TIMEOUT_ALARM configuration parameter to control whether to be notified if network write operations have been blocked for 30 minutes or more.
- NETTYPE configuration parameter
  Use the NETTYPE parameter to tune the network protocols that are defined in the sqlhosts information.
- NS_CACHE configuration parameter
  Use the NS_CACHE configuration parameter to define the maximum retention time for entries in the Informix name service caches: the host name/IP address cache, the service cache, the user cache, and the group cache.
- NUMFDSERVERS configuration parameter
  For network connections on UNIX, use the NUMFDSERVERS configuration parameter to specify the maximum number of poll threads to handle network connections migrating between IBM Informix virtual processors (VPs).
- OFF_RECVRY_THREADS configuration parameter
  Use the OFF_RECVRY_THREADS configuration parameter to specify the number of recovery threads that are used for logical recovery during a cold restore or fast recovery.
- ON_RECVRY_THREADS configuration parameter
  The ON_RECVRY_THREADS configuration parameter is the maximum number of recovery threads that the database server uses for logical recovery when the database server is online (during a warm restore).
- ONDBSPACEDOWN configuration parameter
  Use the ONDBSPACEDOWN configuration parameter to define the action that the database server takes when any disabling event occurs on a primary chunk within a noncritical dbspace.
- ONLIDX_MAXMEM configuration parameter
  Use the ONLIDX_MAXMEM configuration parameter to limit the amount of memory that is allocated to a single *preimage* pool and a single *updator* log pool.
- OPTCOMPIND configuration parameter
  Use the OPTCOMPIND to specify information that helps the optimizer choose an appropriate query plan for your application.
- OPT_GOAL configuration parameter
  Use the OPT_GOAL configuration parameter to specify an optimization goal for queries.
- PC_HASHSIZE configuration parameter
  Use PC_HASHSIZE to specify the number of hash buckets in the caches that the database server uses. PC_HASHSIZE applies to UDR cache only.
- PC_POOLSIZE configuration parameter
  Use the PC_POOLSIZE configuration parameter to specify the maximum number of user-defined routines that are stored in the UDR cache.
- PFSC_BOOST configuration parameter
  Use the PFSC_BOOST configuration parameter to enable or disable the boosted partition free space cache feature.
- PHYSBUFF configuration parameter
  Use the PHYSBUFF configuration parameter to specify the size in kilobytes of the two physical-log buffers in shared memory.
- PHYSFILE configuration parameter
  Use the PHYSFILE configuration parameter to specify the size of the physical log file when you first initialize the disk space and bring the database server online.
- PLOG_OVERFLOW_PATH configuration parameter
  The PLOG_OVERFLOW_PATH parameter specifies the location of the file that is used during fast recovery if the physical log file overflows.
- PLCY_HASHSIZE configuration parameter
  The PLCY_HASHSIZE configuration parameter specifies the number of hash buckets in the security policy information cache.
- PLCY_POOLSIZE configuration parameter
  Use the PLCY_POOLSIZE configuration parameter to specify the maximum number of entries in each hash bucket of the security policy information cache.
- PN_STAGEBLOB_THRESHOLD configuration parameter
  Use the PN_STAGEBLOB_THRESHOLD configuration parameter to reserve space for BYTE and TEXT data in round-robin fragments.
- PRELOAD_DLL_FILE configuration parameter
  The PRELOAD_DLL_FILE configuration parameter specifies the path name for a shared library file that is preloaded when the database server is started.
- QSTATS configuration parameter
  The QSTATS configuration parameter specifies the ability of **onstat -g qst** to print queue statistics.
- REMOTE_SERVER_CFG configuration parameter
  Use the REMOTE_SERVER_CFG configuration parameter to specify the file that lists trusted remote hosts.
- REMOTE_USERS_CFG configuration parameter
  Use the REMOTE_USERS_CFG configuration parameter to specify the file that lists the names of trusted users that exist on remote hosts.
- RESIDENT configuration parameter
  Use the RESIDENT configuration parameter to specify whether resident and virtual segments of shared memory remain resident in operating-system physical memory.
- RESTARTABLE_RESTORE configuration parameter
  Use the RESTARTABLE_RESTORE configuration parameter to control whether the database server performs restartable restores.

- RESTORE_POINT_DIR configuration parameter
  Use the RESTORE_POINT_DIR configuration parameter to change the path name of the directory where restore point files will be placed during a failed upgrade to a new version of the server. IBM Informix will store restore point files in a subdirectory of the specified directory, with the server number as the subdirectory name, only if the CONVERSION_GUARD configuration parameter is enabled.
- ROOTNAME configuration parameter
  ROOTNAME specifies a name for the root dbspace for this database server configuration.
- ROOTOFFSET configuration parameter
  ROOTOFFSET specifies the offset into an allocation of disk space (file, disk partition, or device) at which the initial chunk of the root dbspace begins.
- ROOTPATH configuration parameter
  Use the ROOTPATH configuration parameter to specify the full path name, including the device or file name, of the initial chunk of the root dbspace. The ROOTPATH configuration parameter is stored in the reserved pages as a chunk name.
- ROOTSIZE configuration parameter
  Use the ROOTSIZE configuration parameter to specify the size in kilobytes of the initial chunk of the root dbspace. The size that you select depends on your immediate plans for your database server.
- RSS_FLOW_CONTROL configuration parameter
  Specifies when flow control occurs in a high-availability cluster that contains at least one remote standalone (RS) secondary server.
- RSS_NONBLOCKING_CKPT configuration parameter
  Use the RSS_NONBLOCKING_CKPT configuration parameter to enable non-blocking checkpoint at RS secondary server.
- RTO_SERVER_RESTART configuration parameter
  Use the RTO_SERVER_RESTART configuration parameter to specify recovery time objective (RTO) standards for the amount of time, in seconds, that IBM Informix has to recover from a problem after you restart the server and bring it into online or quiescent mode.
- S6_USE_REMOTE_SERVER_CFG configuration parameter
  Use the S6_USE_REMOTE_SERVER_CFG configuration parameter to control whether the file specified by the REMOTE_SERVER_CFG configuration parameter is used to authenticate secure connections for server clusters and Enterprise Replication.
- SB_CHECK_FOR_TEMP configuration parameter
  Use the SB_CHECK_FOR_TEMP configuration parameter to prevent the copying of a temporary smart large object into a permanent table.
- SBSPACENAME configuration parameter
  Use the SBSPACENAME configuration parameter specifies the name of the default sbspace.
- SBSPACETEMP configuration parameter
  Use the SBSPACETEMP configuration parameter to specify a list of default temporary sbspace for storing temporary smart large objects without metadata or user-data logging. If you store temporary smart large objects in a standard sbspace, the metadata is logged.
- SDS_ALTERNATE configuration parameter
  Use the SDS_ALTERNATE configuration parameter to define an alternate means of communication between the primary server and SD secondary servers in a high-availability cluster.
- SDS_ENABLE configuration parameter
  Use the SDS_ENABLE configuration parameter to enable SD secondary server functionality.
- SDS_FLOW_CONTROL configuration parameter
  Specifies when flow control occurs in a high-availability cluster that contains at least one shared-disk (SD) secondary server.
- SDS_LOGCHECK configuration parameter
  Use the SDS_LOGCHECK configuration parameter to set the number of seconds to delay the secondary server from taking over the role of the primary server. If the secondary server detects that the primary server is generating log records during the delay period, then the failover is prevented. The delay can prevent an unnecessary failover if network communication between the primary and secondary servers is temporarily unavailable.
- SDS_PAGING configuration parameter
  The SDS_PAGING configuration parameter specifies the location of two files that serve as buffer paging files.
- SDS_TEMPDBS configuration parameter
  Use the SDS_TEMPDBS configuration parameter to specify information that the shared disk (SD) secondary server uses to dynamically create temporary dbspaces. This configuration parameter can be specified only on the SD secondary server.
- SDS_TIMEOUT configuration parameter
  Use the SDS_TIMEOUT configuration parameter to specify the amount of time in seconds that the primary server in a high-availability cluster will wait for a log-position acknowledgment to be sent from a shared disk (SD) secondary server.
- SEC_APPLY_POLLTIME configuration parameter
  Use the SEC_APPLY_POLLTIME configuration parameter to control how long log replay thread should poll for new work before yielding.
- SEC_DR_BUFS configuration parameter
  Use the SEC_DR_BUFS configuration parameter to control the number of replication buffers to be used for replicating log records to secondary server. Buffer size is same as LOGBUFF config value.
- SEC_LOGREC_MAXBUFS configuration parameter
  Use the SEC_LOGREC_MAXBUFS configuration parameter to control the number of log buffers to be used for replaying log records at secondary server. Each log buffer is of size 16KB.
- SEC_NONBLOCKING_CKPT configuration parameter
  Use the SEC_NONBLOCKING_CKPT configuration parameter to enable non-blocking checkpoint at HDR and RS secondary server.
- SECURITY_LOCALCONNECTION configuration parameter
  Use the SECURITY_LOCALCONNECTION configuration parameter to verify security on local connections by verifying that the ID of the local user who is running a program is the same ID of the user who is trying to access the database.
- SEQ_CACHE_SIZE configuration parameter
  Use the SEQ_CACHE_SIZE configuration parameter to specify the maximum number of sequence objects that are cached in memory.
- SERVERNUM configuration parameter
  The SERVERNUM configuration parameter specifies a relative location in shared memory.
- SESSION_LIMIT_LOCKS configuration parameter
  The SESSION_LIMIT_LOCKS configuration parameter specifies the maximum number of locks available in a session. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.
- SESSION_LIMIT_LOGSPACE configuration parameter
  The SESSION_LIMIT_LOGSPACE configuration parameter specifies the maximum amount of log space that a session can use for individual transactions. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.
- SESSION_LIMIT_MEMORY configuration parameter
  The SESSION_LIMIT_MEMORY configuration parameter specifies the maximum amount of memory that a session can allocate. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.
- SESSION_LIMIT_TEMPSPACE configuration parameter
  The SESSION_LIMIT_TEMPSPACE configuration parameter specifies the maximum amount of temporary table space that a session can allocate. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

- SESSION_LIMIT_TXN_TIME configuration parameter
  The SESSION_LIMIT_TXN_TIME configuration parameter specifies the maximum amount of time that a transaction can run in a session. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.
- SHMADD configuration parameter
  Use the SHMADD configuration parameter to specify the size of the segments that are dynamically added to the virtual portion of shared memory.
- SHMBASE configuration parameter
  Use the SHMBASE configuration parameter to specifiy the base address where shared memory is attached to the memory space of a virtual processor.
- SHMNOACCESS configuration parameter
  The SHMNOACCESS configuration parameter specifies a virtual memory address range to not use to attach shared memory.
- SHMTOTAL configuration parameter
  Use the SHMTOTAL configuration parameter to specify the total amount of shared memory (resident, virtual, communications, and virtual extension portions) to be used by the database server for all memory allocations. The onconfig.std value of 0 implies that no limit on memory allocation is stipulated.
- SHMVIRT_ALLOCSEG configuration parameter
  Use the SHMVIRT_ALLOCSEG configuration parameter to specify a threshold at which Informix should allocate a new shared memory segment and the level of the event alarm activated if the server cannot allocate the new memory segment.
- SHMVIRTSIZE configuration parameter
  Use the SHMVIRTSIZE configuration parameter to specify the initial size of a virtual shared-memory segment.
- SINGLE_CPU_VP configuration parameter
  The SINGLE_CPU_VP configuration parameter specifies whether or not the database server is running with only one CPU virtual processor.
- SMX_COMPRESS configuration parameter
  Use the SMX_COMPRESS configuration parameter to specify the level of compression that the database server uses before sending data from the source database server to the target database server.
- SMX_NUMPIPES configuration parameter
  The SMX_NUMPIPES configuration parameter sets the number of pipes for server multiplexer group (SMX) connections.
- SMX_PING_INTERVAL configuration parameter
  Use the SMX_PING_INTERVAL configuration parameter to specify the number of seconds in a timeout interval, where a secondary server waits for activity from the primary server in a Server Multiplexer Group (SMX) connection.
- SMX_PING_RETRY configuration parameter
  Use the SMX_PING_RETRY configuration parameter to specify the maximum number of times that a secondary server repeats the timeout interval that is specified by the SMX_PING_INTERVAL configuration parameter if a response from the primary server is not received. If the maximum number is reached without a response, the secondary server prints an error message in the online.log and closes the Server Multiplexer Group (SMX) connection.
- SP_AUTOEXPAND configuration parameter
  Use the SP_AUTOEXPAND configuration parameter to enable or disable the automatic creation or extension of chunks.
- SP_THRESHOLD configuration parameter
  Use the SP_THRESHOLD configuration parameter to define the minimum amount of free kilobytes that can exist in a storage space before IBM Informix automatically runs a task to expand the space, either by extending an existing chunk in the space or by adding a new chunk.
- SP_WAITTIME configuration parameter
  Use the SP_WAITTIME configuration parameter to specify the maximum number of seconds that a thread waits for a dbspace, temporary dbspace, plogspace, sbspace, temporary sbspace, or blobspace space to expand before returning an out-of-space error.
- SQL_LOGICAL_CHAR configuration parameter
  Use the SQL_LOGICAL_CHAR configuration parameter to enable or disable the expansion of size specifications in declarations of built-in character data types.
- SQLTRACE configuration parameter
  Use the SQLTRACE parameter to control the startup environment of SQL tracing.
- SSL_KEYSTORE_LABEL configuration parameter
  Use the SSL_KEYSTORE_LABEL configuration parameter to specify the label of the server digital certificate used in the keystore database, a protected database that stores SSL keys and digital certificates.
- STACKSIZE configuration parameter
  Use the STACKSIZE configuration parameter to specify the stack size for the database server user threads.
- STATCHANGE configuration parameter
  Use the STATCHANGE configuration parameter to specify a positive integer for a global percentage of a change threshold for the server to use to determine if distribution statistics qualify for an update when the automatic mode for UPDATE STATISTICS operations is enabled.
- STMT_CACHE configuration parameter
  Use the STMT_CACHE configuration parameter to determine whether the database server uses the SQL statement cache.
- STMT_CACHE_HITS configuration parameter
  Use the STMT_CACHE_HITS configuration parameter to specify the number of hits (references) to a statement before it is fully inserted in the SQL statement cache.
- STMT_CACHE_NOLIMIT configuration parameter
  Use the STMT_CACHE_NOLIMIT configuration parameter to control whether to insert qualified statements into the SQL statement cache.
- STMT_CACHE_NUMPOOL configuration parameter
  Use the STMT_CACHE_NUMPOOL configuration parameter to specify the number of memory pools for the SQL statement cache. To obtain information about these memory pools, use **onstat -g ssc pool**.
- STMT_CACHE_QUERY_PLAN configuration parameter
  Use the STMT_CACHE_QUERY_PLAN configuration parameter to produce a query plan from any query that exists in the Statement Cache.
- STMT_CACHE_SIZE configuration parameter
  Use the STMT_CACHE_SIZE configuration parameter to specify the size of the SQL statement caches in kilobytes. The new cache size takes effect the next time a statement is added to a cache.
- STOP_APPLY configuration parameter
  Use the STOP_APPLY configuration parameter to stop an RS secondary server from applying log files received from the primary server.
- STORAGE_FULL_ALARM configuration parameter
  Use the STORAGE_FULL_ALARM configuration parameter to configure the frequency and severity of messages and alarms when storage spaces become full.
- SYSALARMPROGRAM configuration parameter
  Use the SYSALARMPROGRAM configuration parameter to specify the full path name of the evidence.sh script. The database server executes evidence.sh when a database server failure occurs. You can use the output from the evidence.sh script to diagnose the cause of a database server failure.
- SYSSBSPACENAME configuration parameter
  Use the SYSSBSPACENAME configuration parameter to specify the name of the sbspace in which the database server stores fragment-level data-distribution statistics, which the **syfragsdist** system catalog table stores as BLOB objects in its **encsdist** column. Also use SYSSBSPACENAME to specify the name of the sbspace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types.
- TBLSPACE_STATS configuration parameter
  Use the TBLSPACE_STATS configuration parameter to turn on and off the collection of tblspace statistics. Use the **onstat -g ppf** command to list tblspace statistics.
- TBLTBLFIRST configuration parameter
  Use the TBLTBLFIRST configuration parameter if you want to specify the first extent size of tblspace **tblspace** in the root dbspace. Set this parameter if you do not

want the database server to automatically manage the extent size.
- TBLTBLNEXT configuration parameter
  The TBLTBLNEXT configuration parameter specifies the next extent size of tblspace **tblspace** in the root dbspace. Set this parameter if you do not want the database server to automatically manage the extent size.
- TEMPTAB_NOLOG configuration parameter
  Use the TEMPTAB_NOLOG configuration parameter to disable logging on temporary tables.
- TENANT_LIMIT_CONNECTIONS configuration parameter
  The TENANT_LIMIT_CONNECTIONS configuration parameter specifies the maximum number of connections to a tenant database.
- TENANT_LIMIT_MEMORY configuration parameter
  The TENANT_LIMIT_MEMORY configuration parameter specifies the maximum amount of shared memory for all sessions that are connected to the tenant database.
- TENANT_LIMIT_SPACE configuration parameter
  The TENANT_LIMIT_SPACE configuration parameter specifies the maximum amount of storage space available to a tenant database. Storage space includes all permanent dbspaces, BLOB spaces, and sbspaces.
- TLS_VERSION configuration parameter
- TXTIMEOUT configuration parameter
  Use the TXTIMEOUT configuration parameter to specify the amount of time that a participant in a two-phase commit waits before it initiates participant recovery. This parameter is used only for distributed queries that involve a remote database server. Nondistributed queries do not use this parameter.
- UNSECURE_ONSTAT configuration parameter
  Use the UNSECURE_ONSTAT configuration parameter to remove the database system administrator (DBSA) user access restriction for onstat commands.
- UPDATABLE_SECONDARY configuration parameter
  Use the UPDATABLE_SECONDARY configuration parameter to set the number of connections to establish between the primary and secondary servers. Setting this configuration parameter enables client applications to perform update, insert, and delete operations on a high-availability secondary server.
- USELASTCOMMITTED configuration parameter
  Use the USELASTCOMMITTED configuration parameter to specify the isolation level for which the LAST COMMITTED feature of the COMMITTED READ isolation level is implicitly in effect.
- USEOSTIME configuration parameter
  Use the USEOTIME configuration parameter to control whether the database server uses subsecond precision when obtaining the current time from the operating system.
- USERMAPPING configuration parameter (UNIX, Linux)
  Use the USERMAPPING configuration parameter to set whether or not the database server accepts connections from mapped users.
- USRC_HASHSIZE configuration parameter
  The USRC_HASHSIZE configuration parameter specifies the number of hash buckets in the LBAC credential memory cache. This memory cache holds information about the LBAC credentials of users.
- USRC_POOLSIZE configuration parameter
  The USRC_POOLSIZE configuration parameter specifies the maximum number of entries in each hash bucket of the LBAC credential memory cache. This memory cache holds information about the LBAC credentials of users.
- USTLOW_SAMPLE configuration parameter
  Use the USTLOW_SAMPLE configuration parameter to enable the generation of index statistics based on sampling when you run UPDATE STATISTICS statements in LOW mode.
- VP_MEMORY_CACHE_KB configuration parameter
  Use the VP_MEMORY_CACHE_KB parameter to create a private memory cache for each CPU virtual processor and tenant virtual processor.
- VPCLASS configuration parameter
  Use the VPCLASS configuration parameter to create and configure virtual processors.
- VP_KAIO_PERCENT configuration parameter
  VP_KAIO_PERCENT is the percentage of total KAIO event resources on the system that each CPU VP will allocate.
- WSTATS configuration parameter
  Use the WSTATS configuration parameter to specify whether the **onstat -g wst** command displays wait statistics for threads within the system.

onconfig file

When you add or change information in the onconfig file, you must follow the conventions that are used in the file.

The parameter description and the possible values are specified in the comments above their entries in the onconfig.std file.

The following line shows the syntax for a parameter line:

```
PARAMETER_NAME parameter_value comments
```

The following rules describe the onconfig file behavior:

- Each parameter is on a separate line.
- Lines that start with the # symbol are comments.
- The maximum line limit of the onconfig file is 512 bytes. Lines that exceed this limit are truncated and might cause configuration problems.
- White space (tabs, spaces, or both) is required between the parameter name, the parameter value, and an optional comment. Do not use any tabs or spaces within a parameter value. Any characters after the parameter value and blank space are interpreted as comments, regardless of whether they are preceded by a # symbol.
- Parameters and their values are case-sensitive. The parameter names are always uppercase. If the value entry is described with uppercase letters, you must use uppercase (for example, the CPU value of the NETTYPE parameter).
- Most parameters can have one valid entry. If more than one entry for these parameters exists in the onconfig file, the first entry is used. Some parameters, however, can have multiple entries, such as the DBSERVERALIASES configuration parameter, which requires a comma between entries. Some parameters, such as the VPCLASS configuration parameter, can exist multiple times.
- Unrecognized parameters are copied but ignored and no error is given.

Tip: If you run a utility like **grep** on the onconfig.std template file, specify the new line character (^) to return just the configuration parameter name and value. Without the new line character, the parameter description is also returned.
For example, the following command returns both the configuration parameter description and the value:

```
grep "MSGPATH" onconfig.std
# MSGPATH      - The path of the IDS message log file
MSGPATH $INFORMIXDIR/tmp/online.log
```

Whereas, the following command returns only the configuration parameter value:

```
grep "^MSGPATH" onconfig.std
MSGPATH $INFORMIXDIR/tmp/online.log
```

Conventions for environment variables

You can enter an environment variable as a value in any configuration parameter in which the variable is applicable. For example, for the DBSERVERNAME configuration parameter you can specify the following environment variable instead of the name of your database server:

```
DBSERVERNAME     $MY_DBSERVERNAME
```

Important: If you enter an environment variable as a value, you must set that environment variable in the environment of any executable program or utility that reads the onconfig file. Utilities that read the onconfig file include the **oninit**, **oncheck**, **onbar**, **ontape**, **onlog**, and **archecker** utilities.

- Modifying the onconfig file
  You can modify the onconfig file for your database server to customize server function or tune server behavior.
- Displaying the settings in the onconfig file
  There are several tools that you can use to display the settings in the onconfig file.

Related tasks:
Setting local environment variables for utilities

Modifying the onconfig file

You can modify the onconfig file for your database server to customize server function or tune server behavior.

By default, the onconfig file is in the INFORMIXDIR/etc directory. The ONCONFIG environment variable specifies the name and location of the onconfig file.

The onconfig.std file is a template configuration file from which you can copy configuration parameter settings. The onconfig.std file is a template and not a functional configuration. You can copy and rename the onconfig.std file, but do not modify or delete theonconfig.std file. If you omit a parameter value in your copy of the configuration file, the database server either uses default values in onconfig.std template file or calculates values that are based on other parameter values.

You can modify the onconfig file by any of the following methods:

- You can use a text editor to modify configuration parameter values. The changes take effect after the next time the database server is shut down and restarted.
- You can modify the values of many configuration parameters dynamically without restarting the database server by running the **onmode -wf** to update configuration parameters permanently or by running the **onmode -wm** command to update configuration parameters in memory.
- You can generate an onconfig file with settings that are optimized for the connections, disk space, and CPU usage that you estimate by running the **genoncfg** utility.
- You can export, import, and modify configuration parameters in groups:
  - Use the **onmode –we** command to export a snapshot of the current configuration to a file. The resulting snapshot can then be archived, used as a configuration file, or imported to another running instance.
  - Use the **onmode –wi** command to import tunable configuration parameters from a previously exported file. Configuration parameters in the file that are not dynamically tunable are ignored.
- You can modify, reset, export, and import a configuration file with SQL administration API commands:
  - Use modify config argument with the **admin()** or **task()** function to change the value of a configuration parameter.
  - Use the export config and import config arguments with the **admin()** or **task()** function to export or import a file that contains one or more dynamically tunable configuration parameters.
  - Use the reset config or reset config all argument with the **admin()** or **task()** function to revert the value of a configuration parameter or all configuration parameters to its value in the onconfig file.

You can compare two onconfig files by running the **onconfig_diff** utility.

Related reference:
The genoncfg Utility
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -we: Export a file that contains current configuration parameters
onmode -wi: Import a configuration parameter file
modify config arguments: Modify configuration parameters (SQL administration API)
reset config argument: Revert configuration parameter value (SQL administration API)
reset config all argument: Revert all dynamically updatable configuration parameter values (SQL administration API)
import config argument: Import configuration parameter values (SQL administration API)
export config argument: Export configuration parameter values (SQL administration API)
The onconfig_diff utility
Related information:
Database server configuration

Displaying the settings in the onconfig file

There are several tools that you can use to display the settings in the onconfig file.

To display the settings in the onconfig file, use one of the following tools:

- Open the onconfig file with a text editor.
- View the contents of the onconfig file with the **onstat -c** command.
- View a list of configuration parameters and their current values by running the **onstat -g cfg** command. If configuration parameters are updated dynamically, the current values differ from the permanent values in the onconfig file.
  You can use additional options with the **onstat -g cfg** command to display only the configuration parameters that were changed dynamically or to display additional information about all configuration parameters.

Related reference:
onstat -c command: Print ONCONFIG file contents
onstat -g cfg command: Print the current values of configuration parameters
Related information:
ONCONFIG environment variable

onconfig Portal: Configuration parameters by functional category

The information in this section lists configuration parameters as they are in the UNIX onconfig.std file.

Category list

To use this section, you first determine the appropriate category from the following list, then follow the link to the configuration parameters for that category. The categories are listed in the same order as they are in the onconfig.std file. Parameters that are not in the onconfig.std file but that you can add to your onconfig file are listed in Table 60.

- WebSphere MQ server configuration parameters
- Non-root user server installation configuration parameters
- Low memory configuration parameters
- Global Security Kitconfiguration parameter
- Connection parameters
- Session limits
- Tenant limits
- Java configuration parameters
- Buffer pool and LRU tuning configuration parameters
- Additional parameters

Root dbspace configuration parameters

Use the following configuration parameters to configure the root dbspace.

Table 1. Root dbspace configuration parameters

Configuration Parameter	Reference
ROOTNAME configuration parameter	The root dbspace name.
ROOTPATH configuration parameter	The path for the root dbspace.
ROOTOFFSET configuration parameter	The offset for the root dbspace.
ROOTSIZE configuration parameter	The size of the root dbspace.
MIRROR configuration parameter	Enables or disables mirroring.
MIRRORPATH configuration parameter	The path for the mirrored root dbspace.
MIRROROFFSET configuration parameter	The offset for the mirrored root dbspace.

Physical log configuration parameters

Use the following configuration parameters to configure physical logs.

Table 2. Physical log configuration parameters

Configuration Parameter	Reference
PHYSFILE configuration parameter	The size of the physical log.
PLOG_OVERFLOW_PATH configuration parameter	The overflow directory for physical log files.
PHYSBUFF configuration parameter	The size of the physical log buffer.

Logical log configuration parameters

Use the following configuration parameters to configure logical logs.

Table 3. Logical log configuration parameters

Configuration Parameter	Reference
LOGFILES configuration parameter	The number of logical log files.
LOGSIZE configuration parameter	The size of each logical log file.
DYNAMIC_LOGS configuration parameter	The type of dynamic log allocation.
LOGBUFF configuration parameter	The size of the logical log buffer.

Long transaction configuration parameters

Use the following configuration parameters to control when long transactions are rolled back.

Table 4. Long transaction configuration parameters

Configuration Parameter	Reference
LTXHWM configuration parameter	The percentage of the logical log files that can be filled before a long transaction is rolled back.
LTXEHWM configuration parameter	The percentage of the logical log files that can be filled before the server suspends other activities so that a long transaction has exclusive use of the logs.

Server message file configuration parameters

Use the following configuration parameters to configure the server message file.

Table 5. Server message file configuration parameters

Configuration Parameter	Reference
MSGPATH configuration parameter	The path of the message file.
CONSOLE configuration parameter	The path of the console message file.

Tblspace configuration parameters

Use the following configuration parameters to configure the **tblspace** in the root dbspace.

Table 6. Tblspace configuration parameters

Configuration Parameter	Reference
TBLTBLFIRST configuration parameter	The first extent size for the tblspace **tblspace**.
TBLTBLNEXT configuration parameter	The next extent size for the tblspace **tblspace**.
TBLSPACE_STATS configuration parameter	Enables or disables tblspace statistics.

Temporary dbspace and sbspace configuration parameters

Use the following configuration parameters to configure the default temporary dbspaces and sbspaces.

Table 7. Temporary dbspace and sbspace configuration parameters

Configuration Parameter	Reference
DBSPACETEMP configuration parameter	The list of dbspaces for temporary objects.
SBSPACETEMP configuration parameter	The list of sbspaces for temporary smart large objects.

Dbspace and sbspace configuration parameters

Use the following configuration parameters to configure the default dbspaces and sbspaces.

Table 8. Default dbspaces and sbspaces configuration parameters

Configuration Parameter	Reference
SBSPACENAME configuration parameter	The default sbspace to store smart large objects.
SYSSBSPACENAME configuration parameter	The default sbspace for system statistics.
ONDBSPACEDOWN configuration parameter	Specifies the behavior of the server when a dbspace is down.

System configuration parameters

Use the following configuration parameters to set server instance information.

Table 9. System configuration parameters

Configuration Parameter	Reference
SERVERNUM configuration parameter	The unique ID for the database server instance.
DBSERVERNAME configuration parameter	The name of the default database server.
DBSERVERALIASES configuration parameter	List of alternative database server names.
FULL_DISK_INIT configuration parameter	Prevents an accidental disk reinitialization of an existing server instance.

Network configuration parameters

Use the following configuration parameters to configure the network.

Table 10. Network configuration parameters

Configuration Parameter	Reference
NETTYPE configuration parameter	The configuration of poll threads for a specific protocol.
LISTEN_TIMEOUT configuration parameter	The time the database server waits for a connection.
MAX_INCOMPLETE_CONNECTIONS configuration parameter	The maximum number of incomplete connections.
FASTPOLL configuration parameter	Enables or disables fast polling.
NUMFDSERVERS configuration parameter	For network connections on UNIX, use the NUMFDSERVERS configuration parameter to specify the maximum number of poll threads to handle network connections that are moving between VPs.
NS_CACHE configuration parameter	Defines the maximum retention time for an individual entry in the host name/IP address cache, the service cache, the user cache, and the group cache.

CPU-related configuration parameters

Use the following configuration parameters to configure CPU virtual processors.

Table 11. CPU virtual processors configuration parameters

Configuration Parameter	Reference

Configuration Parameter	Reference
MULTIPROCESSOR configuration parameter	Setting of 1 supports multiple CPU VPs.
VPCLASS configuration parameter	Defines the properties of each CPU virtual processor class.
VP_MEMORY_CACHE_KB configuration parameter	The amount of private memory blocks for the CPU virtual processors.
SINGLE_CPU_VP configuration parameter	Set to 0 to enable user-defined CPU VPs, or 1 for a single CPU VP.

Automatic tuning configuration parameters

Use the following configuration parameters to automatically tune the configuration of the database server.

Table 12. CPU virtual processors configuration parameters

Configuration Parameter	Reference
AUTO_TUNE configuration parameter	Enable or disables all automatic tuning configuration parameters that have values that are not present in the onconfig file.
AUTO_LRU_TUNING configuration parameter	Enables or disables automatic tuning of LRU queues:
AUTO_AIOVPS configuration parameter	Enables or disables automatic management of AIO virtual processors.
AUTO_CKPTS configuration parameter	Enables or disables automatic checkpoints.
AUTO_REPREPARE configuration parameter	Enables or disables automatically reoptimizing stored procedures and repreparing prepared statements.
AUTO_STAT_MODE configuration parameter	Enables or disables the mode for selectively updating statistics for your system.
AUTO_READAHEAD configuration parameter	Changes the automatic read-ahead mode or disables or enables automatic read ahead for a query.

AIO and cleaner-related configuration parameters

Use the following configuration parameters to configure AIO virtual processors and buffer cleaners.

Table 13. AIO and buffer cleaner configuration parameters

Configuration Parameter	Reference
VPCLASS configuration parameter	Configures the AIO virtual processors.
CLEANERS configuration parameter	The number of page cleaner threads.
DIRECT_IO configuration parameter (UNIX)	Specifies whether to use direct I/O.

Lock-related configuration parameters

Use the following configuration parameters to set locking behavior.

Table 14. Locking configuration parameters

Configuration Parameter	Reference
LOCKS configuration parameter	The initial number of locks at startup.
DEF_TABLE_LOCKMODE configuration parameter	The default table lock mode.

Shared memory configuration parameters

Use the following configuration parameters to configure shared memory.

Table 15. Shared memory configuration parameters

Configuration Parameter	Reference
RESIDENT configuration parameter	Controls whether shared memory is resident.
SHMBASE configuration parameter	The shared memory base address. Do not change this value.
SHMVIRTSIZE configuration parameter	The initial size, in KB, of the virtual segment of shared memory.
SHMADD configuration parameter	The size of virtual shared memory segments.
EXTSHMADD configuration parameter	The size of each virtual-extension shared memory segment for user-defined routines and DataBlade routines that run in user-defined virtual processors.
SHMTOTAL configuration parameter	The maximum amount of shared memory for the database server.
SHMVIRT_ALLOCSEG configuration parameter	Controls when to add a memory segment.
SHMNOACCESS configuration parameter	Lists shared memory addresses that the server cannot access.

Checkpoint and system block configuration parameters

Use the following configuration parameters to configure checkpoints, recovery time objective, and system block time.

Table 16. Checkpoints, recovery time objective, and system block time configuration parameters

Configuration Parameter	Reference
CKPTINTVL configuration parameter	How often to check if a checkpoint is needed.
RTO_SERVER_RESTART configuration parameter	The recovery time objective for a restart after a failure.
BLOCKTIMEOUT configuration parameter	The amount of time for a system block.

Conversion guard configuration parameters

Use the following configuration parameters to control information Informix uses during an upgrade to a new version of the server.

Table 17. Conversion guard configuration parameters

Configuration Parameter	Reference
CONVERSION_GUARD configuration parameter	Specifies whether to stop or continue an upgrade if an error occurs during the upgrade.
RESTORE_POINT_DIR configuration parameter	Specifies the path name to an empty directory where restore point files are placed during a failed upgrade when the CONVERSION_GUARD configuration parameter is enabled.

Transaction-related configuration parameters

Use the following configuration parameters to control distributed transactions.

Table 18. Distributed transaction configuration parameters

Configuration Parameter	Reference
TXTIMEOUT configuration parameter	The distributed transaction timeout period.
DEADLOCK_TIMEOUT configuration parameter	The maximum amount of time to wait for a lock in a distributed transaction.
HETERO_COMMIT configuration parameter	Enables or disables heterogeneous commits for transactions that use an EGM gateway.

ontape Tape device configuration parameters

Use the following configuration parameters to configure the tape device for backups with the **ontape** utility.

Table 19. Tape device configuration parameters

Configuration Parameter	Reference
TAPEDEV configuration parameter	The tape device for backups.
TAPEBLK configuration parameter	The tape block size.
TAPESIZE configuration parameter	The maximum amount of data to put on one backup tape.

ontape Logical log tape device configuration parameters

Use the following configuration parameters to configure the tape device for logical logs with the **ontape** utility.

Table 20. Logical log tape device configuration parameters

Configuration Parameter	Reference
LTAPEDEV configuration parameter	The tape device for logical log backups.
LTAPEBLK configuration parameter	The tape block size for logical log backups.
LTAPESIZE configuration parameter	The maximum amount of data to put on one logical log backup tape.

Backup and restore configuration parameters

Use the following configuration parameters to control backup and restore with the ON-Bar utility. Unless specified otherwise, these configuration parameters are documented in the *IBM Informix Backup and Restore Guide*.

Table 21. ON-Bar configuration parameters

Configuration Parameter	Reference
BAR_ACT_LOG configuration parameter	The location of the ON-Bar activity log file.
BAR_DEBUG_LOG configuration parameter	The location of the ON-Bar debug log file.
BAR_DEBUG configuration parameter	The debug level for ON-Bar.
BAR_MAX_BACKUP configuration parameter	The number of backup threads used in a backup.
BAR_MAX_RESTORE configuration parameter	The number of restore threads used in a restore.
BAR_RETRY configuration parameter	The number of times to try a backup or restore again.
BAR_NB_XPORT_COUNT configuration parameter	The number of data buffers each backup process uses.
BAR_XFER_BUF_SIZE configuration parameter	The size of each data buffer.
RESTARTABLE_RESTORE configuration parameter	Enables ON-Bar to continue a backup after a failure.

Configuration Parameter	Reference
BAR_PROGRESS_FREQ configuration parameter	How often progress messages are put in the activity log.
BAR_BSALIB_PATH configuration parameter	The path for the shared library for ON-Bar and the storage manager.
BACKUP_FILTER configuration parameter	The path of a filter program to use during backups.
RESTORE_FILTER configuration parameter	The path of a filter program to use during restores.
BAR_PERFORMANCE configuration parameter	The type of ON-Bar performance statistics to report.
BAR_CKPTSEC_TIMEOUT configuration parameter	Time in seconds to wait for an archive checkpoint to complete in the secondary server.

Primary Storage Manager configuration parameters

Use the following configuration parameters to configure the IBM Informix Primary Storage Manager.

Table 22. Informix Primary Storage Manager configuration parameters

Configuration Parameter	Reference
PSM_ACT_LOG configuration parameter	Specifies the location of the Informix Primary Storage Manager activity log if you do not want the log information included in the ON-Bar activity log.
PSM_DEBUG_LOG configuration parameter	Specifies the location of the Informix Primary Storage Manager debug log if you do not want the log information included in the ON-Bar debug log.
PSM_DEBUG configuration parameter	Specifies the amount of information that prints in the Informix Primary Storage Manager debug log if you want to use a debug level that is different from the one used by ON-Bar.
PSM_CATALOG_PATH configuration parameter	Specifies the full path to the directory that contains the Informix Primary Storage Manager catalog tables.
PSM_DBS_POOL configuration parameter	Specifies the name of the pool in which theInformix Primary Storage Manager places backup and restore dbspace data.
PSM_LOG_POOL configuration parameter	Specifies the name of the pool in which theInformix Primary Storage Manager places backup and restore log data.

Data dictionary cache configuration parameters

Use the following configuration parameters to configure the data dictionary caches.

Table 23. Data dictionary cache configuration parameters

Configuration Parameter	Reference
DD_HASHSIZE configuration parameter	The number of hash buckets in the data dictionary cache.
DD_HASHMAX configuration parameter	The maximum number of tables in each hash bucket.

Data distribution configuration parameters

Use the following configuration parameters to configure the data distribution pools.

Table 24. Data distribution configuration parameters

Configuration Parameter	Reference
DS_HASHSIZE configuration parameter	The number of hash buckets in the data distribution cache and other caches.
DS_POOLSIZE configuration parameter	The maximum number of entries in the data distribution cache and other caches.

User defined routine (UDR) configuration parameters

Use the following configuration parameters to configure UDRs.

Table 25. UDR configuration parameters

Configuration Parameter	Reference
PC_HASHSIZE configuration parameter	The number of hash buckets in the UDR cache.
PC_POOLSIZE configuration parameter	The maximum number of entries in the UDR cache.
PRELOAD_DLL_FILE configuration parameter	The C UDR shared library path name to load when the server starts.

SQL statement cache configuration parameters

Use the following configuration parameters to configure the SQL statement cache.

Table 26. SQL statement cache configuration parameters

Configuration Parameter	Reference
STMT_CACHE configuration parameter	Controls SQL statement caching.
STMT_CACHE_HITS configuration parameter	The number of times an SQL statement is run before it is cached.

Configuration Parameter	Reference
STMT_CACHE_SIZE configuration parameter	The size of the SQL statement cache.
STMT_CACHE_NOLIMIT configuration parameter	Controls additional memory consumption of the SQL statement cache.
STMT_CACHE_NUMPOOL configuration parameter	The number of pools for the SQL statement cache.

Operating system session-related configuration parameters

Use the following configuration parameters to configure operating system and session features.

Table 27. Operating system and session configuration parameters

Configuration Parameter	Reference
USEOSTIME configuration parameter	The precision of SQL statement timing.
STACKSIZE configuration parameter	The size of a session stack.
ALLOW_NEWLINE configuration parameter	Whether embedded new line characters are allowed in SQL statements.
USELASTCOMMITTED configuration parameter	Controls committed read isolation level.

Index-related configuration parameters

Use the following configuration parameters to configure index features.

Table 28. Index configuration parameters

Configuration Parameter	Reference
FILLFACTOR configuration parameter	The percentage of index page fullness.
MAX_FILL_DATA_PAGES configuration parameter	Enables or disables filling data pages as full as possible if they have variable length rows.
BTSCANNER Configuration Parameter	Configures B-tree scanner threads.
ONLIDX_MAXMEM configuration parameter	The amount of memory for the pre-image and updator log pools.

Parallel database queries (PDQ) configuration parameters

Use the following configuration parameters to configure PDQ.

Table 29. PDQ configuration parameters

Configuration Parameter	Reference
MAX_PDQPRIORITY configuration parameter	The maximum percentage of resources for a single query.
DS_MAX_QUERIES configuration parameter	The maximum number of concurrent decision support queries.
DS_TOTAL_MEMORY configuration parameter	The maximum amount of decision support memory.
DS_MAX_SCANS configuration parameter	The maximum number of decision support scans.
DS_NONPDQ_QUERY_MEM configuration parameter	The amount of non-PDQ query memory.
DATASKIP Configuration Parameter	Whether to skip a dbspace when processing a query.

Optimizer configuration parameters

Use the following configuration parameters to influence query execution optimizer plans and directives.

Table 30. Optimizer configuration parameters

Configuration Parameter	Reference
OPTCOMPIND configuration parameter	Controls how the optimizer determines the best query path.
DIRECTIVES configuration parameter	Enables or disables inline optimizer directives.
EXT_DIRECTIVES configuration parameter	Enables or disables external directives.
OPT_GOAL configuration parameter	Controls how to optimize for fastest retrieval.
IFX_FOLDVIEW configuration parameter	Enables or disables folding views.
STATCHANGE configuration parameter	Specifies a positive integer for a global percentage of a change threshold to identify data distribution statistics that need to be updated.
USTLOW_SAMPLE configuration parameter	Enables or disables the generation of index statistics based on sampling when you run UPDATE STATISTICS statements in LOW mode.

Scan configuration parameters

Use the following configuration parameters to set read-ahead behavior.

Table 31. Scan configuration parameters

Configuration Parameter	Reference
BATCHEDREAD_TABLE configuration parameter	Enables or disables light scans on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data.
BATCHEDREAD_INDEX configuration parameter	Enables the optimizer to perform light scans for indexes.

SQL tracing configuration parameters

Use the following configuration parameters to set SQL tracing.

Table 32. SQL tracing configuration parameters

Configuration Parameter	Reference
EXPLAIN_STAT configuration parameter	Enables or disables including query statistics in the explain output file.
SQLTRACE configuration parameter	Configures SQL tracing.

Security configuration parameters

Use the following configuration parameters to configure security options.

Table 33. Security configuration parameters

Configuration Parameter	Reference
DBCREATE_PERMISSION configuration parameter	Specifies users who can create databases.
DB_LIBRARY_PATH configuration parameter	Specifies the locations of UDR or UDT shared libraries.
IFX_EXTEND_ROLE configuration parameter	Controls how to specify which users can register external routines.
SECURITY_LOCALCONNECTION configuration parameter	Whether the database server checks the security of local connections.
UNSECURE_ONSTAT configuration parameter	Whether non-DBSA users can run **onstat** commands.
ADMIN_USER_MODE_WITH_DBSA configuration parameter	Controls who can connect to the server in administration mode.
ADMIN_MODE_USERS configuration parameter	Lists the users who can connect in administration mode.
SSL_KEYSTORE_LABEL configuration parameter	The SSL label.
TLS_VERSION configuration parameter	Specifies the Transport Layer Security (TLS) version for network connections.
DISK_ENCRYPTION configuration parameter	Controls the encryption of storage spaces.

Label-based access control configuration parameters

Use the following configuration parameters to configure the label-based access control (LBAC) cache. These configuration parameters are documented in the *IBM Informix Security Guide*.

Table 34. LBAC configuration parameters

Configuration Parameter	Reference
PLCY_POOLSIZE configuration parameter	The number of hash buckets in the LBAC security information cache.
PLCY_HASHSIZE configuration parameter	The maximum number of entries in each hash bucket of the LBAC security information cache.
USRC_POOLSIZE configuration parameter	The number of hash buckets in the LBAC credential memory cache.
USRC_HASHSIZE configuration parameter	The maximum number of entries in each hash bucket of the LBAC credential memory cache.

Built-in character data types configuration parameters

Use the following configuration parameter to configure built-in character data types.

Table 35. Built-in character data types configuration parameters

Configuration Parameter	Reference
SQL_LOGICAL_CHAR configuration parameter	Enables or disables the expansion of size specifications in declarations of built-in character data types.

Sequence cache configuration parameters

Use the following configuration parameter to configure the sequence cache:

Table 36. Sequence cache data types configuration parameters

Configuration Parameter	Reference
SEQ_CACHE_SIZE configuration parameter	Specifies the maximum number of sequence objects that are cached in memory.

High-availability and Enterprise Replication security configuration parameters

Use the following configuration parameters to configure security for high-availability clusters and Enterprise Replication.

Table 37. High-availability and Enterprise Replication security configuration parameters

Configuration Parameter	Reference
ENCRYPT_HDR configuration parameter	Enables or disables encryption for HDR.
ENCRYPT_SMX configuration parameter	The level of encryption for SDS or RSS servers.
ENCRYPT_CDR Configuration Parameter	The level of encryption for Enterprise Replication.
ENCRYPT_CIPHERS configuration parameter	Lists encryption ciphers and modes.
ENCRYPT_MAC configuration parameter	The level of the message authentication code (MAC).
ENCRYPT_MACFILE configuration parameter	The paths of MAC key files.
ENCRYPT_SWITCH configuration parameter	The frequency to switch ciphers and keys.

Enterprise Replication configuration parameters

Use the following configuration parameters to configure Enterprise Replication (ER). These configuration parameters are documented in the .

Table 38. Enterprise Replication configuration parameters

Configuration Parameter	Reference
CDR_EVALTHREADS Configuration Parameter	The numbers of evaluator threads.
CDR_DSLOCKWAIT Configuration Parameter	The amount of time data sync threads wait for database locks.
CDR_QUEUEMEM Configuration Parameter	The maximum amount of memory for send and receive queues.
CDR_NIFCOMPRESS Configuration Parameter	The network interface compression level.
CDR_SERIAL Configuration Parameter	The incremental size and starting value of serial columns.
CDR_DBSPACE Configuration Parameter	The dbspace name for the **syscdr** database.
CDR_QDATA_SBSPACE Configuration Parameter	The names of sbspaces for spooled transactions.
CDR_SUPPRESS_ATSRISWARN Configuration Parameter	The data sync warnings and errors to suppress in ATS and RIS files.
CDR_DELAY_PURGE_DTC configuration parameter	The amount of time to retain delete tables.
CDR_LOG_LAG_ACTION configuration parameter	The action taken when the database server comes close to overwriting a logical log that Enterprise Replication did not yet process.
CDR_LOG_STAGING_MAXSIZE Configuration Parameter	The maximum amount of space that Enterprise Replication uses to stage log files.
CDR_MAX_DYNAMIC_LOGS Configuration Parameter	The maximum number of dynamic log requests that Enterprise Replication can make in a session.
GRIDCOPY_DIR Configuration Parameter	The default directory used by the **ifx_grid_copy** procedure.
CDR_TSINSTANCEID configuration parameter	The unique identifier for time series instances that are replicated.
CDR_MAX_FLUSH_SIZE configuration parameter	The maximum number of transactions that are applied before the logs are flushed to disk.
CDR_AUTO_DISCOVER configuration parameter	Allow auto-configuration of Enterprise Replication though the **cdr autoconfig serv** command, installation wizard, or **ifxclone** utility.
CDR_MEM configuration parameter	Specifies the method of memory pool allocation for Enterprise Replication.

Parallel sharded queries configuration parameters

Use the following configuration parameters to configure parallel sharded queries.

Table 39. Parallel sharded queries configuration parameters

Configuration Parameter	Reference
SHARD_MEM configuration parameter	Specifies how to allocate shared memory for sharded queries on a shard server.
SHARD_ID configuration parameter	Sets the unique ID for a shard server in a shard cluster.

High-availability cluster configuration parameters

Use the following configuration parameters to configure high-availability clusters.

Table 40. High-availability cluster configuration parameters

Configuration Parameter	Reference
DRAUTO configuration parameter	Controls automatic failover of primary servers.
DRINTERVAL configuration parameter	The maximum interval between buffer flushes.
HDR_TXN_SCOPE configuration parameter	Adjust transaction synchronization between client applications, the primary server, and the HDR secondary server.

Configuration Parameter	Reference
DRTIMEOUT configuration parameter	The network timeout period.
DRLOSTFOUND configuration parameter	The path of the HDR lost-and-found file.
DRIDXAUTO configuration parameter	Enables or disables automatic index repair.
HA_ALIAS configuration parameter	The server alias for a high-availability cluster.
HA_FOC_ORDER configuration parameter	Defines a single failover rule used by Connection Managers.
LOG_INDEX_BUILDS configuration parameter	Enables or disables index page logging.
SDS_ENABLE configuration parameter	Enables or disables and SD secondary server.
SDS_TIMEOUT configuration parameter	The time the primary waits for acknowledgment from an SD secondary server.
SDS_TEMPDBS configuration parameter	The temporary dbspace used by an SD secondary server.
SDS_ALTERNATE configuration parameter	The alternate means of communication between the primary server and SD secondary servers in a high-availability cluster.
SDS_PAGING configuration parameter	The paths of SD secondary paging files.
SDS_LOGCHECK configuration parameter	Whether the primary server is generating log activity and to allow or prevent failover of the primary server.
UPDATABLE_SECONDARY configuration parameter	Whether the secondary server can accept update, insert, or delete operations from clients.
FAILOVER_CALLBACK configuration parameter	The program called when a secondary server makes the transition to a standard or primary server.
TEMPTAB_NOLOG configuration parameter	The default logging mode for temporary tables.
DELAY_APPLY Configuration Parameter	The delay time for applying transactions on an RS secondary server.
STOP_APPLY configuration parameter	Stops applying transactions on an RS secondary server.
LOG_STAGING_DIR configuration parameter	The directory to stage log files.
RSS_FLOW_CONTROL configuration parameter	Enables flow control for RS secondary servers.
FAILOVER_TX_TIMEOUT configuration parameter	Enables or disables transaction survival behavior during failover.
ENABLE_SNAPSHOT_COPY configuration parameter	Whether the server instance can be cloned by the **ifxclone** utility.
SMX_COMPRESS configuration parameter	The level of compression that the database server uses when sending data from the source database server to the target database server.
SMX_PING_INTERVAL configuration parameter	The number of seconds in a timeout interval.
SMX_PING_RETRY configuration parameter	The number of timeout intervals before a secondary server closes the SMX connection to the primary server.
CLUSTER_TXN_SCOPE configuration parameter	Controls when transaction commits can be returned to a client application.
SMX_NUMPIPES configuration parameter	Sets the number of pipes for SMX connections.
SEC_NONBLOCKING_CKPT configuration parameter	Enables non-blocking checkpoint at HDR and RS secondary server.

Logical recovery configuration parameters

Use the following configuration parameters to set logical recovery threads.

Table 41. Logical recovery configuration parameters

Configuration Parameter	Reference
ON_RECVRY_THREADS configuration parameter	The number of logical recovery threads that run in parallel during a warm restore.
OFF_RECVRY_THREADS configuration parameter	The number of logical recovery threads used in a cold restore and for fast recovery.

Diagnostic dump configuration parameters

Use the following configuration parameters to control diagnostic dump information.

Table 42. Diagnostic configuration parameters

Configuration Parameter	Reference
DUMPDIR configuration parameter	The location of assertion failure diagnostic files.
DUMPSHMEM configuration parameter (UNIX)	Controls shared memory dumps.
DUMPGCORE configuration parameter (UNIX)	Enables or disables whether the database server dumps a core to the **gcore** file.
DUMPCORE configuration parameter (UNIX)	Enables or disables whether the database server dumps a core after an assertion failure.
DUMPCNT configuration parameter (UNIX)	The maximum number of shared memory dumps for a session.

Alarm program configuration parameters

Use the following configuration parameters to configure the alarm program.

Table 43. Alarm program configuration parameters

Configuration Parameter	Reference
ALARMPROGRAM configuration parameter	The alarm program to display event alarms.
ALRM_ALL_EVENTS configuration parameter	Whether the alarm program runs for all events.
STORAGE_FULL_ALARM configuration parameter	How often messages and events are raised when a storage space is full or a partition runs out of pages or extents.
SYSALARMPROGRAM configuration parameter	The system alarm program triggered after an assertion failure.

Technical support configuration parameters

The following configuration parameters to are used by technical support and are set automatically.

Table 44. Technical support configuration parameters

Configuration Parameter	Reference
RAS_PLOG_SPEED	Reserved for support.
RAS_LLOG_SPEED	Reserved for support.

Character processing configuration parameter

Use the following configuration parameter to control whether Informix checks if characters are valid for the locale.

Table 45. Character processing configuration parameter

Configuration Parameter	Reference
EILSEQ_COMPAT_MODE configuration parameter	Enables or disables checking character validity.

Statistics configuration parameters

Use the following configuration parameters to control the collection of queue and wait statistics.

Table 46. Queue and wait statistics configuration parameters

Configuration Parameter	Reference
QSTATS configuration parameter	Enables or disables collecting queue statistics.
WSTATS configuration parameter	Enables or disables collecting wait statistics.

User mapping configuration parameter

Use this configuration parameter to control user mapping.

Table 47. User mapping

Configuration Parameter	Description
USERMAPPING configuration parameter (UNIX, Linux)	Whether mapped users can connect to , and if so, whether the mapped user can have administrative privileges.

Storage provisioning configuration parameters

Use the following configuration parameters to control information that enables the server to automatically extend or add a chunk when more space is needed in an existing storage space (dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace).

Table 48. Storage provisioning configuration parameters

Configuration Parameter	Reference
SP_AUTOEXPAND configuration parameter	Enables or disables the automatic creation or extension of chunks in a storage space.
SP_THRESHOLD configuration parameter	Defines the minimum amount of free KB that can exist in a storage space.
SP_WAITTIME configuration parameter	Specifies the maximum number of seconds that a thread waits for a storage pool to expand before returning an "out of space" error.

Automatic location of database objects

Use the following configuration parameter to enable automatic location and fragmentation.

Table 49. Automatic location configuration parameter

Configuration Parameter	Reference
AUTOLOCATE configuration parameter	Enables the automatic location of databases and tables and the automatic fragmentation of tables.

Default escape configuration parameter

Use the following configuration parameter as needed.

Table 50. Default escape configuration parameter

Configuration Parameter	Reference
DEFAULTESCCHAR configuration parameter	Specifies a default escape character.

WebSphere MQ server configuration parameters

Use the following configuration parameters to configure the database server for MQ messaging. These configuration parameters are documented in the *IBM Informix Database Extensions User's Guide*.

Table 51. MQ configuration parameters

Configuration Parameter	Reference
MQSERVER configuration parameter	Defines a channel, specifies the location of the WebSphere® MQ server, and specifies the communication method to be used.
MQCHLLIB configuration parameter	Specifies the path to the directory that contains the WebSphere MQ client channel definition table.
MQCHLTAB configuration parameter	Specifies the name of WebSphere the client channel definition table.

Non-root user server installation configuration parameters

Use the following configuration parameters with non-root server installations.

Table 52. Non-root user server installation

Configuration Parameter	Reference
REMOTE_SERVER_CFG configuration parameter	Specifies the name of a file that lists the remote hosts that are trusted by the database server computer.
REMOTE_USERS_CFG configuration parameter	Specifies the name of a file that lists names of trusted users that exist on remote hosts.
S6_USE_REMOTE_SERVER_CFG configuration parameter	Specifies the file used to authenticate secure server connections in a trusted network environment.

Low memory configuration parameters

Use the following configuration parameters to manage low memory.

Table 53. Low memory configuration parameters

Configuration Parameter	Reference
LOW_MEMORY_RESERVE configuration parameter	Reserves a specific amount of memory for use when critical activities are needed and the server has limited free memory.
LOW_MEMORY_MGR configuration parameter	Change the default behavior of the server when it reaches the memory limit.

Global Security Kitconfiguration parameter

Use this parameter to set the IBM Informix Global Security Kit (GSKit) version.

Table 54. Global Security Kit

Configuration Parameter	Description
GSKIT_VERSION configuration parameter	Specifies which version of IBM Global Security Kit (GSKit) the database server uses.

Connection parameters

Use the following parameters to manage connections.

Table 55. Connection configuration parameters

Configuration Parameter	Description
INFORMIXCONRETRY configuration parameter	Specifies the number of connection attempts that can be made to the database server after the initial connection attempt fails. With the INFORMIXCONTIME configuration parameter, specifies the frequency at which the CONNECT statement tries to connect to the database server.
INFORMIXCONTIME configuration parameter	Specifies the duration, in seconds, that the CONNECT statement attempts to establish a connection to the database server. With the INFORMIXRETRY configuration parameter, specifies the frequency at which the CONNECT statement tries to connect to the database server.

Session limits

Use the following configuration parameters to create limits for individual sessions.

Table 56. Session-limit configuration parameters.

Configuration Parameter	Reference
SESSION_LIMIT_LOCKS configuration parameter	Limits the number of locks.
SESSION_LIMIT_MEMORY configuration parameter	Limits the available memory.
SESSION_LIMIT_TEMPSPACE configuration parameter	Limits temporary table space.
SESSION_LIMIT_LOGSPACE configuration parameter	Limits logspace available to individual transactions.
SESSION_LIMIT_TXN_TIME configuration parameter	Limits the amount of time that a transaction can run.

Tenant limits

Use the following configuration parameters to specify limits on tenant databases.

Table 57. Tenant limits configuration parameters.

Configuration Parameter	Reference
TENANT_LIMIT_SPACE configuration parameter	Limits the amount of storage space available to a tenant database.
TENANT_LIMIT_MEMORY configuration parameter	Limits the amount of shared memory for all sessions that are connected to the tenant database.
TENANT_LIMIT_CONNECTIONS configuration parameter	Limits the number of connections to a tenant database.

Java configuration parameters

Use the following configuration parameters to configure Java™ virtual processors. These configuration parameters are documented in the *IBM J/Foundation Developer's Guide*.

Table 58. Java configuration parameters

Configuration Parameter	Reference
VPCLASS	Configures a Java virtual processor class.
JVPPROPFILE	The Java VP property file.
JVPLOGFILE	The Java VP log file.
JVPARGS	Configures the Java VM.
JVPCLASSPATH	The Java class path.

Buffer pool and LRU tuning configuration parameters

Use the following configuration parameters to configure buffer pools and tune LRU queues.

Table 59. Buffer pool and LRU tuning configuration parameters

Configuration Parameter	Reference
BUFFERPOOL configuration parameter	Configures buffer pools.

Additional parameters

Some configuration parameters are not in the onconfig.std file. You can add these parameters to your onconfig file as necessary.

Table 60. Parameters that are not in the onconfig.std file

Configuration Parameter	Reference
AUTO_TUNE_SERVER_SIZE configuration parameter	Sets the size of the database server based on the number of expected users. If you create a server during installation, this parameter is set in your onconfig file.
AUTO_LLOG configuration parameter	Automatically adds logical logs in the specified dbspace to improve performance and to limit the total size of logical log files. If you create a server during installation, this parameter is set in your onconfig file.
CDR_APPLY Configuration Parameter	Specifies the minimum and maximum number of data sync threads.
CDR_ENV Configuration Parameter	Sets some specific Enterprise Replication environment variables.
CHECKALLDOMAINSFORUSER configuration parameter	Specifies how the database server searches for user names in a networked Windows environment.
DISABLE_B162428_XA_FIX configuration parameter	Specifies whether to free global transactions after a rollback operation.
DRDA_COMMBUFFSIZE configuration parameter	Specifies the size of the DRDA communications buffer.
IFXGUARD configuration parameter	Enables auditing with IBM Security Guardium® and sets the actions of the database server if the IBM Security Guardium server does not respond in the timeout period.
IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter	Enables the transaction manager to use same XID to represent global transactions on different databases in the same database server instance.
LIMITNUMSESSIONS configuration parameter	Specifies the maximum number of sessions that can connect to the database server.

Configuration Parameter	Reference
MSG_DATE configuration parameter	Inserts a date stamp at the beginning of messages that are printed to the online log.
NET_IO_TIMEOUT_ALARM configuration parameter	Sends notification if network write operations are blocked for 30 minutes or more.
PN_STAGEBLOB_THRESHOLD configuration parameter	Reserves space for BYTE and TEXT data in round-robin fragments.

Related reference:
Database server files

ADMIN_MODE_USERS configuration parameter

The ADMIN_MODE_USERS configuration parameter specifies a list of users, besides the user **informix** and members of the DBSA group, that you want to access the database server in the administration mode.

onconfig.std value
: Not set. Only user informix and members of the DBSA group can access Informix® in administration mode.

separators
: Comma-separated user names, such as: `Karin,Sarah,Andrew`, as a string of up to 127 bytes

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The list of users is in the ADMIN_MODE_USERS configuration parameter is preserved indefinitely. You can use the **onmode -wm** or **onmode -wf** command to remove users.

Use the **onmode -j -U** command to allow one or more users to access the database server in administration mode when the database is running.

You must set the ADMIN_USER_MODE_WITH_DBSA configuration parameter to 1 to enable the users that are listed in the ADMIN_MODE_USERS configuration parameter to connect to the database server in the administration mode.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Changing the Database Server to Administration Mode with the -j Option
ADMIN_USER_MODE_WITH_DBSA configuration parameter
The oninit utility

ADMIN_USER_MODE_WITH_DBSA configuration parameter

The ADMIN_USER_MODE_WITH_DBSA configuration parameter specifies which users, besides the user **informix**, can connect to the database server in the administration mode.

onconfig.std value
: Not set. Only the user **informix** can connect to the database server in administration mode.

values
: `0` = Only the user **informix** can connect in the administration mode
: `1` = If the ADMIN_USER_MODE configuration parameter is not set, the following users can connect in the administration mode:

- The user **informix**
- Members of the DBSA group

If the ADMIN_USER_MODE configuration parameter is set to a list of one or more user names, then following users can connect in the administration mode:

- The user **informix**
- The users who have the **informix** group included in their group list (UNIX only)
- Members of the DBSA group
- The administration users that are listed in the ADMIN_MODE_USERS configuration parameter

takes effect
: After you edit your onconfig file and restart the database server.

Related reference:
ADMIN_MODE_USERS configuration parameter

ALARMPROGRAM configuration parameter

Use the ALARMPROGRAM configuration parameter to specify the full pathname of the alarmprogram file that handles event alarms and controls logical-log backups.

onconfig.std value
On UNIX: $INFORMIXDIR/etc/alarmprogram.sh
On Windows: %INFORMIXDIR%\etc\alarmprogram.bat

if not present
On UNIX: $INFORMIXDIR/etc/no_log.sh
On Windows: %INFORMIXDIR%\etc\no_log.bat

value
pathname = Full path name of the alarmprogram file.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

You can set the ALRM_ALL_EVENTS configuration parameter to specify whether the ALARMPROGRAM configuration parameter runs for all events that are logged in the MSGPATH, or only for specified noteworthy events (events greater than severity 1).

If the script that the ALARMPROGRAM configuration parameter specifies does not exist, the default alarm handler, no_log.sh or no_log.bat, is substituted. After you have the correct script in place, update the value of the ALARMPROGRAM configuration parameter to specify the script. You can make this update with the server online by using the **onmode -wm** command.

The following sample scripts are provided.

Table 1. Sample scripts

Script name (UNIX)	Script name (Windows)	Description
log_full.sh	log_full.bat	To back up logical logs automatically when the database server issues a log-full event alarm, set ALARMPROGRAM to log_full.sh or log_full.bat. You can modify the script and set it to the full path of ALARMPROGRAM in the onconfig file.
no_log.sh	no_log.bat	To disable automatic logical-log backups, set ALARMPROGRAM to no_log.sh or no_log.bat.
alarmprogram.sh	alarmprogram.bat	Handles event alarms and controls logical-log backups. Modify alarmprogram.sh or alarmprogram.bat and set ALARMPROGRAM to the full path name of alarmprogram.sh or alarmprogram.bat. See Customizing the ALARMPROGRAM Scripts.

Instead of using the supplied scripts, you can write your own shell script, batch file, or binary program to execute events. Set ALARMPROGRAM to the full pathname of this file. The database server executes this script when noteworthy events occur. These events include database, table, index, or simple-large-object failure; all logs are full; internal subsystem failure; initialization failure; and long transactions. You can have the events noted in an email or pagermail message.

To generate event alarms, set ALARMPROGRAM to $INFORMIXDIR/etc/alarmprogram.sh or %INFORMIXDIR%\etc\alarmprogram.bat and modify the file according.

Important: When you choose automatic logical-log backups, backup media should always be available for the backup process.
Do not use the continuous log backup command (**onbar -b -l -C**) if you have automatic log backup setup through the ALARMPROGRAM parameter.

Related concepts:
Event Alarms
Related tasks:
Customizing the ALARMPROGRAM Scripts
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Writing Your Own Alarm Script
ALRM_ALL_EVENTS configuration parameter

ALLOW_NEWLINE configuration parameter

Use the ALLOW_NEWLINE configuration parameter to allow or disallow newline characters in quoted strings for all sessions.

To allow all remote sessions in a distributed query to support embedded newline characters, specify ALLOW_NEWLINE in their onconfig files.

onconfig.std value
ALLOW_NEWLINE 0
values
0 = Disallow the newline character in quoted strings for all sessions.
1 = Allow the newline character in quoted strings for all sessions.

takes effect
After you edit your onconfig file and restart the database server.

Usage

You can specify that you want the database server to allow the newline character (**\n**) in a quoted string either for all sessions or for a specific session. A session is the duration of a client connection to the database server.

To allow or disallow newline characters in quoted strings for the current session when ALLOW_NEWLINE is not set, you can execute the built-in **ifx_allow_newline()** routine with `'t'` or `'f'` as its only argument.

- `'t'` enables support for newline characters within quoted strings.
- `'f'` has the opposite effect.

Calls to **ifx_allow_newline()** affect only the user session from which that routine is invoked.
Related information:
Quoted String
Newline characters in quoted strings

ALRM_ALL_EVENTS configuration parameter

Use the ALRM_ALL_EVENTS configuration parameter to specify whether the ALARMPROGRAM configuration parameter runs for all events that are logged in the MSGPATH configuration parameter, or only for noteworthy events.

onconfig.std value
ALRM_ALL_EVENTS 0
values
`0` = Only for noteworthy events.
`1` = The parameter triggers the ALARMPROGRAM configuration parameter and the ALRM_ALL_EVENTS configuration parameter displays all event alarms.

takes effect
After you edit your onconfig file and restart the database server.

Related concepts:
Event Alarms
Related reference:
ALARMPROGRAM configuration parameter

AUTO_AIOVPS configuration parameter

The AUTO_AIOVPS configuration parameter enables the database server to automatically increase the number of asynchronous I/O virtual processors (AIO VPs) and page cleaner threads when the database server detects that the I/O workload outpaced the performance of the existing AIO VPs.

onconfig.std value
Not set. If the AUTO_TUNE configuration parameter is set to 1, AIO VPs and page cleaner threads are automatically increased.
values
`0` = Off
`1` = On

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
If an AUTO_AIOVPS value is not set in your current onconfig file and you edit the AUTO_TUNE configuration parameter and restart the database server

Usage

The VPCLASS **aio** configuration parameter controls the number of AIO VPs, If the VP **aio** parameter is not set in the onconfig file, the initial number of AIO VPs the database server starts when AUTO_AIOVPS is enabled is equal to the number of AIO chunks. The maximum number of AIO VPs the database server can start if VP **aio** is not set is 128.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_TUNE configuration parameter
VPCLASS configuration parameter
DIRECT_IO configuration parameter (UNIX)
Related information:
Automatic checkpoints, LRU tuning, and AIO virtual processor tuning

AUTO_CKPTS configuration parameter

The AUTO_CKPTS configuration parameter allows the server to trigger checkpoints more frequently to avoid the blocking of transactions.

onconfig.std value
Not set. If the AUTO_TUNE configuration parameter is set to 1, automatic checkpoints are enabled.
values

0 = Off
1 = On

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
If an AUTO_CKPTS value is not set in your current onconfig file and you edit the AUTO_TUNE configuration parameter and restart the database server

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_TUNE configuration parameter
Related information:
Checkpoints
Automatic checkpoints, LRU tuning, and AIO virtual processor tuning

AUTO_LLOG configuration parameter

Use the AUTO_LLOG configuration parameter to automatically add logical logs in the specified dbspace to improve performance.

onconfig.std value
Not in the onconfig.std file.
default value if you created a server during installation

AUTO_LLOG 1,llog,*max_size*

The *max_size* value depends on the value of the AUTO_TUNE_SERVER_SIZE configuration parameter.
values
0 = Default. Disabled. Logical logs are not automatically added to improve performance.
1,*dbspace_name*,*max_size*

- 1 = Enabled. Logical logs are automatically added when needed to improve performance.
- *dbspace_name* = The name of the dbspace in which to add logical log files. The dbspace must have the default page size for the operating system.
- *max_size* = Optional. Default is 2048000 KB (2 GB). The maximum size, in KB, of all logical log files, including any logical log files that are not stored in the dbspace *dbspace_name*. When the maximum size is reached, the database server no longer adds logical log files to improve performance. If *max_size* is not specified, the AUTO_TUNE_SERVER_SIZE configuration parameter setting affects the maximum size. See the Usage section.

separators
Separate fields with a comma.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If you created a server during installation, the AUTO_LLOG configuration parameter is enabled automatically. A dbspace that is named **llog** is created for logical logs. The installation program sets the initial size and value of the *max_size* option of the dbspace based on the value of the AUTO_TUNE_SERVER_SIZE configuration parameter. You can change the *max_size* option by resetting the value of the AUTO_LLOG configuration parameter.

If you did not create a server during installation, you can enable the AUTO_LLOG configuration parameter to automatically add logical log files when the database server detects that adding logical log files improves performance. For optimal performance, choose a dbspace on a separate disk from the root dbspace and the physical log.

When the AUTO_LLOG configuration parameter is enabled, the database server adds logical logs when the lack of logical logs causes too high a percentage of checkpoints, blocking checkpoints, or long checkpoints.

When the maximum size of the logical log files is reached, logical log files are no longer added to improve performance. However, if the DYNAMIC_LOGS configuration parameter is enabled, logical logs are added to prevent transaction blocking. The settings of the DYNAMIC_LOGS and the AUTO_LLOG configuration parameters do not interact. Similarly, you can continue to manually add logical log files.

If the value of the *max_size* field is larger than the size of the specified dbspace, make sure that your storage pool has available space.

Example

The following setting enables the automatic addition of logical log files until size of all logical log files is 204800 KB and sets the dbspace for logical log files to **llog**:

```
AUTO_LLOG 1,llog,204800
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_TUNE_SERVER_SIZE configuration parameter
Related information:
AUTO_LLOG and its effect on logging

AUTO_TUNE_SERVER_SIZE configuration parameter

Use the AUTO_TUNE_SERVER_SIZE configuration parameter to set the sizes of memory and storage spaces to allocate based on the number of expected concurrent users.

onconfig.std value
: Not in the onconfig.std file.

Default value
: Not set.

value if you created a server during installation
: Depends on the number of users you specify in the installation program.

values
: SMALL = 1 - 100 users
: MEDIUM = 101 - 500 users
: LARGE = 501 - 1000 users
: XLARGE = more than 1000 users

takes effect
: If you create a server during installation.
: After you edit your onconfig file and restart the database server for the first time.

Usage

If you create a server during installation, you specify the number of expected users for the database server. The AUTO_TUNE_SERVER_SIZE configuration parameter is set to the corresponding size, which affects the size of the following properties:

- The size of the buffer pool.
- The maximum size of logical log files before the server stops automatically adding logical logs to improve performance
- The initial size of the following created storage spaces, which are created automatically during installation:
  - An extendable plogspace for the physical log
  - A dbspace for the logical log
  - Dbspaces for databases and tables
  - A temporary dbspace
  - An sbspace
  - A temporary sbspace

The following table shows how the value of the AUTO_TUNE_SERVER_SIZE configuration parameter affects sizes.

Table 1. Effect on memory and storage space allocations

Value	Maximum size of buffer pools (BUFFERPOOL)	Initial size of automatically created storage spaces	Maximum size of logical log files (AUTO_LLOG)
SMALL	10% of available shared memory	50 MB	200 MB
MEDIUM	20%	100 MB	500 MB
LARGE	33%	200 MB	1 GB
XLARGE	50%	500 MB	2 GB

If you did not create a server during installation, or you change the value of the AUTO_TUNE_SERVER_SIZE configuration parameter after you initialize the server for the first time, the new value affects the size of only the following properties:

- The size of the buffer pool, if the BUFFERPOOL configuration parameter setting includes the **memory='auto'** option.
- The maximum size of all logical log files before the server stops automatically adding logical logs to improve performance.

Related reference:
BUFFERPOOL configuration parameter
AUTO_LLOG configuration parameter

AUTO_LRU_TUNING configuration parameter

Use the AUTO_LRU_TUNING configuration parameter to enable automatic LRU tuning, which automatically maintains enough clean pages for page replacement.

onconfig.std value
: Not set. If the AUTO_TUNE configuration parameter is set to 1, automatic LRU tuning is enabled.

values
: 0 = Off
: 1 = On

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.
: If an AUTO_LRU_TUNING value is not set in your current onconfig file and you edit the AUTO_TUNE configuration parameter and restart the database server

Usage

Automatic LRU tuning changes affect all buffer pools and adjust the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_TUNE configuration parameter
BUFFERPOOL configuration parameter
Related information:
Automatic checkpoints, LRU tuning, and AIO virtual processor tuning

AUTO_READAHEAD configuration parameter

Use the AUTO_READAHEAD configuration parameter to change the automatic read-ahead mode or to disable automatic read-ahead operations for a query.

onconfig.std value
: Not set. If the AUTO_TUNE configuration parameter is set to 1, read ahead is performed automatically in the standard mode.

values
: An integer from 0 - 2 that specifies the mode, optionally followed by a comma and an integer that specifies the number of pages that are automatically requested to be read ahead. For example, the value `1,4096` enables automatic read-ahead in standard mode for 4096 pages at a time.
0 = Disable automatic read-ahead requests.

1 = Enable automatic read-ahead requests in the standard mode. The database server automatically processes read-ahead requests only when a query waits on I/O.

2 = Enable automatic read-ahead requests in the aggressive mode. The database server automatically processes read-ahead requests at the start of the query and continuously through the duration of the query.

number_of_pages = 4 - 4096, indicating the number of pages that are automatically requested to be read ahead. The default is 128 pages.

separators
: Separate the mode and the number of pages with a comma.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
If an AUTO_READAHEAD value is not set in your current onconfig file and you edit the AUTO_TUNE configuration parameter and restart the database server

Usage

Automatic read-ahead operations help improve query performance by issuing asynchronous page requests when the database server detects that the query is encountering I/O. Asynchronous page requests can improve query performance by overlapping query processing with the processing necessary to retrieve data from disk and put it in the buffer pool.

Generally, the default value of `1` is appropriate for most production environments.

While there are no specific circumstances in which aggressive read-ahead operations perform significantly better than standard read-ahead operations, aggressive read-ahead might be slightly more effective:

- For some scans that read a small amount of data
- In situations in which you switch between turning read-ahead off for small scans and on for longer scans
- For scans that look only at a small number of rows, because the server performs read-ahead operations immediately rather than waiting for the scan to encounter I/O.

For scans that might turn read-ahead operations off and on because the scan hits pockets of cached data, aggressive read-ahead operations do not turn off read-ahead operations.

Use aggressive read-ahead operations only in situations in which you tested both settings and know that aggressive read-ahead operations are more effective. Do not use aggressive read-ahead operations if you are not sure that they are more effective.

You can use the AUTO_READAHEAD environment option of the SET ENVIRONMENT statement of SQL to enable or disable the value of the AUTO_READAHEAD configuration parameter for a session.

The precedence of read-ahead setting is as follows:

1. A SET ENVIRONMENT AUTO_READAHEAD statement for a session.
2. The AUTO_READAHEAD configuration parameter value of `1` or `2`.
3. If the value for the AUTO_READAHEAD configuration parameter is not present in the onconfig file, the server performs read-ahead on 128 data pages (which equates to AUTO_READAHEAD mode set to 1), when the server completes a query.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_TUNE configuration parameter
Related information:
Sequential scans
Read-ahead operations
AUTO_READAHEAD session environment option

AUTO_REPREPARE configuration parameter

The AUTO_REPREPARE configuration parameter controls whether the database server automatically reoptimizes SPL routines and reprepares prepared objects after the schema of a table that is referenced by the SPL routine or by the prepared object was changed.

onconfig.std value
: Not set. If the AUTO_TUNE configuration parameter is set to 1, SPL routines are automatically reoptimized and prepared objects are automatically reprepared.

values
: `0` = Disables the automatic repreparation of prepared objects after the schema of a directly or an indirectly referenced table is modified. Also disables the automatic reoptimization of SPL routines after the schema of an indirectly referenced table is modified.
`1` = Enables automatic repreparation.

`3` = Enables automatic repreparation in optimistic mode.

`5` = Enables automatic repreparation on update statistics.

`7` = Enables automatic repreparation in optimistic mode and on update statistics.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
If an AUTO_REPREPARE value is not set in your current onconfig file and you edit the AUTO_TUNE configuration parameter and restart the database server
When you reset the value in memory by running the **onmode -wm** command.

Usage

Enable the AUTO_REPREPARE configuration parameter to reduce the number of reprepare operations that you must perform explicitly after modifying the schema of a table that is referenced by a dynamic SQL statement or a DML statement in an SPL routine.

For example, certain DDL statements modify the schema of a table, such as CREATE INDEX, DROP INDEX, DROP COLUMN, and RENAME COLUMN. If the AUTO_REPREPARE configuration parameter is disabled when these DDL statements are run, users might receive `-710` errors. These errors occur the next time that you run:

- An SPL routine that directly or indirectly references tables that were modified by the DDL statements
- A prepared object that references the tables that were modified by the DDL statements

Optimistic mode offers faster performance by not checking statements that successfully executed less than a second ago. In the unlikely event that tables were modified in the interim, some -710 errors might occur.

Set automatic repreparation on update statistics if you want to avoid the database server using an older, suboptimal execution plan.
Restriction:
Enabling AUTO_REPREPARE might have no effect on prepared statements or on SPL routines that reference tables in which DDL operations change the number of columns in the table, or change the data type of a column. After these schema changes, typically you must reissue the DESCRIBE statement, the PREPARE statement (for prepared objects), and the UPDATE STATISTICS FOR ROUTINE statement (for cursors associated with routines) for optimized execution plans of SPL routines that reference the table whose schema has been modified. Otherwise, the database server might issue SQL error `-710`.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_TUNE configuration parameter
Related information:
IFX_AUTO_REPREPARE session environment option
PREPARE statement
SET ENVIRONMENT statement
UPDATE STATISTICS statement

AUTO_STAT_MODE configuration parameter

Use the AUTO_STAT_MODE configuration parameter to enable or disable the mode for selectively updating only stale or missing data distributions in UPDATE STATISTICS operations instead of updating statistics for all data distributions.

onconfig.std value
: Not set. If the AUTO_TUNE configuration parameter is set to 1, statistics are updated selectively.

values
: `0` = Disables selective UPDATE STATISTICS operations.
`1` = Enables selective UPDATE STATISTICS operations.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
If an AUTO_STAT_MODE value is not set in your current onconfig file and you set the AUTO_TUNE configuration parameter.

Usage

When the AUTO_STAT_MODE configuration parameter or the AUTO_STAT_MODE session environment variable have enabled the automatic mode for selectively updating only stale or missing data distributions in UPDATE STATISTICS operations, the database server uses the value of the STATCHANGE configuration parameter to identify table or fragment distribution statistics that need to be updated.

In sessions where the AUTO_STAT_MODE configuration parameter and the AUTO_STAT_MODE session environment variable have different settings, the session environment variable takes precedence for the duration of that session, or until the AUTO_STAT_MODE session environment variable is reset.

Related reference:
STATCHANGE configuration parameter
AUTO_TUNE configuration parameter
Related information:
Statistics options of the CREATE TABLE statement
AUTO_STAT_MODE session environment option

AUTO_TUNE configuration parameter

Use the AUTO_TUNE configuration parameter to enable or disable all automatic tuning configuration parameters that have values that are not present in the onconfig file.

onconfig.std value
AUTO_TUNE 1
values
0 = disabled
1 = enabled
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If an individual automatic tuning configuration parameter is not set in your current onconfig file, the database server uses the value specified in the AUTO_TUNE configuration parameter for that configuration parameter.

The automatic tuning configuration parameters are:

- AUTO_AIOVPS
- AUTO_CKPTS
- AUTO_LRU_TUNING
- AUTO_READAHEAD
- AUTO_REPREPARE
- AUTO_STAT_MODE

If an automatic tuning configuration parameter is set in the current onconfig file, the database server uses the value that is in the onconfig file. The AUTO_TUNE configuration parameter does not change that value.

Your onconfig file is in the %INFORMIXDIR%\etc or $INFORMIXDIR/etc directory.

Examples

Example 1: Suppose some of your automatic tuning configuration parameters are not set, but others have values:

- AUTO_LRU_TUNING (value not set)
- AUTO_STAT_MODE (value not set)
- AUTO_LRU_CKPTS (value not set)
- AUTO_AIOVPS 0
- AUTO_REPREPARE 1
- AUTO_READAHEAD 0

If you set the AUTO_TUNE configuration parameter to 1, the database server automatically changes the values that are not set to 1. The values that were previously set remain the same. The automatic tuning configuration parameters now have the following values:

- AUTO_LRU_TUNING 1
- AUTO_STAT_MODE 1
- AUTO_CKPTS 1
- AUTO_AIOVPS 0
- AUTO_REPREPARE 1
- AUTO_READAHEAD 0

Example 2: Suppose all of your automatic tuning configuration parameters are set and have the following values:

- AUTO_LRU_TUNING 1
- AUTO_STAT_MODE 1
- AUTO_LRU_CKPTS 1
- AUTO_AIOVPS 0

- AUTO_REPREPARE 1
- AUTO_READAHEAD 0

In this situation, the AUTO_TUNE configuration does not change any of the values.

Example 3: Suppose that you removed the automatic tuning configuration parameters from your onconfig file but now want to use them. You can set AUTO_TUNE to 1 to re-enable all of the automatic tuning configuration parameters.

Related reference:
AUTO_AIOVPS configuration parameter
AUTO_CKPTS configuration parameter
AUTO_LRU_TUNING configuration parameter
AUTO_REPREPARE configuration parameter
AUTO_STAT_MODE configuration parameter
AUTO_READAHEAD configuration parameter

AUTOLOCATE configuration parameter

Use the AUTOLOCATE configuration parameter to enable the automatic location of databases, indexes, and tables, and the automatic fragmentation of tables.

onconfig.std and default value
AUTOLOCATE 0
values
`0` = Disable automatic location and fragmentation.
`1 - 32` = Enable automatic location and fragmentation. The number indicates how many round-robin fragments to initially allocate to a table.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in memory and in your onconfig file by running the **onmode -wf** command.
When you reset the value dynamically in memory by running the **onmode -wm** command.

Usage

Use the AUTOLOCATE configuration parameter to control whether the database server controls the location of new databases, indexes, and tables and the fragmentation of those tables. If you set the AUTOLOCATE configuration parameter to a positive integer, the database server performs the following tasks:

- Stores new databases for which you do not specify a location in the optimal dbspace instead of in the root dbspace. By default, all dbspaces except dbspaces that are dedicated to tenant databases are available. However, you can control the list of available dbspaces.
- Fragments new tables by round-robin, where the number of fragments is equal to the value of the AUTOLOCATE configuration parameter.
- Adds more table fragments as the table grows.

If you set the value of the AUTOLOCATE configuration parameter to 0, new databases are created in the root dbspace by default. New tables and indexes are created in the same dbspace as the database and are not fragmented.

Automatic location is not applicable to tenant databases or the tables, fragments, and indexes within tenant databases.

You can override the automatic location of a database by specifying a dbspace with the IN clause in the CREATE DATABASE statement. Similarly, you can override the automatic location and fragmentation of a table by specifying a dbspace with the IN clause or a fragmentation strategy with the FRAGMENT BY clause in the CREATE TABLE statement.

When this configuration parameter is enabled, you can use the autolocate database arguments with the **admin()** or **task()** function to:

- Manage the list of dbspaces for automatic location and fragmentation. The list of available dbspaces is in the **sysautolocate** system catalog table.
- Disable automatic location and fragmentation for the specified database.

You can use the AUTOLOCATE environment option of the SET ENVIRONMENT statement of SQL to enable or disable the value of the AUTOLOCATE configuration parameter for a session.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)
autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)
Related information:
AUTOLOCATE session environment option
Managing automatic location and fragmentation

BATCHEDREAD_INDEX configuration parameter

Use the BATCHEDREAD_INDEX configuration parameter to enable the optimizer to execute light scans for indexes. This reduces the number of times that a buffer is read, thus improving performance.

onconfig.std value
BATCHEDREAD_INDEX 1
values
0 = Disable light scans for indexes.
1 = Enable light scans for indexes.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

In sessions where the IFX_BATCHEDREAD_INDEX configuration parameter and the IFX_BATCHEDREAD_INDEX session environment variable have different settings, the session environment variable takes precedence for the duration of that session, or until the IFX_BATCHEDREAD_INDEX session environment variable is reset.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

BATCHEDREAD_TABLE configuration parameter

Use the BATCHEDREAD_TABLE configuration parameter to enable or disable light scans on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data.

onconfig.std value
BATCHEDREAD_TABLE 1
values
0 = Disable light scans on variable record-length tables
1 = Enable light scans on variable record-length tables.

Compressed tables, and tables with rows longer than a page, are treated here as of variable record-length.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Except for compressed tables, tables with rows that are larger than a page, and tables of varying record length (such as VARCHAR, LVARCHAR, and NVARCHAR columns), the setting of BATCHEDREAD_TABLE has no effect on whether the query optimizer chooses a query execution path that includes a light scan.

The database server does not perform light scans on indexes, on system tables, nor on user tables whose rows include large objects with any of these storage attributes:

- blobspaces
- smartblob spaces
- partition blob.

You can use the IFX_BATCHEDREAD_TABLE environment option of the SET ENVIRONMENT statement to override the value of the BATCHEDREAD_TABLE configuration parameter for the current session.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
SET ENVIRONMENT statement
Light scans

BLOCKTIMEOUT configuration parameter

Use the BLOCKTIMEOUT configuration parameter to specify the number of seconds that a thread or database server will hang. After the timeout, the thread or database server will either continue processing or fail.

onconfig.std value
BLOCKTIMEOUT 3600
units
Seconds
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

BTSCANNER Configuration Parameter

Use the BTSCANNER configuration parameter to set the B-tree scanner. The B-tree scanner improves transaction processing for logged databases when rows are deleted from a table with indexes. The B-tree scanner threads remove deleted index entries and rebalance the index nodes. The B-tree scanner automatically determines which index items are to be deleted.

onconfig.std value
: BTSCANNER num=1,threshold=5000,rangesize=-1,alice=6,compression=default

range of values
: See the Usage section.

separators
: Use a comma between each field.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -C** command.
: After you run the SQL administration API **task()** or **admin()** function with the **onmode** and **C** arguments.

Usage

By default, the BTSCANNER configuration parameter starts one index cleaner thread, prioritizes cleaning indexes that have over 5000 deleted items, automatically adjusts the mode of index cleaning, and merges index pages at a level appropriate for indexes that have moderate growth and changes.

Syntax for the BTSCANNER configuration parameter

```
>>-BTSCANNER--+------------------------+------------------------>
              |         .-1-------.    |
              '-num--=--+-threads-+--,-'

>--+-----------------------------+--+----------------------+--->
   '-threshold--=--thresh_size--,-'  '-rangesize--=--100--,-'

>--+-------------------------+---------------------------------->
   '-alice--=--alice_mode--,-'

>--+-----------------------------+----------------------------><
   |                 .-default-. |
   '-compression--=--+-low-----+-'
                     +-med-----+
                     '-high----'
```

Table 1. Options for the BTSCANNER configuration parameter value

Field	Values
num	The *threads* value is a positive integer that sets the number of B-tree scanner threads to start at system startup. The default is `1`.
threshold	The *thresh_size* value is the minimum number of deleted items an index must encounter before an index is prioritized for cleaning. The default is `5000`.
rangesize	Specifies whether to allow leaf scans for small indexes: • -1 = Off. The alice mode is used for all index cleaning. • 100 = Small indexes are scanned by the leaf scan method.
alice	The *alice_mode* value controls index cleaning: • 0 = Off. • 1 = Uses exactly 8 bytes of memory. • 2 = Uses exactly 16 bytes of memory. • 3 - 12 = Default is 6. Sets the initial amount of memory that is used for index cleaning. Subsequently, the B-tree scanners automatically adjust the mode based on the efficiency of past cleaning operations.
compression	The level at which two partially used index pages are merged: • `low` = Use if you expect an index to grow quickly with frequent splits. • `med` or `default` = Default. Use if an index has moderate growth or changes. • `high` = Use if an index is 90 percent or more read-only or does not have many changes.

After all of the indexes above the threshold are cleaned, the indexes below the threshold are added to the prioritized list of indexes to be cleaned. Systems updated frequently should increase this value by a factor of 10 times or 100 times.

Related reference:
onmode -C: Control the B-tree scanner
onmode and C arguments: Control the B-tree scanner (SQL administration API)
Related information:
Configure B-tree scanner information to improve transaction processing

BUFFERPOOL configuration parameter

Use the BUFFERPOOL configuration parameter to configure how many data pages are cached in shared memory and how often those pages are flushed to disk between checkpoints. The default values of the BUFFERPOOL configuration parameter are adequate for many systems. However, you can change the values to tune the performance of your system.

onconfig.std values
: Operating systems with 2 KB default page size:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,
lru_max_dirty=60.50
BUFFERPOOL size=2k,buffers=50000,lrus=8,lru_min_dirty=50,
lru_max_dirty=60
```

: Operating systems with 4 KB default page size:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,
lru_max_dirty=60.50
BUFFERPOOL size=4k,buffers=10000,lrus=8,lru_min_dirty=50,
lru_max_dirty=60
```

default value if you created a server during installation

```
BUFFERPOOL default,memory='auto'
BUFFERPOOL size=page_size,memory=memory_size
```

: The *page_size* value is the default page size. The initial size of the buffer pool is 32 MB. The maximum size, which is specified by the value of the **memory** field as either **auto** or the *memory_size* value, depends on the value of the AUTO_TUNE_SERVER_SIZE configuration parameter.

values
: See the Usage section.

separators
: Separate fields with a comma.

takes effect
: After you edit your onconfig file and restart the database server.
: When you add an entry dynamically in your onconfig file by running the **onparams -b** command.
: When you add an entry dynamically by adding a dbspace with a different page size by running the **onspaces -c -d** command.
: After you add an entry dynamically in your onconfig file by running the SQL administration API **task()** or **admin()** function with the **add bufferpool** argument.

Usage

Cached data pages are held in buffers. Buffers are contained in buffer pools. You need a buffer pool for each page size that you use for storage spaces. When the database server moves new data pages into shared memory, data pages that are the least-recently used are moved out of shared memory. The BUFFERPOOL configuration parameter controls the size of the buffer pool and how frequently data pages are flushed to disk.

The BUFFERPOOL configuration parameter has two entries in the onconfig.std file or in the onconfig file that was generated if you created a server during installation:

- The first entry specifies the default values for a buffer pool for a dbspace with a non-default page size.
- The second entry specifies the default values for a buffer pool that is based on the default page size of the system.

The BUFFERPOOL configuration parameter entries that include the **size** field take precedence over the entry that includes the **default** field.

The BUFFERPOOL configuration parameter has two formats:

- Use the BUFFERPOOL configuration parameter with the **memory** field if you want to specify the size of your buffer pool in units of memory like MB or GB.
- Use the BUFFERPOOL configuration parameter with the **buffers** field if you want to specify the size of your buffer pool in units of pages, or to retain settings from a previous release.

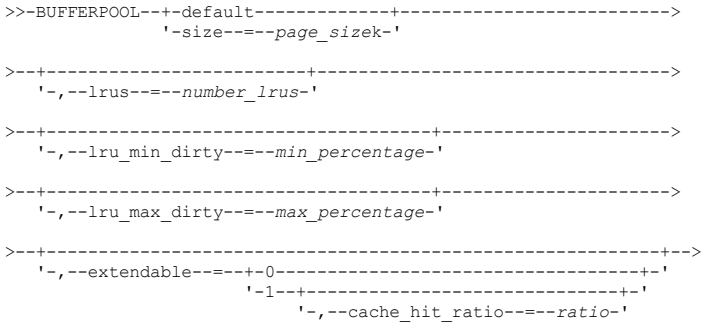
You can use either format to enable the database server to expand the size of the buffer pool as needed to improve performance.

Restriction: You cannot combine formats in the onconfig file. All entries for the BUFFERPOOL configuration parameter in the onconfig file must have the same format or the database server does not start and the following error shows:

```
ERROR: Cannot mix buffer arguments with memory arguments.  (BUFFERPOOL)
```

The fields in the BUFFERPOOL entries are not case-sensitive and the fields can be listed in any order.

Syntax with the memory field



```
>--+-------------------------------------------+--------------->
   '-,--start_memory--=--+-auto--------------+-'
                         '-start_size--+----+-'
                                       +-kb-+
                                       +-mb-+
                                       '-gb-'

>--,--memory--=--+-auto-------------+-------------------------><
                 '-max_size--+----+-'
                             +-kb-+
                             +-mb-+
                             '-gb-'


Syntax with the buffers field

>>-BUFFERPOOL--+-default-------------------+-------------------->
               '-size--=--page_size--+---+-'
                                     '-k-'

>--+-------------------------+---------------------------------->
   '-,--lrus--=--number_lrus-'

>--+-----------------------------------+---------------------->
   '-,--lru_min_dirty--=--min_percentage-'

>--+-----------------------------------+---------------------->
   '-,--lru_max_dirty--=--max_percentage-'

>--,--buffers--=--number_buffers-------------------------------->

>--+-------------------------------------------------------+-----><
   '-,--extendable--=--+-0---------------------------+-'
                       '-1--+-------------------------+-'
                            '-| extendable options |-'

extendable options

|--+---------------------------+------------------------------->
   '-,--max_extends--=--extends-'

>--+-----------------------------------+---------------------->
   '-,--next_buffers--=--number_buffers-'

>--+-------------------------------+----------------------------|
   '-,--cache_hit_ratio--=--ratio-'
```

Table 1. Options for the BUFFERPOOL configuration parameter value

Field	Values
buffers	Default is 1000. The *number_buffers* value is an integer >= 1000 that specifies the maximum number of shared-memory buffers. The maximum allowed number of buffers depends on the operating system, the bit size, and the page size: • UNIX, 32-bit, with a 2 KB page size: 1000 - 1843200 • UNIX, 32-bit, with a 4 KB page size: 1000 - 921600 • Windows, 32-bit: 100 - 524288 • 64-bit: 100 - (2^{31}-1). For the actual value for your 64-bit platform, see your machine notes. For example, the maximum number of buffers on the Solaris platform is 536,870,912. Set the value of the **buffers** field to at least four buffers per user. If your system handles more than 500 concurrent users, specify at least 2000 buffers. Each buffer is the size of the operating system page. Therefore, the number of buffers that the database server requires depends on the amount of physical memory and how much memory is used by applications. For example, if the database server accesses 15 percent of the application data 90 percent of the time, allocate enough buffers to hold 15 percent of the data. Increasing the number of buffers can improve system performance. The number of buffers can have a significant affect on performance and use a large percentage of physical memory. For more information, see The BUFFERPOOL configuration parameter and memory utilization.
cache_hit_ratio	Default is 90. The *ratio* value is an integer 0 - 100 that represents the threshold below which the buffer pool is extended. When the average read cache hit ratio remains below the value of *ratio* for approximately five minutes, the database server extends the buffer pool. The **cache_hit_ratio** field is valid only if **extendable=1** is set.
extendable	Default is 1 if the **memory** field is set. Default is 0 if the **buffers** field is set. Whether the database server can extend the size of the buffer pool: • 0 = Disabled. The buffer pool cannot grow. • 1 = Enabled. The buffer pool can grow.

Field	Values
lru_max_dirty	Default is 60.00. The *max_percentage* value is a decimal number 0 - 100.00 that sets the percentage of modified pages in the LRU queues at which the queue is cleaned. This value is updated automatically as needed if the AUTO_LRU_TUNING configuration parameter is enabled.
lru_min_dirty	Default is 50.00. The *min_percentage* value is a decimal number 0 - 100.00 that sets the percentage of modified pages in the LRU queues at which page cleaning is no longer mandatory. Page cleaners might continue cleaning beyond the specified percentage under some circumstances. This value is updated automatically as needed if the AUTO_LRU_TUNING configuration parameter is enabled.
lrus	Default is 8. If the MULTIPROCESSOR configuration parameter is enabled, the default is the greater of 8 or the number of CPU VPs. The *number_lrus* value is a positive integer that specifies the number of LRU (least recently used) queues in the buffer pool. The range of values depends on the bit size of the operating system: • 32-bit platforms: 8 - 128 • 64-bit platforms: 8 - 512 The more LRU queues that you specify, the more page cleaners work in parallel. However, setting the value of **lrus** field too high might result in excessive page-cleaner activity. The value of **lrus** field, in combination with the **lru_min_dirty** and **lru_max_dirty** fields control how frequently the shared-memory buffers are flushed to disk. For more information, see BUFFERPOOL and its effect on page cleaning.
max_extends	Default is 8. The *extends* value represents the maximum number of times that the database server can extend the buffer pool. The value of *extends* is 0 through the maximum number of segments, which depends on the operating system and bit size: • 32 bit = 16 • UNIX 64 bit = 24 • Windows 64 bit = 8 The **max_extends** field is valid only if **buffers** and **extendable=1** are set.
memory	Default is **auto**. The *max_size* value represents the maximum size of the buffer pool. The range of values for *max_size* is: • An integer that represents 32 MB - 4 TB. You can specify the size units of KB, MB, or GB. If you do not specify units, the default units are KB. • **auto** = The database server determines the maximum amount of shared memory to allocate to the buffer pool. The value of the AUTO_TUNE_SERVER_SIZE configuration parameter, if it is set, controls the maximum size of the buffer pool.
next_buffers	Default is 1000. The *number_buffers* value is an integer >= 1000 that specifies the number of shared-memory buffers by which the database server extends the buffer pool. The maximum value of *number_buffers* is limited by the amount of virtual shared memory. The *number_buffers* value is doubled every four extensions. The **next_buffers** field is valid only if **buffers** and **extendable=1** are set.
size	The *page_size* value specifies the page size for buffers, in KB. The page size must be 2 - 16 KB and must be a multiple of the default page size. For example, if the default page size is 2 KB, the page size can be 2, 4, 6, 8, 10, 12, 14, or 16. If the default page size is 4 KB, the page size can be 4, 8, 12, or 16. The default value depends on the system default page size: • 2 KB default page size: `size=2k` • 4 KB default page size: `size=4k` The `k` is optional.
start_memory	Default is 32 MB. The *start_size* value represents the initial size of the buffer pool when the database server starts: • An integer that represents 32 MB through the maximum amount of shared memory that is available. You can specify the size units of KB, MB, or GB. If you do not specify units, the default units are KB. The initial size of the buffer pool might be larger than the value of *start_size* because the size must be a multiple of the size of a shared memory segment. • **auto** = The database server determines the initial amount of shared memory to allocate to the buffer pool. If you do not set the **start_memory** field, the initial size of the buffer pool is equal to the value of the **memory** field. The **start_memory** field is valid only if the **memory** field is set.

The size of the buffer pool with the memory format

If you use the memory format, by default the buffer pool grows in size as needed. Shared memory segments are added to the buffer pool when the average cache read hit ratio is under the threshold. You can set the initial and maximum size of the buffer pool or allow the database server to determine the optimal sizes.

If the **extendable** field is set to 0, the buffer pool does not grow. The size is equal to the value of the **start_memory** field, if it is set, otherwise, the value of the **memory** field.

When you restart the server, the size of the buffer pool is reset to the value of the **start_memory** field.

The size of the buffer pool with the buffers format

If you use the **buffers** format, by default the buffer pool does not grow in size. The size is equal to the value of the **buffers** field.

If you set the **extendable** field to 1, shared memory segments are added to the buffer pool when the average cache read hit ratio is under the threshold. You must set the initial number of buffers in the **buffers** field. You can optionally set the number of buffers by which to extend the buffer pool, and the maximum number of times that the buffer pool can be extended, and the cache hit ratio. The number of buffers that are added to the buffer pool doubles every fourth extension.

Example: Adding a BUFFERPOOL entry with the memory field

The following entry creates a buffer pool that has a 10 KB page size:

```
BUFFERPOOL size=10k,start_memory=auto,memory=4gb
```

The buffer pool is extendable up to 4 GB. The database server determines the initial size of the buffer pool and the sizes of extensions to the buffer pool.

Example: Adding a BUFFERPOOL entry with the buffers field

The following entry creates a buffer pool that has a 2 KB page size:

```
BUFFERPOOL size=2k,extendable=1,buffers=1000,next_buffers=2000,max_extends=8
```

The buffer pool is extendable eight times. The buffer pool starts with 1000 buffers. The first three extensions to the buffer pool add 2000 buffers. The fourth through seventh extensions add 4000 buffers. The eighth extension adds 8000 buffers.

Example: Adding a BUFFERPOOL entry by adding a dbspace with a different page size

When you add a dbspace with a different page size with the **onspaces** utility, or when you add a buffer pool with the **onparams** utility, a BUFFERPOOL configuration parameter entry is added in the onconfig file. The following example shows a third entry:

```
BUFFERPOOL default,buffers=10000,lrus=8,lru_min_dirty=50.00,lru_max_dirty=60.50
BUFFERPOOL size=2k,buffers=10000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
BUFFERPOOL size=6k
```

When you create a dbspace with a non-default page size, the database server uses the existing BUFFERPOOL entry for that page size, if that entry exists. Otherwise, the database server uses the values from the BUFFERPOOL default line.

Related reference:
onparams -b: Add a buffer pool
add bufferpool argument: Add a buffer pool (SQL administration API)
AUTO_LRU_TUNING configuration parameter
AUTO_TUNE_SERVER_SIZE configuration parameter
onstat -g buf command: Print buffer pool profile information
Related information:
The BUFFERPOOL configuration parameter and memory utilization
BUFFERPOOL and its effect on page cleaning
Buffer pool portion of shared memory
FIFO/LRU queues

CHECKALLDOMAINSFORUSER configuration parameter

Use the CHECKALLDOMAINSFORUSER configuration parameter to check all of the domains for all users.

onconfig.std value
: Not in the onconfig.std file

values
: 0 = Disabled
: 1 = Enabled

takes effect
: After you edit your onconfig file and restart the database server.

Related information:
Windows network domain

CKPTINTVL configuration parameter

Use the CKPTINTVL configuration parameter to specify the frequency, expressed in seconds, at which the database server checks to determine whether a checkpoint is needed. When a checkpoint occurs, all pages in the shared-memory buffer pool are written to disk.

onconfig.std value
: CKPTINTVL 300

values
: Any value greater than or equal to 0

units
: Seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The RTO_SERVER_RESTART and CKPTINTVL configuration parameters are mutually exclusive. If the RTO_SERVER_RESTART configuration parameter is enabled, it will trigger checkpoints and CKPTINTVL values are ignored. Otherwise, CKPTINTVL values are used to trigger checkpoints.

If you set the CKPTINTVL configuration parameter to an interval that is too short, the system spends too much time performing checkpoints, and the performance of other work suffers. If you set the CKPTINTVL configuration parameter to an interval that is too long, fast recovery might take too long.

In practice, 30 seconds is the smallest interval that the database server checks. If you specify a checkpoint interval of 0, the database server does not check if the checkpoint interval has elapsed. However, the database server still performs checkpoints. Other conditions, such as the physical log becoming 75 percent full, also cause the database server to perform checkpoints.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
RTO_SERVER_RESTART configuration parameter
Related information:
Checkpoints
Performance Guide

CLEANERS configuration parameter

Use the CLEANERS configuration parameter to specify the number of page-cleaner threads available during the database server operation. By default, the database server always runs one page-cleaner thread. A general guideline is one page cleaner per disk drive. The value specified has no effect on the size of shared memory.

Based on the server work load, the server automatically attempts to optimize AIO VPs and page-cleaner threads and adjust the number of AIO VPs and page-cleaner threads upward when needed. Automatic AIO VP and page-cleaner thread tuning can be disabled using the environmental variable IFX_NO_AIOVP_TUNING or the **onmode -wm** utility option.

onconfig.std value
: CLEANERS 8

values
: 1 - 128

units
: Number of page-cleaner threads

takes effect
: After you edit your onconfig file and restart the database server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -F command: Print counts
Related information:
Flush data to disk

CLUSTER_TXN_SCOPE configuration parameter

Set the CLUSTER_TXN_SCOPE configuration parameter to configure your high-availability cluster so that when a client session issues a commit, the server blocks the session until the transaction is applied in that session, on a secondary server, or across the cluster.

onconfig.std value
: CLUSTER_TXN_SCOPE SERVER

values

- `SESSION` = When a client session issues a commit, the database server blocks the session until the effects of the transaction commit are returned to that session. After control is returned to the session, other sessions at the same database server or on other database servers in the cluster might be unaware of the transaction commit and the transaction's effects.
- `SERVER` (default behavior) = When a client session issues a commit, the database server blocks the session until the transaction is applied at the database server from which the client session issued the commit. Other sessions at that database server are aware of the transaction commit and the transaction's

effects. Sessions at other database servers in the cluster might be unaware of the transaction's commit and its effects. This behavior is default for high-availability cluster servers.
- `CLUSTER` = When a client session issues a commit, the database server blocks the session until the transaction is applied at all database servers in the high-availability cluster, excluding RS secondary servers that are using DELAY_APPLY or STOP_APPLY. Other sessions at any database server in the high-availability cluster, excluding RS secondary servers that are using DELAY_APPLY or STOP_APPLY, are aware of the transaction commit and the transaction's effects.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
After you run the SQL administration API **task()** or **admin()** function with the **-wf CLUSTER_TXN_SCOPE=***value* or **-wm CLUSTER_TXN_SCOPE=***value* arguments.

Usage

Set the CLUSTER_TXN_SCOPE configuration parameter to control transaction-commit returns from a high-availability cluster to client applications. Cluster transaction coordination can delay the returning of a transaction commit to a client application until the transaction is applied to a secondary-server or all secondary servers in a high-availability cluster. This process prevents operation failures due to asynchronous log processing, and ensures that the steps of multistep processes occur in serial order.

Cluster transaction coordination does not apply to RS secondary servers that have a DELAY_APPLY or STOP_APPLY configuration parameter value other than `0`. Transactions do not need to be applied on the RS secondary servers before client applications can receive commits.

CLUSTER_TXN_SCOPE affects sessions on read-only secondary servers and updatable secondary servers.

Before IBM® Informix® version 11.70.xC6, high-availability cluster servers had the following default behaviors:

- Primary servers had a cluster transaction scope of SERVER.
- Read-only secondary servers were in the dirty-read isolation level, and could read uncommitted data.
- Updatable secondary servers had a cluster transaction scope of SESSION.

Example 1: Transactions coordination between high-availability cluster servers

In this example, a client application starts a two-step process. The client application inserts data on the primary database server, and then starts processing of the data on an HDR secondary server.

If a SELECT on the inserted data is attempted on the HDR secondary server before the logs from the primary server are applied on the HDR secondary server, the operation fails. To prevent this failure, set the primary server's CLUSTER_TXN_SCOPE configuration parameter to `CLUSTER`, so that the client application does not receive a commit, and cannot start data processing, until the data insertion is also applied on the HDR secondary server.

Example 2: Transaction coordination on a database server

In this example, you have a client application that is divided into several stages of processing. Each stage of processing uses a different SQL session to connect to the database server. The application updates data, and then another part of the application processes the updated data in a different SQL session.

If CLUSTER_TXN_SCOPE is set to SESSION, the part of the application that processes the updated data might not be aware of an update's results and a failure can occur. To prevent this failure, set the database server's CLUSTER_TXN_SCOPE configuration parameter to `SERVER`, so that the client application does not receive a commit, and cannot start data processing until the update completes on the database server.

Related reference:
DELAY_APPLY Configuration Parameter
STOP_APPLY configuration parameter
Related information:
SET ENVIRONMENT statement
CLUSTER_TXN_SCOPE session environment option
Cluster transaction coordination

CONSOLE configuration parameter

Use the CONSOLE configuration parameter to specify the path and name for console-message file.

onconfig.std values
On UNIX: $INFORMIXDIR/tmp/online.con
On Windows: online.con

values
pathname = Full path name of the online.con file.
takes effect
After you edit your onconfig file and restart the database server.

CONVERSION_GUARD configuration parameter

Use the CONVERSION_GUARD configuration parameter to specify whether IBM® Informix® stops or continues an upgrade to a new version of the server if an error occurs during the upgrade process.

onconfig.std value
: CONVERSION_GUARD 2

values
: 0 = Disabled.
1 = Enable a restore point as part of the upgrade process, and stop the upgrade if an error related to capturing restore point data occurs.

2 = Enable a restore point as part of the upgrade process, and continue the upgrade even if an error related to capturing restore point data occurs.

units
: Integer

takes effect
: When the database server is restarted

Usage

By default:

- The CONVERSION_GUARD configuration parameter is on (set to 2). If an upgrade to the new version of the server fails, you can use the **onrestorept** utility to restore your data.
- The server stores the restore point data in the $INFORMIXDIR/tmp directory.

If the CONVERSION_GUARD configuration parameter is set to 1 or 2 and the upgrade to the new version of the server fails, you can use the **onrestorept** utility to restore your data.

If the CONVERSION_GUARD configuration parameter is set to 2 and conversion guard operations fail (for example, because the server has insufficient space to store restore point data), and the upgrade fails, you cannot use the **onrestorept** utility to restore your data.

You can change the value of the CONVERSION_GUARD configuration parameter or change the directory specified in the RESTORE_POINT_DIR configuration parameter before starting the server that initiates an upgrade to a new version of the server. You cannot change the CONVERSION_GUARD or RESTORE_POINT_DIR values during an upgrade.

Related reference:
RESTORE_POINT_DIR configuration parameter
Related information:
The onrestorept utility

DATASKIP Configuration Parameter

Use the DATASKIP configuration parameter to control whether the database server skips a dbspace that is unavailable during the processing of a transaction.

onconfig.std value
: Not set. No dbspaces are skipped.

values
: See the Usage section.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onspaces -f** command.
After you run the SQL administration API **task()** or **admin()** function with the **set dataskip** argument.

Usage

Whenever the database server skips over a dbspace during query processing, a warning is returned.

Enable the DATASKIP configuration parameter with caution because the results are always suspect. Only enable the parameter in the following situations:

- You can accept the compromised integrity of transactions.
- You can determine that the integrity of the transaction is not compromised, which can be difficult and time consuming.

Syntax for the DATASKIP configuration parameter

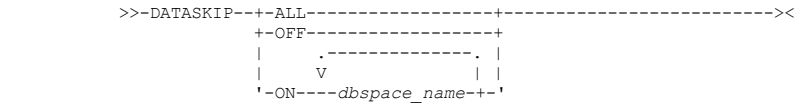


Table 1. Options for the DATASKIP configuration parameter value

Field	Description
ALL	Skip all unavailable fragments.
OFF	All fragments, including unavailable fragments, are processed.

Field	Description
ON	The *dbspace_name* value specifies one or more dbspaces to skip, separated by commas.

An application can use the SQL statement SET DATASKIP to override the value of the DATASKIP configuration parameter.

The previously reserved SQLCA warning flag **sqlwarn.sqlwarn7** is set to `W` for IBM® Informix® ESQL/C.

Related reference:
onspaces -f: Specify DATASKIP parameter
onstat -f command: Print dbspace information affected by dataskip
set dataskip argument: Start or stop skipping a dbspace (SQL administration API)
Related information:
How DATASKIP affects table I/O
SET DATASKIP statement

DBCREATE_PERMISSION configuration parameter

Use the DBCREATE_PERMISSION configuration parameter to restrict the permission to create databases to the user that you specify.

The **informix** user always has permission to create databases. To restrict the ability to create databases to the **informix** user, set the DBCREATE_PERMISSION configuration parameter to **informix**.

onconfig.std value
: On UNIX: Not set. Any user can create databases.
On Windows: #DBCREATE_PERMISSION informix

default value
: Any user can create databases.

units
: user names

separator
: Comma. You can also include multiple copies of the DBCREATE_PERMISSION configuration parameter in the onconfig file to give more users permission to create databases.

takes effect
: After you edit your onconfig file and restart the database server.

The DBCREATE_PERMISSION configuration parameter does not provide permissions to create tenant databases. Users must have the TENANT privilege to create tenant databases. Grant the TENANT privilege by running the **admin()** or **task()** SQL administration API function with the grant admin argument.

Related reference:
grant admin argument: Grant privileges to run SQL administration API commands

DB_LIBRARY_PATH configuration parameter

Use the DB_LIBRARY_PATH configuration parameter to specify a comma-separated list of valid directory prefix locations from which the database server can load external modules, such as DataBlade modules. You can also include server environment variables, such as $INFORMIXDIR, in the list.

You must specify the paths to the external modules exactly as the paths are registered with the database server. Relative paths or paths that include double periods (..) are not valid. External modules in the file systems that are not specified by this parameter cannot be loaded. This list is scanned prior to loading C language modules.

If you set this configuration parameter, you must also include the string `$INFORMIXDIR/extend` as part of the value. If the string `$INFORMIXDIR/extend` is not included in DB_LIBRARY_PATH, built-in extensions, DataBlade modules, and the BladeManager utility do not load.

onconfig.std value
: Not set

if not present
: The database server can load external modules from any location

values
: List of path names (up to 512 bytes)

separators
: Comma

takes effect
: After you edit your onconfig file and restart the database server.

DBSERVERALIASES configuration parameter

Use the DBSERVERALIASES configuration parameter to specify an alias name, or a list of unique alias names for the database server. Each alias defined by the DBSERVERALIASES configuration parameter can be used in a different connection, as specified by entries in the sqlhosts information.

onconfig.std value
: Not set. No aliases are defined.

values
: One to 32 alias names, separated by commas. Each alias name can be optionally followed by a minus sign and an integer from 1 - 50 that specifies the number of multiple listener threads to use for the **onimcsoc** or **onsoctcp** protocols. For example, the following two alias names each have four listener threads: `alias_a-4,alias_b-4`. The listener thread number is ignored for other protocols.
The maximum length of an alias is 128 bytes. Additional aliases beyond 32 are ignored. The maximum length of a DBSERVERALIASES entry is 512 bytes. You can include multiple lines of DBSERVERALIASES configuration parameters in the onconfig file.
An alias name must begin with a letter and can include any printable character, except the following:

- Uppercase characters
- A field delimiter (blank space or tab)
- A newline character
- A comment character (#)
- A hyphen or minus (= ASCII 45) character
- The @ character
- A blank space

separators
: Separate entries with a comma. Do not include blank spaces.

takes effect
: After you edit your onconfig file and restart the database server and update the sqlhosts information of each database server.

Usage

You can use the DBSERVERALIASES configuration parameter to specify aliases for both Secure Sockets Layer (SSL) and for non-SSL connection protocols.

If Informix® supports more than one communication protocol (for example, both an IPC mechanism and the TCP network protocol), you must describe each valid connection to the database server with an entry in the sqlhosts information. For example, suppose you have a server that has the name `sanfrancisco` defined by the DBSERVERNAME configuration parameter setting, and you set a DBSERVERALIASES value of `menlo` for an SSL connection. You must specify information for both of the `sanfrancisco` and `menlo` servers in the sqlhosts information. Similarly, if the database server needs to support both the standard Informix protocols and the Distributed Relational Database Architecture™ (DRDA) protocols, assign an alias to the DRDA database server and add an entry for this alias in the sqlhosts file.

For each alias listed in the DBSERVERALIASES configuration parameter, the database server starts an additional listener thread. If you have many client applications connecting to the database server, you can distribute the connection requests between several listener threads and speed connection times. To take advantage of the alternate connections, program some of your client applications to connect to a database server alias name instead of the database server name.

If you use Informix MaxConnect with more than one communication protocol, specify additional database server aliases for the DBSERVERALIASES configuration parameter. The value of the INFORMIXSERVER environment variable on the client must match either the value of the DBSERVERNAME configuration parameter or one of the values of the DBSERVERALIASES configuration parameter.

High-availability cluster servers that use shared-memory connections must also have TCP connection aliases for server-to-server communication. If a high-availability cluster server's DBSERVERNAME is associated with a shared-memory sqlhosts file entry, you must create a TCP alias for the server by setting a DBSERVERALIASES value, setting the HA_ALIAS configuration parameter to the DBSERVERALIASES value, and then creating a TCP sqlhost file entry for the alias.

Note: Service name used for loopback replication group should be added to DBSERVERALIAS list, and it should appear after service name used for primary Enterprise Replication group.

Related reference:
DBSERVERNAME configuration parameter
HA_ALIAS configuration parameter
NETTYPE configuration parameter
NUMFDSERVERS configuration parameter
onmode -d: Set data-replication types
onmode -d: Set High Availability server characteristics
onmode -d command: Replicate an index with data-replication

Related information:
Configuration parameters related to connectivity
Multiple connection types
Add listen threads

DBSERVERNAME configuration parameter

Use the DBSERVERNAME configuration parameter to specify a unique name that you want to associate with the database server. You specify this configuration parameter when you install the database server.

onconfig.std value
: Not set. A database server name is not defined.

if not present
: On UNIX: *hostname*
On Windows: ol_*hostname*

The *hostname* variable is the name of the host computer.

values
: A database server name that has a maximum length of 128 bytes.The database server name can be optionally followed by a minus sign and an integer from 1 - 50 that specifies the number of multiple listener threads to use for the **onimcsoc** or **onsoctcp** protocols. The default number of listener threads is 1. For example, the following database server name has four listener threads: `ifxserver-4`. The listener thread number is ignored for other protocols.
A database server name must begin with a letter and can include any printable character, except the following:

- Uppercase characters
- A field delimiter (blank space or tab)
- A newline character
- A comment character (#)
- A hyphen or minus (= ASCII 45) character
- The @ character
- A blank space

takes effect
: After you edit your onconfig file and restart the database server and update the sqlhosts file or registry of each database server. In addition, the INFORMIXSERVER environment variable for all users might need to be changed.

Usage

The database server name is associated with a communication protocol that is specified in the sqlhosts file or registry. If the database server uses multiple communication protocols, define values for database server names with the DBSERVERALIASES configuration parameter.

Client applications use the database server name in the INFORMIXSERVER environment variable and in SQL statements such as CONNECT and DATABASE, which establish a connection to a database server.
Important: To avoid conflict with other instances of Informix® database servers on the same computer or node, you should use the DBSERVERNAME configuration parameter to assign a database server name explicitly.
For Informix MaxConnect users, the value of the INFORMIXSERVER environment variable on the client must match either the value of the DBSERVERNAME configuration parameter or one of the entries of the DBSERVERALIASES configuration parameter.

High-availability cluster servers that use shared-memory connections must also have TCP connection aliases for server-to-server communication. If a high-availability cluster server's DBSERVERNAME is associated with a shared-memory sqlhosts file entry, you must create a TCP alias for the server by setting a DBSERVERALIASES value, setting the HA_ALIAS configuration parameter to the DBSERVERALIASES value, and then creating a TCP sqlhost file entry for the alias.

Related reference:
DBSERVERALIASES configuration parameter
HA_ALIAS configuration parameter
NETTYPE configuration parameter
NUMFDSERVERS configuration parameter
onmode -d: Set data-replication types
onmode -d: Set High Availability server characteristics
onmode -d command: Replicate an index with data-replication
Related information:
Connection information set in the DBSERVERNAME configuration parameter
Multiple connection types
Add listen threads
INFORMIXSERVER environment variable

DBSPACETEMP configuration parameter

Use the DBSPACETEMP configuration parameter to specify a list of dbspaces that the database server uses to globally manage the storage of temporary tables.

DBSPACETEMP improves performance by enabling the database server to spread out I/O for temporary tables efficiently across multiple disks. The database server also uses temporary dbspaces during backups to store the before-images of data that are overwritten while the backup is occurring.

onconfig.std value
: Not set. Temporary tables are stored in the root dbspace.

separators
: Comma or colon (no white space)

values
: One or more dbspace names. Dbspaces can be standard dbspace, temporary dbspaces, or both. Separate dbspace names with a colon or comma. The length of the list cannot exceed 254 bytes.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

DBSPACETEMP can contain dbspaces with a non-default page size and dbspaces in the DBSPACETEMP list can have different page sizes.

If a client application needs to specify an alternative list of dbspaces to use for its temporary-table locations, the client can use the DBSPACETEMP environment variable to list them. The database server uses the storage locations that the DBSPACETEMP environment variable specifies only when you use the HIGH option of UPDATE STATISTICS.

If both standard and temporary dbspaces are listed in the DBSPACETEMP configuration parameter or environment variable, the following rules apply:

- Sort, backup, implicit, and nonlogging explicit temporary tables are created in temporary dbspaces if adequate space exists.
- Explicit temporary tables created without the WITH NO LOG option are created in standard (rather than temporary) dbspaces.

When you create a temporary dbspace with the **onspaces** utility or the SQL administration API **admin()** or **task()** function, the database server does not use the newly created temporary dbspace until you modify the DBSPACETEMP configuration parameter to include the new temporary dbspace or set the DBSPACETEMP environment

variable and restart the session. Note that the DBSPACETEMP configuration parameter may be modified dynamically with the **onmode -wf** or **onmode -wm** commands.

The DBSPACETEMP environment variable takes effect immediately and overrides the DBSPACETEMP configuration parameter.

- Use Hash Join Overflow and DBSPACETEMP
  Informix uses an operating-system directory or file to direct any overflow that results from certain database operations, if you do not set the **DBSPACETEMP** environment variable or DBSPACETEMP configuration parameter.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onspaces -c -d: Create a dbspace
onstat -d command: Print chunk information
Related information:
Temporary tables
Configure dbspaces for temporary tables and sort files
DBSPACETEMP environment variable
PSORT_DBTEMP environment variable

Use Hash Join Overflow and DBSPACETEMP

Informix® uses an operating-system directory or file to direct any overflow that results from certain database operations, if you do not set the **DBSPACETEMP** environment variable or DBSPACETEMP configuration parameter.

You can specify the operating-system directory or file in the following ways:

- SELECT statement with GROUP BY clause
- SELECT statement with ORDER BY clause
- Hash-join operation
- Nested-loop join operation
- Index builds

Location of the sort overflow files

The following table lists the environment variables and ONCONFIG configuration parameters that you can use to specify the location of the sort overflow files.

Table 1. Location of sort overflow files

Variable or Parameter	Location of the sort overflow files
PSORT_DBTEMP environment variable	The location specified in the environment variable
DBSPACETEMP environment variable	The location specified in the environment variable
DBSPACETEMP configuration parameter specified in the ONCONFIG file	The dbspace that is specified in the ONCONFIG file DBSPACETEMP configuration parameter

If more than one variable or parameter is specified, the priority by which the Informix determines the location of the sort overflow files is:

1. PSORT_DBTEMP environment variable
2. DBSPACETEMP environment variable
3. DBSPACETEMP ONCONFIG variable
4. DUMPDIR
5. $INFORMIXDIR/tmp

If the environment variables or configuration parameter are not set, the sort overflow files are placed in the **$INFORMIXDIR/tmp** directory and the temporary tables are placed in the rootdbspace.

DD_HASHMAX configuration parameter

Use the DD_HASHMAX configuration parameter to specify the maximum number of tables in each hash bucket in the data-dictionary cache.

A *hash bucket* is the unit of storage (typically a page) whose address is computed by the hash function. A hash bucket contains several records.

For example, if the DD_HASHMAX configuration parameter is set to 10 and the DD_HASHSIZE configuration parameter is set to 59, you can store information about 590 tables in the data-dictionary cache, and each hash bucket can have a maximum of 10 tables.

Use a text editor to modify the configuration file.

onconfig.std value
: DD_HASHMAX 10

values
: Positive integers

units
: Maximum number of tables in a hash bucket

takes effect
: After you edit your onconfig file and restart the database server.

Related reference:
DD_HASHSIZE configuration parameter
Related information:
Effect of configuration on memory utilization

DD_HASHSIZE configuration parameter

Use the DD_HASHSIZE configuration parameter to specify the number of hash buckets or lists that are in the data-dictionary cache.

Use a text editor to modify the configuration file.

onconfig.std value
: DD_HASHSIZE 31

values
: Any positive integer; a prime number is recommended

units
: Number of hash buckets or lists

takes effect
: After you edit your onconfig file and restart the database server.

Related reference:
DD_HASHMAX configuration parameter
Related information:
Effect of configuration on memory utilization

DEADLOCK_TIMEOUT configuration parameter

Use the DEADLOCK_TIMEOUT configuration parameter to specify the maximum number of seconds that a database server thread can wait to acquire a lock.

Use this parameter only for distributed queries that involve a remote database server. Do not use this parameter for nondistributed queries.

onconfig.std value
: DEADLOCK_TIMEOUT 60

values
: Positive integers

units
: Seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

If a distributed transaction is forced to wait longer than the number of seconds specified with the DEADLOCK_TIMEOUT configuration parameter, the thread that owns the transaction assumes that a multi-server deadlock exists.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -p command: Print profile counts
Related information:
Multiphase commit protocols
Configuration parameters used in two-phase commits

DEF_TABLE_LOCKMODE configuration parameter

Use the DEF_TABLE_LOCKMODE configuration parameter to specify the lock mode at the page or row level for new tables.

onconfig.std value
: PAGE

values
: PAGE = sets lock mode to page for new tables
: ROW = sets lock mode to row for new tables

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

precedence rules
You can supersede all other lock mode settings for a specific table by including the LOCK MODE clause in the CREATE TABLE or ALTER TABLE statement.
The IFX_DEF_TABLE_LOCKMODE environment variable set on the client takes precedence over the variable on the server and the DEF_TABLE_LOCKMODE configuration parameter.
The IFX_DEF_TABLE_LOCKMODE environment variable set on the server takes precedence over the DEF_TABLE_LOCKMODE configuration parameter.

Usage

If the DEF_TABLE_LOCKMODE configuration parameter is set to `ROW`, it sets the lock mode to row for every newly created table for all sessions that are connected to logging or nonlogging databases. This parameter has no effect on the lock mode for existing tables.

If the DEF_TABLE_LOCKMODE configuration parameter is set to PAGE, the USELASTCOMMITTED configuration parameter and COMMITTED READ LAST COMMITTED option of the SET ISOLATION statement cannot enable access to the most recently committed data in tables on which uncommitted transactions hold exclusive locks, unless the tables were explicitly created or altered to have ROW as their locking granularity.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
USELASTCOMMITTED configuration parameter
Related information:
IFX_DEF_TABLE_LOCKMODE environment variable
Configuring the lock mode
Precedence and Default Behavior

DEFAULTESCCHAR configuration parameter

The DEFAULTESCCHAR configuration parameter specifies the default escape character that is used for LIKE and MATCHES conditions.

onconfig.std value
DEFAULTESCCHAR backslash character (\ ).
if not present
The backslash character (\ ) is used if no value is set in the onconfig file.
values
\ = The backslash character is used as the escape character.
NONE = No default escape character.

character = Any one-character value can be used as the escape character.

takes effect
After you edit your onconfig file and restart the database server.

Usage

The default value can be overridden in a session by using the `SET ENVIRONMENT DEFAULTESCCHAR` statement with the escape character that you want to use. For example:

```
SET ENVIRONMENT DEFAULTESCCHAR '\'
```

Related information:
DEFAULTESCCHAR session environment option

DELAY_APPLY Configuration Parameter

Use the DELAY_APPLY configuration parameter to configure RS secondary servers to wait for a specified period of time before applying logs.

onconfig.std value
DELAY_APPLY 0
default value
0
values
0 = Apply logs
-1= Stage log files and apply data immediately.
A number followed by a time unit: for example, `1H` sets the delay to one hour.
number: `-1-999` = Number of days, minutes, hours, or seconds to wait.
time_unit: `D`, `H`, `M`, or `S`, where *D* = Days, *H* = Hours, *M* = Minutes, and *S* = Seconds. Values are not case sensitive.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Delaying the application of log files allows you to recover quickly from erroneous database modifications by restoring the data from the RS secondary server. When setting the value of DELAY_APPLY you must also set LOG_STAGING_DIR. If DELAY_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, then the server cannot be initialized.

You must specify a valid and secure location for the log files by setting the LOG_STAGING_DIR configuration parameter. The logs in the staging directory are purged after the last checkpoint has been processed on the RS secondary server.

To see information about the data being sent to the log-staging directory set for a RS secondary server, run the **onstat -g rss verbose** command on the RS secondary server.

If the write to the staging file fails, the RS secondary server raises event alarm 40007.

If a remote stand-alone secondary (RSS) server has its DELAY_APPLY configuration parameter set to a value other than 0, that server cannot use cluster transaction coordination.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
STOP_APPLY configuration parameter
CLUSTER_TXN_SCOPE configuration parameter
LOG_STAGING_DIR configuration parameter
onstat -g cluster command: Print high-availability cluster information
Related information:
CLUSTER_TXN_SCOPE session environment option
Delayed application of log records

DIRECT_IO configuration parameter (UNIX)

Use the DIRECT_IO configuration parameter to control the use of direct I/O for cooked files used for dbspace chunks.

This parameter enables direct I/O (bypassing file system buffering) on UNIX platforms or concurrent IO (bypassing both file system buffering and unnecessary write serialization) on AIX® operating systems.

onconfig.std value
    DIRECT_IO 0
values
    0 = Neither direct I/O or concurrent I/O is used
    1 = Direct I/O, which bypasses file system buffering, is used if available
    2 = Concurrent I/O is enabled on AIX operating systems (The concurrent I/O option includes direct I/O and concurrent I/O.)
takes effect
    After you edit your onconfig file and restart the database server.

Usage

Direct I/O can only be used for dbspace chunks whose file systems support direct I/O for the page size.

By using direct I/O, you might be able to reduce the number of AIO virtual processors.

If direct I/O is enabled, KAIO (kernel asynchronous I/O) is used if the file system supports it. However, KAIO is not used if the environment variable KAIOOFF is set. When direct IO and KAIO are both used, the number of AIO virtual processors can be reduced. If direct IO is used, but KAIO is not, the number of AIO virtual processors should not be reduced.

IBM® Informix® does not use direct or concurrent I/O for cooked files used for temporary dbspace chunks.

On AIX, if Informix uses concurrent I/O for a chunk, another program (such as an online external backup program) must also use concurrent I/O. If not, the file open operation will fail.

If Informix uses direct I/O for a chunk, and another program tries to open the chunk file without using direct I/O, the open operation will normally succeed, but there can be a performance penalty. The penalty can occur because the file system might attempt to ensure that each open operation views the same file data, either by not using direct I/O at all for the duration of the conflicting open operation, or by flushing the file system cache before each direct I/O and invalidating the file system cache after each direct write.

Direct I/O is used for dbspace chunks on Windows platforms regardless of the value of the DIRECT_IO configuration parameter.

Related reference:
AUTO_AIOVPS configuration parameter
onstat -d command: Print chunk information
Related information:
Improving the performance of cooked-file dbspaces by using direct I/O
Direct I/O (UNIX)
Concurrent I/O (AIX only)

DIRECTIVES configuration parameter

Use the DIRECTIVES configuration parameter to enable or disable the use of optimizer directives. These directives specify behavior for the query optimizer in developing query plans for SELECT, UPDATE, and DELETE statements.

onconfig.std value
DIRECTIVES 1
values
0 = Optimizer directives disabled
1 = Optimizer directives enabled

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
environment variable
IFX_DIRECTIVES

Usage

Set DIRECTIVES to 1, which is the default value, to enable the database server to process optimizer directives. Set DIRECTIVES to 0 to disable the database server from processing directives.

Client programs also can set the IFX_DIRECTIVES environment variable to ON or OFF to enable or disable processing of directives by the database server. The setting of the IFX_DIRECTIVES environment variable overrides the setting of the DIRECTIVES configuration parameter. If you do not set the IFX_DIRECTIVES environment variable, all sessions for a client inherit the database server configuration for processing directives.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Optimizer directives
IFX_DIRECTIVES environment variable
Other syntax segments

DISABLE_B162428_XA_FIX configuration parameter

Use the DISABLE_B162428_XA_FIX configuration parameter to specify when transactions are freed.

onconfig.std value
Not in the onconfig.std file.
values
0 = (Default) Frees transactions only when an xa_rollback is called.
1 = Frees transactions if transaction rollback for other than an xa_rollback.

units
Integer
takes effect
After you edit your onconfig file and restart the database server.

Usage

Set DISABLE_B162428_XA_FIX to 1 to immediately free all global transactions after a transaction rollback, which is the default for IBM® Informix® 9.40 and earlier versions. The default behavior for Informix 10.0 is to free global transactions after an xa_rollback is called, and this behavior is required to confirm to the XA state table that a transaction can be freed only after xa_rollback is called. Setting DISABLE_B162428_XA_FIX to 1 ensures that applications written for the earlier version of Informix work properly.

You can override the DISABLE_B162428_XA_FIX configuration parameter for a client session with the IFX_XASTDCOMPLIANCE_XAEND environment variable. Setting IFX_XASTDCOMPLIANCE_XAEND to 1 will free transactions only when an xa_rollback is called. Setting IFX_XASTDCOMPLIANCE_XAEND to 0 will free transactions if the transaction rollback is for other than an xa_rollback.

Related information:
IFX_XASTDCOMPLIANCE_XAEND environment variable

DISK_ENCRYPTION configuration parameter

The DISK_ENCRYPTION configuration parameter controls the encryption of storage spaces.

onconfig.std value
Not set. Storage space encryption is disabled.
values
See Usage section.
takes effect
After you edit your onconfig file and restart the database server.

Usage

Use the DISK_ENCRYPTION configuration parameter to enable storage space encryption, set the name of the encryption file names, and specify the encryption cipher. Any storage spaces that you create after you set the DISK_ENCRYPTION configuration parameter are encrypted by default. Storage spaces that you created before you set the DISK_ENCRYPTION configuration parameter are not automatically encrypted. When storage space encryption is enabled, you can restore a storage space as encrypted or unencrypted, regardless of whether the space was encrypted at the time of the back up.

```
Syntax for the DISK_ENCRYPTION configuration parameter

>>-DISK_ENCRYPTION--keystore--=--keystore_name------------------>

>--+-------------------------+-------------------------------->
   '-,--cipher--=--+-aes128-+-'
                   +-aes192-+
                   '-aes256-'

>--+---------------------------------------+------------------><
   '-,--rollfwd_create_dbs--=--+-encrypt-+-'
                               '-decrypt-'
```

Table 1. Options for the DISK_ENCRYPTION configuration parameter value

Field	Value
keystore	The *keystore* specifies the name of the keystore and stash file names. The files are created in the INFORMIXDIR/etc directory: • *keystore*.p12 = The keystore file that contains the security certificates. • *keystore*.sth = The stash file that contains the encryption password. You must manually back up the keystore and password stash files. These files are not backed up when you run a back up with the ON-Bar or **ontape** utilities.
cipher	Specifies the encryption cipher: • **aes128** = Default. Advanced Encryption Standard cipher with 128-bit keys. • **aes192** = Advanced Encryption Standard cipher with 192-bit keys. • **aes256** = Advanced Encryption Standard cipher with 256-bit keys.
rollfwd_create_dbs	Specifies whether to encrypt a storage space that is created by the rolling forward of the logical log during a restore: • **encrypt** = Encrypt the newly created storage space • **decrypt** = Do not encrypt the newly created storage space By default, storage spaces that are created by the rolling forward of the logical log have the same encryption state as the original storage space.

Related information:
Storage space encryption

DRDA_COMMBUFFSIZE configuration parameter

Use the DRDA_COMMBUFFSIZE configuration parameter to specify the size of the DRDA communications buffer.

When a DRDA session is established, the session is allocated a communication buffer equal to the current buffer size. If the buffer size is subsequently changed, existing connections are not affected, but new DRDA connections use the new size. IBM® Informix® silently resets values greater than 2 Megabyte to 2 Megabytes and resets values less than 4 Kilobytes to the 32 Kilobyte default value.

onconfig.std value
: Not in the onconfig.std file.

if not present
: 32K

values
: Minimum = 4 Kilobytes
: Maximum = 2 Megabytes

takes effect
: When shared memory is initialized

Usage

Users might specify the DRDA_COMMBUFFSIZE value in either MB or KB by adding either ‘M' or ‘K' to the value. The letter is not case sensitive, and the default is kilobytes. For example, a one megabyte buffer can be specified in any of these ways:

- DRDA_COMMBUFFSIZE 1M
- DRDA_COMMBUFFSIZE 1m
- DRDA_COMMBUFFSIZE 1024K
- DRDA_COMMBUFFSIZE 1024k
- DRDA_COMMBUFFSIZE 1024

Related information:
Specify the size of the DRDA communication buffer with the DRDA_COMMBUFFSIZE configuration parameter

DRAUTO configuration parameter

Set the DRAUTO configuration parameter to specify a HDR-failover method for HDR high-availability systems.

onconfig.std value
DRAUTO 0
Range of values

Value	Description
0	Automatic failover is disabled. When the primary server fails or loses network connectivity, the HDR secondary server becomes read-only.
1	Automatic failover is enabled. When the primary server fails or loses network connectivity, convert the HDR secondary server to a standard server. The HDR secondary server gracefully ends client connections and shuts down, and then restarts as a standard server. When the failed primary server restarts or reconnects to the network, convert the standard server back to the HDR secondary server. Do not use this setting if you configured Connection Managers to perform failover.
2	Automatic failover is enabled. When the primary server fails or loses network connectivity, convert the HDR secondary server to a primary server. The HDR secondary server maintains client connections and does not shut down. When the failed primary server restarts or reconnects to the network, convert it to an HDR secondary server. Do not use this setting if you configured Connection Managers to perform failover.
3	Failover is controlled by Connection Managers. Connection Managers must be configured and active for automatic failover.

Takes effect
When shared memory is initialized.

Usage

All servers of a high-availability cluster must have the same DRAUTO configuration parameter setting.

The DRAUTO configuration parameter does not control failover for SDS secondary servers or RS secondary servers. To set automatic failover for a high-availability cluster that has SD secondary servers or RS secondary servers, configure Connection Managers.

Important: If you are using Connection Managers to control failover, the DRAUTO configuration parameter must be set to 3 on all cluster servers. You must not perform a manual failover while Connection Managers are active.
Related reference:
onstat -g dri command: Print high-availability data replication information
Related information:
Fully synchronous mode for HDR replication
Asynchronous mode for HDR replication
Nearly synchronous mode for HDR replication
Replication of primary-server data to secondary servers

DRIDXAUTO configuration parameter

Use the DRIDXAUTO configuration parameter to specify whether the primary High-Availability Data Replication (HDR) server automatically starts index replication if the secondary HDR server detects a corrupted index.

onconfig.std value
DRIDXAUTO 0
values
0 = Off
1 = On

takes effect
After you edit your onconfig file and restart the database server.

Usage

To alter the value of the DRIDXAUTO configuration parameter for an active server instance, use the **onmode -d idxauto** command. You do not need to restart the server instance. However, the **onmode -d idxauto** command will not change the value of the DRIDXAUTO configuration parameter in the onconfig file.

Related reference:
onstat -g dri command: Print high-availability data replication information
onmode -d command: Replicate an index with data-replication

DRINTERVAL configuration parameter

Use the DRINTERVAL configuration parameter to specify the maximum number of seconds between flushes of the data-replication buffer, whether to use HDR SYNC mode, or whether to use the synchronization mode that is specified by the HDR_TXN_SCOPE configuration parameter.

onconfig.std value
DRINTERVAL 0
values
-1 = Use HDR SYNC mode. Replication is synchronous if the primary server uses unbuffered logging.
0 = The value of the HDR_TXN_SCOPE configuration parameter determines the synchronization mode for HDR data replication.

positive integers = Use HDR ASYNC mode. The positive integer is the maximum number of seconds between flushes of the data-replication buffer.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The DRINTERVAL configuration parameter controls replication latency, and is used to set the replication synchronization.

If used with unbuffered logging, HDR SYNC mode is the same as the nearly synchronous mode that is set through the HDR_TXN_SCOPE configuration parameter.

Table 1. Matrix of DRINTERVAL, HDR_TXN_SCOPE, and logging settings, and their resulting HDR replication modes.

DRINTERVAL	HDR_TXN_SCOPE	Logging	Result
-1	n/a	buffered	Asynchronous replication
-1	n/a	unbuffered	Nearly synchronous replication
0	FULL_SYNC	buffered	Fully synchronous replication
0	FULL_SYNC	unbuffered	Fully synchronous replication
0	ASYNC	buffered	Asynchronous replication
0	ASYNC	unbuffered	Asynchronous replication
0	NEAR_SYNC	buffered	Nearly synchronous replication
0	NEAR_SYNC	unbuffered	Nearly synchronous replication
positive integer	n/a	buffered	Asynchronous replication
positive integer	n/a	unbuffered	Asynchronous replication

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g dri command: Print high-availability data replication information
HDR_TXN_SCOPE configuration parameter
FAILOVER_TX_TIMEOUT configuration parameter
Related information:
HDR_TXN_SCOPE session environment option
Fully synchronous mode for HDR replication
Asynchronous mode for HDR replication
Nearly synchronous mode for HDR replication
Replication of primary-server data to secondary servers
Replication latency for secondary servers

DRLOSTFOUND configuration parameter

Use the DRLOSTFOUND configuration parameter to specify the path name to the HDR lost-and-found file. This file indicates that some transactions were committed on the HDR primary database server before that were not committed on the secondary database server when the primary database server experienced a failure.

onconfig.std values
On UNIX: $INFORMIXDIR/etc/dr.lostfound
On Windows: $INFORMIXDIR\tmp

values
pathname = Path name of the dr.lostfound file
takes effect
After you edit your onconfig file and restart the database server.

The DRLOSTFOUND configuration parameter is not applicable if updates between the primary and secondary database servers occur synchronously, when the DRINTERVAL configuration parameter is set to -1.

The lost-and-found file, dr.lostfound.*timestamp*, is created with a time stamp that is appended to the file name so that the database server does not overwrite another lost and found file if another file exists. You cannot use the lost-and-found file to reapply lost transactions.

Related reference:
onstat -g dri command: Print high-availability data replication information
Related information:
Lost-and-found transactions

DRTIMEOUT configuration parameter

Use the DRTIMEOUT configuration parameter to specify the length of time, in seconds, that a database server in a high-availability data-replication pair waits for a transfer acknowledgment from the other database server in the pair. This parameter applies only to high-availability data-replication pairs.

onconfig.std value
: DRTIMEOUT 30

values
: Positive integers

units
: Seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

Use the following formula to calculate the value to specify for the DRTIMEOUT configuration parameter:

```
DRTIMEOUT = wait_time / 4
```

In this formula, *wait_time* is the length of time, in seconds, that a database server in a high-availability data-replication pair must wait before the server assumes that a high-availability data-replication failure occurred.

For example, you determine that *wait_time* for your system is 160 seconds. Use the preceding formula to set DRTIMEOUT as follows:

```
DRTIMEOUT = 160 seconds / 4 = 40 seconds
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g dri command: Print high-availability data replication information
Related information:
Fully synchronous mode for HDR replication
Asynchronous mode for HDR replication
Nearly synchronous mode for HDR replication
Replication of primary-server data to secondary servers

DS_HASHSIZE configuration parameter

Use the DS_HASHSIZE configuration parameter to specify the number of hash buckets in the data-distribution cache and other caches. The database server stores and accesses column statistics that the UPDATE STATISTICS statement generates in the MEDIUM or HIGH mode in the data-distribution cache.

onconfig.std value
: DS_HASHSIZE 31

values
: Any positive integer; a prime number is recommended

units
: Number of hash buckets or lists

takes effect
: After you edit your onconfig file and restart the database server.

Usage

Update the value of the DS_HASHSIZE and the DS_POOLSIZE configuration parameter to improve the performance of frequently used queries in a multiuser environment.

The DS_HASHSIZE configuration parameter sets the number of hash buckets for the following caches:

- Data-distribution cache
- Extend type name cache
- Extended type ID cache
- Cast cache
- Operator class instance cache
- Routine resolution cache

- Aggregate cache
- Secondary transient cache

Related reference:
DS_POOLSIZE configuration parameter
onstat -g dsc command: Print distribution cache information
Related information:
Data-distribution configuration
Configure and monitor memory caches

DS_MAX_QUERIES configuration parameter

Use the DS_MAX_QUERIES configuration parameter to specify the maximum number of parallel database queries (PDQ) that can run concurrently.

The value of the DS_MAX_QUERIES configuration parameter is dependent on the setting for the DS_TOTAL_MEMORY configuration parameter:

- If the DS_TOTAL_MEMORY configuration parameter is set, then the value of the DS_MAX_QUERIES is `DS_TOTAL_MEMORY / 128`, rounded down to the nearest integer value.
- If the DS_TOTAL_MEMORY configuration parameter is not set, then the value of the DS_MAX_QUERIES configuration parameter is `2 * num`, where `num` is the number of CPUs specified in the VPCLASS configuration parameter.

onconfig.std value
: Not set.

if not present
: `2* num * 128`, where `num` is the number of CPUs specified in the VPCLASS configuration parameter.

values
: Minimum value = 1
Maximum value = 8,388,608 (8 megabytes)

units
: Number of queries

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The Memory Grant Manager (MGM) reserves memory for a query based on the following formula:

```
memory_reserved = DS_TOTAL_MEMORY *
                    (PDQ-priority / 100) *
                  (MAX_PDQPRIORITY / 100)
```

The value of PDQPRIORITY is specified in either the PDQPRIORITY environment variable or the SQL statement SET PDQPRIORITY.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -D, -M, -Q, -S: Change decision-support parameters
onstat -g mgm command: Print MGM resource information
VPCLASS configuration parameter
Related information:
Parallel database query (PDQ)
PDQPRIORITY environment variable

DS_MAX_SCANS configuration parameter

Use the DS_MAX_SCANS configuration parameter to limit the number of PDQ scan threads that the database server can execute concurrently.

onconfig.std value
: DS_MAX_SCANS 1048576 or (1024 * 1024)

values
: 10 - (1024 * 1024)

units
: Number of PDQ scan threads

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When a user issues a query, the database server apportions some number of scan threads, depending on the following values:

- The value of PDQ priority (set by the environment variable PDQPRIORITY or the SQL statement SET PDQPRIORITY)
- The ceiling that you set with DS_MAX_SCANS
- The factor that you set with MAX_PDQPRIORITY
- The number of fragments in the table to scan (*nfrags* in the formula)

The Memory Grant Manager (MGM) tries to reserve scan threads for a query according to the following formula:

```
reserved_threads = min (nfrags, (DS_MAX_SCANS *
                    PDQPRIORITY / 100 *
                   MAX_PDQPRIORITY / 100) )
```

If the DS_MAX_SCANS part of the formula is greater than or equal to the number of fragments in the table to scan, the query is held in the ready queue until as many scan threads are available as there are table fragments. Once underway, the query executes quickly because threads are scanning fragments in parallel.

For example, if *nfrags* equals `24`, DS_MAX_SCANS equals `90`, PDQPRIORITY equals `50`, and MAX_PDQPRIORITY equals `60`, the query does not begin execution until *nfrags* scan threads are available. Scanning takes place in parallel.

If the DS_MAX_SCANS formula falls below the number of fragments, the query might begin execution sooner, but the query takes longer to execute because some threads scan fragments serially.

If you reduce DS_MAX_SCANS to `40` in the previous example, the query needs fewer resources (12 scan threads) to begin execution, but each thread needs to scan two fragments serially. Execution takes longer.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -D, -M, -Q, -S: Change decision-support parameters
onstat -g mgm command: Print MGM resource information
Related information:
Parallel database query (PDQ)
PDQPRIORITY environment variable

DS_NONPDQ_QUERY_MEM configuration parameter

Use the DS_NONPDQ_QUERY_MEM configuration parameter to increase the amount of memory that is available for a query that is not a Parallel Database Query (PDQ). (You can only use this parameter if PDQ priority is set to zero.)

onconfig.std value
: DS_NONPDQ_QUERY_MEM:

  - On UNIX: 256
  - On Windows: 128

values
: From the default value to 25 percent of the value of DS_TOTAL_MEMORY

units
: Kilobytes

takes effect
: After you edit your onconfig file and restart the database server.
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
  When you reset the value in memory by running the **onmode -wm** command.

Usage

If you specify a value for the DS_NONPDQ_QUERY_MEM parameter, determine and adjust the value based on the number and size of table rows.
Tip: Set the value to generally not exceed the largest available temporary dbspace size.
The DS_NONPDQ_QUERY_MEM value is calculated during database server initialization based on the calculated DS_TOTAL_MEMORY value. If during the processing of the DS_NONPDQ_QUERY_MEM, the database server changes the value that you set, the server sends a message in this format:

```
DS_NONPDQ_QUERY_MEM recalculated and changed from old_value Kb to new_value Kb.
```

In the message, *old_value* represents the value that you assigned to DS_NONPDQ_QUERY_MEM in the user configuration file, and *new_value* represents the value determined by the database server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g mgm command: Print MGM resource information

DS_POOLSIZE configuration parameter

Use the DS_POOLSIZE parameter to specify the maximum number of entries in the data-distribution cache and other caches. The database server stores and accesses column statistics that the UPDATE STATISTICS statement generates in the MEDIUM or HIGH mode in the data-distribution cache.

onconfig.std value
: DS_POOLSIZE 127

values
: A positive value 127 or greater that represents half of the initial maximum number of entries in the cache. The maximum value is dependent upon the shared memory configuration and available shared memory for the server instance.

takes effect
: After you edit your onconfig file and restart the database server.
When you increase the value in memory by running the **onmode -wm** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Use the DS_HASHSIZE and the DS_POOLSIZE configuration parameters to improve performance of frequently run queries in a multi-user environment.

The initial number of entries in the cache is twice the value of the DS_POOLSIZE configuration parameter. For example, if the DS_POOLSIZE configuration parameter is set to 127, 254 entries are allowed in the cache. If all entries in a cache are full, the cache size automatically grows by 10%. To reduce the size of the cache, decrease the value of the DS_POOLSIZE configuration parameter in the onconfig file and restart the server.

The DS_POOLSIZE configuration parameter sets the number of entries in the following caches:

- Data-distribution cache
- Extend type name cache
- Extended type ID cache
- Cast cache
- Operator class instance cache
- Routine resolution cache
- Aggregate cache
- Secondary transient cache

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DS_HASHSIZE configuration parameter
onstat -g dsc command: Print distribution cache information
Related information:
Data-distribution configuration
Configure and monitor memory caches

DS_TOTAL_MEMORY configuration parameter

Use the DS_TOTAL_MEMORY configuration parameter to specify the amount of memory available for PDQ queries. The amount should be smaller than the computer physical memory, minus fixed overhead such as operating-system size and buffer-pool size.

onconfig.std value
: Not set.

if not present
: If SHMTOTAL=0 and DS_MAX_QUERIES is set, `DS_TOTAL_MEMORY = DS_MAX_QUERIES * 128`.
If SHMTOTAL=0 and DS_MAX_QUERIES is not set, `DS_TOTAL_MEMORY = num_cpu_vps * 2 * 128`.

values
: If DS_MAX_QUERIES is set, the minimum value is `DS_MAX_QUERIES * 128`.
If DS_MAX_QUERIES is not set, the minimum value is `num_cpu_vps * 2 * 128`.

There is no maximum value limit other than any limit that you might have with the software that you use on your machine.

units
: Kilobytes

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Do not confuse DS_TOTAL_MEMORY with the configuration parameters SHMTOTAL and SHMVIRTSIZE. The SHMTOTAL setting specifies all the memory for the database server (total of the resident, virtual, and message portions of memory). The SHMVIRTSIZE setting specifies the size of the virtual portion. DS_TOTAL_MEMORY is a logical subset of SHMVIRTSIZE.

For OLTP applications, set DS_TOTAL_MEMORY to between 20 and 50 percent of the value of SHMTOTAL in kilobytes.

For applications that involve large decision-support (DSS) queries, increase the value of DS_TOTAL_MEMORY to between 50 and 80 percent of SHMTOTAL. If you use your database server for DSS queries exclusively, set this parameter to 90 and 100 percent of SHMTOTAL.

Set the DS_TOTAL_MEMORY configuration parameter to any value not greater than the quantity (`SHMVIRTSIZE - 10 megabytes`).

For information on the maximum memory available on your platform, see the machine notes.

- Algorithm for DS_TOTAL_MEMORY

Related reference:

onmode -wf, -wm: Dynamically change certain configuration parameters
SHMTOTAL configuration parameter
SHMVIRTSIZE configuration parameter
VPCLASS configuration parameter
onmode -D, -M, -Q, -S: Change decision-support parameters
onstat -g mgm command: Print MGM resource information
Related information:
Algorithm for determining DS_TOTAL_MEMORY

Algorithm for DS_TOTAL_MEMORY

The database server derives a value for DS_TOTAL_MEMORY when you do not set DS_TOTAL_MEMORY, or if you set it to an inappropriate value. For information on the algorithms, see configuration effects on memory utilization in your *IBM® Informix® Performance Guide*.

DUMPCNT configuration parameter (UNIX)

Use the DUMPCNT configuration parameter to specify the number of assertion failures in a thread for which a database server dumps shared memory or generates a core file by calling the **gcore** utility.

onconfig.std value
: DUMPCNT 1

values
: Positive integers or a number of comma-separated fields as described in the Usage section.

units
: Number of shared memory dumps or core files that can be generated by each thread.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

An assertion failure occurs when the database server cannot continue normal processing.

Assertion failures can generate as many core files or shared memory dumps as permitted by the DUMPCNT configuration parameter. Further assertion failures generate errors in the message log and perhaps to the application, but no further diagnostic information is saved.

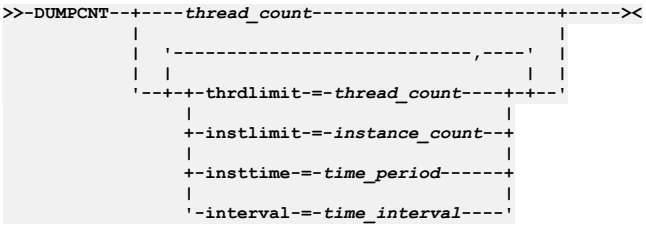


Table 1. Options for the DUMPCNT configuration parameter value

Field	Values
thrdlimit	The *thread_count* value is a positive integer that represents the maximum number of shared memory dumps to be created by each thread. It also represents the maximum number of core files to be created by each thread if the DUMPGCORE onconfig parameter is enabled. The default is `1`.
instlimit	The *instance_count* value is a positive integer that represents the maximum number of shared memory dumps to be created by all threads in the database server instance in the time period specified by the insttime field. The default is `0` which implies no limit.
insttime	The *time_period* value is a positive integer to represent the period of time in seconds to be applied to the instlimit field. When the count of shared memory dumps reaches the limit set by instlimit then no further dumps will be created until the expiry of the time period. The start of the time period is the completion of the first shared memory dump. The default value is `0` which means that there is no time period and in this case, the instlimit field becomes an absolute count on the number of shared memory dumps to be created which remains in effect until the database server is restarted.
interval	The *time_interval* value is a positive integer representing the minimum time interval in seconds between the completion of one shared memory dump before another is permitted. The default value is `300 seconds`.

Note: When used, the fields instlimit, insttime, and interval are valid only for determination of creating a shared memory dump, not a core file. They are effective only when managed shared memory dumps are enabled by the DUMPSHMEM onconfig parameter. Furthermore, in order to prevent the loss of useful diagnostic information, they are not applied for an AFCRASH event.
Related reference:

onmode -wf, -wm: Dynamically change certain configuration parameters
DUMPDIR configuration parameter
DUMPSHMEM configuration parameter (UNIX)
Related information:
Collect diagnostic information

DUMPCORE configuration parameter (UNIX)

Use the DUMPCORE configuration parameter to control whether assertion failures cause a virtual processor to dump a core image. The core file is left in the directory from which the database server was last invoked. (The DUMPDIR parameter has no impact on the location of the core file.)

onconfig.std value
DUMPCORE 0
values
0 = Do not dump core image.
1 = Dump core image.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Warning: When DUMPCORE is set to 1, an assertion failure causes a virtual processor to dump a core image, which in turn causes the database server to abort. Set DUMPCORE only for debugging purposes in a controlled environment.
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Collect diagnostic information

DUMPDIR configuration parameter

DUMPDIR specifies a directory in which the database server dumps shared memory, gcore files, or messages from a failed assertion.

Because shared memory can be large, set DUMPDIR to a file system with a significant amount of space. The directory to which DUMPDIR is set must exist for the server to start.

onconfig.std values
On UNIX: $INFORMIXDIR/tmp
On Windows: $INFORMIXDIR\tmp

values
Any directory to which user **informix** has write access
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
DUMPCNT configuration parameter (UNIX)
DUMPSHMEM configuration parameter (UNIX)
Related information:
Collect diagnostic information

DUMPGCORE configuration parameter (UNIX)

Use the DUMPGCORE configuration parameter to specify whether to dump the **gcore** core file. Use this configuration parameter with operating systems that support **gcore**.

onconfig.std value
DUMPGCORE 0
values
0 = Do not dump **gcore**.
1 = Dump **gcore**.

takes effect

After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If you set DUMPGCORE, but your operating system does not support **gcore**, messages in the database server message log indicate that an attempt was made to dump a core image, but the database server cannot find the expected file. (If your operating system does not support **gcore**, set DUMPCORE instead.)

If DUMPGCORE is set, the database server calls **gcore** whenever a virtual processor encounters an assertion failure. The **gcore** utility directs the virtual processor to dump a core image to the core.pid.cnt file in the directory that DUMPDIR specifies and continue processing.

The **pid** value is the process identification number of the virtual processor. The **cnt** value is incremented each time that this process encounters an assertion failure. The **cnt** value can range from 1 to the value of DUMPCNT. After that, no more core files are created. If the virtual processor continues to encounter assertion failures, errors are reported to the message log (and perhaps to the application), but no further diagnostic information is saved.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Collect diagnostic information

DUMPSHMEM configuration parameter (UNIX)

Use the DUMPSHMEM configuration parameter to indicate whether a shared memory dump is created on an assertion failure. This configuration parameter also specifies how much memory is written to the shmem.*pid.cnt* file in the directory specified by the DUMPDIR configuration parameter.

onconfig.std value
DUMPSHMEM 1
values
0 = Do not create a shared memory dump.
1 = Create a shared memory dump of all the shared memory that the database uses.

2 = Create a shared memory dump that excludes the buffer pools.

5 = Enables the managed shared memory dump feature. When permitted, create a shared memory dump of all the shared memory that the database uses.

6 = Enables the managed shared memory dump feature. When permitted, create a shared memory dump that excludes the buffer pools.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If DUMPSHMEM is set to 1, all the shared memory that the database server uses is dumped, which can result in a large file. When space is limited, set DUMPSHMEM to 2 because this setting creates a smaller shared-memory dump file.

The values of 5 and 6 enables the managed shared memory dump feature. This determines whether or not a shared memory dump will be created depending on the type of Assertion Failure and how many shared memory dumps may have already been created by the thread or instance. When enabled the feature manages concurrent shared memory dump requests according to the type of Assertion Failure:

- A non-fatal request (AFWARN, AFFAIL) will be ignored if a shared memory dump is already in progress. The requesting thread will continue immediately.
- A fatal request (AFCRASH) will block the requesting thread if a shared memory dump is in progress. The thread is allowed to continue upon completion of the shared memory dump.

The DUMPCNT onconfig parameter provides options to control the working of the feature.

The *pid* value is the process identification number for the virtual processor. The *cnt* value increments each time that this virtual processor encounters an assertion failure. The *cnt* value can range from 1 to the value of the DUMPCNT configuration parameter. After the value of DUMPCNT is reached, no more files are created. If the database server continues to detect inconsistencies, errors are reported to the message log (and perhaps to the application), but no further diagnostic information is saved.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DUMPCNT configuration parameter (UNIX)
DUMPDIR configuration parameter
Running onstat Commands on a Shared Memory Dump File
onstat -o command: Output shared memory contents to a file
Related information:
Collect diagnostic information

DYNAMIC_LOGS configuration parameter

Use the DYNAMIC_LOGS configuration parameter to allow logical logs to be dynamically added when necessary to prevent transaction blocking.

onconfig.std value
DYNAMIC_LOGS 2
values
0 = Turn off dynamic-log allocation.
1 = Set off the “log file required” alarm and pause to allow manual addition of a logical-log file. You can add a log file immediately after the current log file or to the end of the log file list.

2 = Turn on dynamic-log allocation. When the database server dynamically adds a log file, it sets off the “dynamically added log file” alarm.

takes effect
For HDR: when the database server is shut down and restarted
For Enterprise Replication: when Enterprise Replication is started
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If DYNAMIC_LOGS is 2, the database server automatically allocates a new log file when the next active log file contains an open transaction. Dynamic-log allocation prevents long transaction rollbacks from blocking transactions.

If you want to choose the size and location of the new logical-log file, set DYNAMIC_LOGS to 1. Use the **onparams -a** command with the size (-s), location (-d dbspace), and -i options to add a log file after the current log file.

If the value of the DYNAMIC_LOGS configuration parameter is 0 and transaction blocking occurs, shut down the database server, set DYNAMIC_LOGS to 1 or 2, and then restart the database server.
Important: If you are using Enterprise Replication with dynamic log allocation, set LTXEHWM to no higher than 70.
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
LTXEHWM configuration parameter
LTXHWM configuration parameter
onparams -a -d dbspace: Add a logical-log file
Related information:
Logical log

EILSEQ_COMPAT_MODE configuration parameter

Use the EILSEQ_COMPAT_MODE configuration parameter to control if IBM® Informix® checks whether character data inserted by a client application contains code point sequences not recognized by the locale of the current database.

onconfig.std value
EILSEQ_COMPAT_MODE 0
values
0 = validates incoming character sequences with the current locale and returns error -202 if any characters are not valid.
1 = does not validate incoming character sequences.

takes effect
After you edit your onconfig file and restart the database server.

Usage

If you set the EILSEQ_COMPAT_MODE configuration parameter to 0, only valid byte sequences can be inserted to the database.

The EILSEQ_COMPAT_MODE configuration parameter prevents a 202 error in these conditions:

- When data is being retrieved from the database.
- When an invalid character is at the end of the string and is a partial character.

Related information:
DB_LOCALE environment variable
GL_USEGLU environment variable

ENABLE_SNAPSHOT_COPY configuration parameter

Use the ENABLE_SNAPSHOT_COPY configuration parameter to enable or disable the ability to clone a server using the **ifxclone** utility.

onconfig.std value
0
values
0 = prohibit clone
1 = permit clone

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The ENABLE_SNAPSHOT_COPY configuration parameter determines whether you can create a clone of a server using the **ifxclone** utility. Set the ENABLE_SNAPSHOT_COPY configuration parameter to 1 to allow cloning. Set the value to 0 to prohibit cloning the server using the **ifxclone** utility.

If you created a server during installation, the ENABLE_SNAPSHOT_COPY configuration parameter is enabled automatically.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
The ifxclone utility

ENCRYPT_CIPHERS configuration parameter

Use the ENCRYPT_CIPHERS configuration parameter to define all ciphers and modes that can be used by the current database session. ENCRYPT_CIPHERS is used for Enterprise Replication and High-Availability Data Replication only.

onconfig.std value
Not set. Encryption ciphers are not used.
values
See the Usage section.
takes effect
After you edit your onconfig file and restart the database server.

Usage

The encryption cipher and mode used is randomly chosen among the ciphers common between the two servers. If a specific cipher is discovered to have a weakness, you should reset the ENCRYPT_CIPHERS configuration parameter value to eliminate that cipher by using the **allbut** option.

Important: Including all ciphers is more secure than including specific ciphers.

Syntax for the ENCRYPT_CIPHERS configuration parameter

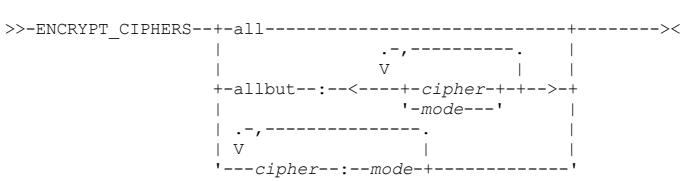


Table 1. Options for the ENCRYPT_CIPHERS configuration parameter value

Field	Description
all	Include all available ciphers and modes, except ECB mode, which is considered weak. For example: `ENCRYPT_CIPHERS all`
allbut	Include all ciphers and modes, except ECB and the ciphers and modes listed. For example: `ENCRYPT_CIPHERS allbut:<cbc,bf>` The cipher list can include unique, abbreviated entries. For example, `bf` can represent bf-1, bf-2, and bf-3; however, if the abbreviation is the name of an actual cipher, then only that cipher is eliminated. Therefore, `des` eliminates only the des cipher, but `de` eliminates the des, des3, and desx ciphers.
cipher	The following ciphers are supported: • des = DES (64-bit key) • des3 = Triple DES • desx = Extended DES (128-bit key). Only supports cbc mode. • aes = AES 128bit key • aes192 = AES 192bit key • bf-1 = Blow Fish (64-bit key) • bf-2 = Blow Fish (128-bit key) • bf-3 = Blow Fish (192-bit key) • aes128 = AES 128bit key • aes256 = AES 256bit key All modes are supported for all ciphers, except the desx cipher. For an updated list of supported ciphers, see the Release Notes.

Field	Description
mode	The following modes are supported: • ecb = Electronic Code Book (ECB). Only included if specified. • cbc = Cipher Block Chaining • cfb = Cipher Feedback • ofb = Output Feedback

Related reference:
ENCRYPT_HDR configuration parameter
ENCRYPT_MAC configuration parameter
ENCRYPT_MACFILE configuration parameter
ENCRYPT_SWITCH configuration parameter
Related information:
Encrypting data traffic between HDR database servers
Set configuration parameters for replication

ENCRYPT_HDR configuration parameter

Use the ENCRYPT_HDR configuration parameter to enable or disable HDR encryption.

onconfig.std value
Not set.
values
`0` = Disables HDR encryption
`1` = Enables HDR encryption

takes effect
When the server is initialized

Usage

Enabling HDR encryption provides a secure method for transferring data from one server to another in an HDR pair. HDR encryption works in conjunction with Enterprise Replication (ER) encryption. However, it is not necessary to have ER encryption enabled for HDR encryption. HDR encryption works whether ER encryption is enabled or not. HDR and ER share the same encryption configuration parameters: ENCRYPT_CIPHERS, ENCRYPT_MAC, ENCRYPT_MACFILE and ENCRYPT_SWITCH.

Related reference:
ENCRYPT_CIPHERS configuration parameter
ENCRYPT_MAC configuration parameter
ENCRYPT_MACFILE configuration parameter
ENCRYPT_SWITCH configuration parameter
Related information:
Encrypting data traffic between HDR database servers
Using High-Availability Clusters with Enterprise Replication

ENCRYPT_MAC configuration parameter

Use the ENCRYPT_MAC configuration parameter to control the level of message authentication code (MAC) generation. This configuration parameter is used only for Enterprise Replication and High-Availability Data Replication.

onconfig.std value
Not set
values
`off` = Does not use MAC generation
`low` = Uses XOR folding on all messages

`medium` = Uses SHA1 MAC generation for all messages that are greater than 20 bytes long and XOR folding on smaller messages

`high` = Uses SHA1 MAC generation on all messages.

example
`ENCRYPT_MAC medium,high`
takes effect
For HDR: when the database server is shut down and restarted
For Enterprise Replication: when Enterprise Replication is started

Usage

The level is prioritized to the highest value. For example, if one node has a level of **high** and **medium** enabled and the other node has only **low** enabled, then the connection attempt fails. Use the **off** entry between servers only when a secure network connection is guaranteed.

Related reference:
ENCRYPT_CIPHERS configuration parameter
ENCRYPT_HDR configuration parameter
ENCRYPT_MACFILE configuration parameter
ENCRYPT_SWITCH configuration parameter
Related information:
Encrypting data traffic between HDR database servers
Using High-Availability Clusters with Enterprise Replication

ENCRYPT_MACFILE configuration parameter

Use the ENCRYPT_MACFILE configuration parameter to specify a list of the full path names of MAC key files. This configuration parameter is used only for Enterprise Replication and High-Availability Data Replication.

onconfig.std value
: Not set.

values
: One or more full path and file names separated by commas, and the optional **builtin** keyword. For example:

```
ENCRYPT_MACFILE /usr/local/bin/mac1.dat, /usr/local/bin/mac2.dat,builtin
```

units
: Path names up to 1536 bytes in length

takes effect
: For HDR: when the database server is shut down and restarted.
For Enterprise Replication: when Enterprise Replication is started.

Usage

Each of the entries for the ENCRYPT_MACFILE configuration parameter is prioritized and negotiated at connect time. The prioritization for the MAC key files is based on their creation time by the **GenMacKey** utility. The entry created from the **builtin** keyword has the lowest priority. Because the MAC key files are negotiated, you should periodically change the keys.

Related reference:
ENCRYPT_CIPHERS configuration parameter
ENCRYPT_HDR configuration parameter
ENCRYPT_MAC configuration parameter
ENCRYPT_SWITCH configuration parameter
Related information:
Encrypting data traffic between HDR database servers
Using High-Availability Clusters with Enterprise Replication
Generating a new MAC key file

ENCRYPT_SMX configuration parameter

Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high-availability configurations on secondary servers and between Enterprise Replication Servers.

onconfig.std value
: Not set.

values
: 0 = Off. Do not encrypt.
1 = On. Encrypt where possible. Encrypt SMX transactions when the database server being connected to also supports encryption.
2 = On. Always encrypt. Only connections to encrypted database servers are allowed.

takes effect
: After you edit your onconfig file and restart the database server.

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.
Related information:
Set the wait time for SMX activity between servers
Using High-Availability Clusters with Enterprise Replication

ENCRYPT_SWITCH configuration parameter

Use the ENCRYPT_SWITCH configuration parameter to define the frequency at which ciphers or secret keys are renegotiated. This configuration parameter is used only for Enterprise Replication and High-Availability Data Replication.

The longer the secret key and encryption cipher remains in use, the more likely the encryption rules might be broken by an attacker. To avoid this, cryptologists recommend changing the secret keys on long-term connections. The default time that this renegotiation occurs is once an hour.

onconfig.std value
: Not set.

values
: Two positive integers separated by a comma. The first integer represents the number of minutes between cipher renegotiation. The second integer represents the number of minutes between secret key renegotiation. For example: ENCRYPT_SWITCH 2,5.

units
: minutes

takes effect
: For HDR: when the database server is shut down and restarted
: For Enterprise Replication: when Enterprise Replication is started

Related reference:
ENCRYPT_CIPHERS configuration parameter
ENCRYPT_HDR configuration parameter
ENCRYPT_MAC configuration parameter
ENCRYPT_MACFILE configuration parameter
Related information:
Encrypting data traffic between HDR database servers
Using High-Availability Clusters with Enterprise Replication

EXPLAIN_STAT configuration parameter

Use the EXPLAIN_STAT configuration parameter to enable or disable the inclusion of a Query Statistics section in the explain output file.

You can generate the output file by using either the SET EXPLAIN statement or the **onmode -Y** *sessionid* command. When you enable the EXPLAIN_STAT configuration parameter, the Query Statistics section shows the estimated number of rows and the actual number of returned rows in the Query Plan.

onconfig.std value
: EXPLAIN_STAT 1

values
: 0 = Disable the inclusion of a Query Statistics section in the explain output file.
: 1 = Enable the inclusion of a Query Statistics section in the explain output file.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -Y: Dynamically change SET EXPLAIN
Related information:
SET EXPLAIN statement
Sample query plan reports

EXT_DIRECTIVES configuration parameter

Use the EXT_DIRECTIVES configuration parameter to enable or disable the use of external query optimizer directives.

onconfig.std value
: EXT_DIRECTIVES 0

values
: 0 (default) = Off. The directive cannot be enabled even if IFX_EXTDIRECTIVES is on.
: 1 = On. The directive can be enabled for a session if IFX_EXTDIRECTIVES is on.

: 2 = On. The directive can be used even if IFX_EXTDIRECTIVES is not set.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

Enable external directives by using the EXT_DIRECTIVES configuration parameter in combination with the client-side IFX_EXTDIRECTIVES environment variable as follows:

The setting of the IFX_EXTDIRECTIVES environment variable overrides the setting of the EXT_DIRECTIVES configuration parameter. If you do not set the IFX_EXTDIRECTIVES environment variable, all sessions for a client inherit the database server configuration for processing external directives.

The setting specified by the SET ENVIRONMENT EXTDIRECTIVES statement of SQL overrides (for the current user session only) the settings of both the IFX_EXTDIRECTIVES environment variable and of the EXT_DIRECTIVES configuration parameter.

Related information:
External optimizer directives
IFX_EXTDIRECTIVES environment variable
EXTDIRECTIVES session environment option
Optimizer Directives

EXTSHMADD configuration parameter

Use the EXTSHMADD configuration parameter to specify the size of virtual-extension segments that are added when a user-defined routine or a DataBlade routine is run in a user-defined virtual processor.

onconfig.std value
EXTSHMADD 8192
values
32-bit operating systems: 1024 - 524288
64-bit operating systems: 1024 - 4294967296

units
KB
takes effect
After you edit your onconfig file and restart the database server.

Usage

When a thread is run in a user defined virtual processor, a virtual-extension segment is created. In the output of the **onstat -g seg** command, the virtual-extension segment has a class of VX. If the EXTSHMADD configuration parameter is not set in the onconfig file, the size of virtual-extension segments is set by the value of the SHMADD configuration parameter.

Related reference:
onstat -g seg command: Print shared memory segment statistics
SHMADD configuration parameter
Related information:
Virtual-extension portion of shared memory

FAILOVER_CALLBACK configuration parameter

Use the FAILOVER_CALLBACK configuration parameter to specify the script executed by the database server when a database server transitions from a secondary server to a primary or standard server.

onconfig.std value
Not set.
values
pathname = The full path name of the script specified by the FAILOVER_CALLBACK parameter.
takes effect
After you edit your onconfig file and restart the database server.

Usage

Set FAILOVER_CALLBACK to the full path name of the script.

FAILOVER_TX_TIMEOUT configuration parameter

In high-availability cluster environments, use the FAILOVER_TX_TIMEOUT configuration parameter to enable transactions to complete after failover of the primary server.

Use the FAILOVER_TX_TIMEOUT configuration parameter to indicate the maximum number of seconds after failover that the server waits before it begins rolling back transactions. Set the FAILOVER_TX_TIMEOUT configuration parameter to the same value on all servers in a high-availability cluster.

onconfig.std value
FAILOVER_TX_TIMEOUT 0
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When a failover occurs in a high-availability cluster environment, one of the secondary servers takes over the role of the primary server. The secondary server that becomes the new primary server is called the *failover server*.

You enable transaction survival by setting the FAILOVER_TX_TIMEOUT configuration parameter to a value greater than zero. When transaction survival is enabled, the failover server must be able to contact the remaining secondary servers to synchronize and resume any open transactions. Similarly, the surviving secondary servers must be able to establish connections to the failover server to re-send any pending transactions. The FAILOVER_TX_TIMEOUT configuration parameter specifies how long the servers wait before they begin rolling back transactions.

On the failover server, if the number of seconds specified by FAILOVER_TX_TIMEOUT is exceeded, any open transactions that are not synchronized with a surviving server are terminated and rolled back.

On the remaining secondary servers, if the number of seconds specified by FAILOVER_TX_TIMEOUT is exceeded, any open transactions on that server return an error.

Set FAILOVER_TX_TIMEOUT to 0 to immediately roll back all open transactions when failover occurs.

If the primary server fails and a secondary server fails to take over the role of the primary server, then any open transactions are rolled back, and the client is unable to make updates. For example, if an update activity has been started on a secondary server and the primary server fails, and then that failover processing does not complete and a new primary server is not established, after a predetermined amount of time, the client request times out, placing the sqlexec thread in an indeterminate state.

In the preceding scenario, active transactions are rolled back, but the physical rollback cannot occur until the new primary server is established (because the primary server manages the logs). Under these circumstances, the session can be unaware of operations that were performed on the secondary server. The session can be unaware of the rollback of a partially applied transaction because the rollback of the partial transaction cannot occur until a new primary server is established.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DRINTERVAL configuration parameter
HDR_TXN_SCOPE configuration parameter
Related information:
Replication of primary-server data to secondary servers
Fully synchronous mode for HDR replication
Nearly synchronous mode for HDR replication

FASTPOLL configuration parameter

Use the FASTPOLL configuration parameter to enable or disable fast polling of your network. FASTPOLL is a platform-specific configuration parameter.

onconfig.std value
FASTPOLL 1
values
0 = Disables fast polling.
1 = Enables fast polling.

takes effect
After you edit your onconfig file and restart the database server.

FILLFACTOR configuration parameter

Use the FILLFACTOR configuration parameter to specify the degree of index-page fullness. A low value provides room for growth in the index. A high value compacts the index.

If an index is full (100 percent), any new inserts result in splitting nodes. You can also set the FILLFACTOR as an option on the CREATE INDEX statement. The setting on the CREATE INDEX statement overrides the ONCONFIG file value.

You cannot use the FILLFACTOR configuration parameter with a forest of trees index.

onconfig.std value
FILLFACTOR 90
values
1 - 100
units
Percent
takes effect
When the index is built. Existing indexes are not changed. To use the new value, the indexes must be rebuilt.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related concepts:
Structure of B-Tree Index Pages
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

FULL_DISK_INIT configuration parameter

Use the FULL_DISK_INIT configuration parameter to prevent an accidental disk reinitialization of an existing database server instance. This configuration parameter specifies whether or not the disk initialization command (**oninit -i**) can run on your IBM® Informix® instance when a page zero exists at the root path location, which is at the first page of the first chunk location.

onconfig.std value
: FULL_DISK_INIT 0

values
: `0` = The **oninit -i** command runs only if there is not a page zero at the root path location.
`1` = The **oninit -i** command runs under all circumstances, but also resets the FULL_DISK_INIT configuration parameter to `0` after the disk initialization.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

When the FULL_DISK_INIT configuration parameter is set to `1`, any instance startup command (for example, **oninit** as well as **oninit -i**) resets the configuration parameter to `0`.

If you start to run the **oninit -i** command when the FULL_DISK_INIT configuration parameter is set to `0` and the database server finds a page zero, the **oninit -i** command does not run and the server reports an error in the online.log.

Page zero is the system page that contains general information about the server instance. This page is created when the server instance is initialized.

Related reference:
The oninit utility

GSKIT_VERSION configuration parameter

Use the GSKIT_VERSION configuration parameter to specify the major version of IBM® Global Security Kit (GSKit) that the database server uses for encryption and SSL communication.

onconfig.std value
: Not set. The version of IBM Global Security Kit installed with Informix® is used.

values
: IBM Global Security Kit versions are whole numbers that run between 7 and the latest major release number.

units
: Positive integer

takes effect
: During database initialization

If the database server is used with other products on the same computer, and a different version of IBM Global Security Kit is installed with one of the other products, the database server can be configured to use the different version of IBM Global Security Kit.

Related information:
Secure sockets layer protocol

HA_ALIAS configuration parameter

The HA_ALIAS configuration parameter defines a network alias that is used for server-to-server communication in a high-availability cluster. The specified network alias is also used by Connection Managers, the **ifxclone** utility, and **onmode -d** commands.

onconfig.std value
: Not set. The HA_ALIAS configure parameter applies to high-availability cluster servers.

values
: The HA_ALIAS configuration parameter value must match a DBSERVERNAME or DBSERVERALIASES configuration parameter value that is associated with a TCP sqlhosts file entry. If the DBSERVERNAME or the DBSERVERALIASES configuration parameter value includes the optional number of listener threads, omit the optional listener thread value from the HA_ALIAS configuration parameter value. For example, if DBSERVERNAME is set to `my_server-4`, HA_ALIAS is set to `my_server`.

takes effect
: After you edit your onconfig file and restart the database server.
For the primary server in a high-availability cluster, reset the value dynamically in your onconfig file by running the **onmode -wf** command. This method does not work for secondary servers in a high-availability cluster.
For the primary server in a high-availability cluster, reset the value in memory by running the **onmode -wm** command. This method does not work for secondary servers in a high-availability cluster.

Usage

The HA_ALIAS configuration parameter is required for high-availability cluster servers that use shared-memory connections.

For example, if a high-availability cluster server's DBSERVERNAME configuration parameter is associated with a shared-memory sqlhosts file entry, set a DBSERVERALIAS configuration parameter and a matching HA_ALIAS configuration parameter value, and then create a TCP sqlhosts file entry for the for the alias.

onconfig file values:

```
DBSERVERNAME my_server
DBSERVERALIAS alias_1
HA_ALIAS alias_1
```

sqlhosts file values:

```
#dbservername  nettype   hostname  servicename  options
 my_server     onipcshm  host_1    port_1               #client-to-server
 alias_1       onsoctcp  host_1    port_2               #server-to-server
```

Setting the HA_ALIAS configuration parameter for all servers in a high-availability cluster also enables you to separate client/server communication from server-to-server communication .

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DBSERVERALIASES configuration parameter
DBSERVERNAME configuration parameter
onmode -d: Set data-replication types
onmode -d: Set High Availability server characteristics
onmode -d command: Replicate an index with data-replication
Related information:
Connection information set in the HA_ALIAS configuration parameter

HA_FOC_ORDER configuration parameter

Use the HA_FOC_ORDER configuration parameter to define a single connection-management failover rule for a high-availability cluster of servers.

onconfig.std value
: HA_FOC_ORDER SDS,HDR,RSS

values
: A list of secondary server types, which are separated by commas and listed in priority order. For example, the default value of `SDS,HDR,RSS` means that the primary server fails over to the SD secondary server, then the HDR secondary server, and then the RS secondary server.

  - `HDR` = High-availability data replication server
  - `RSS` = Remote stand-alone secondary server
  - `SDS` = Shared-disk secondary server

  `MANUAL` = Disable automated failover for all Connection Managers in the cluster.

separators
: Separate values with a comma.

takes effect
: After you edit your onconfig file and restart the database server.
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
  When you reset the value in memory by running the **onmode -wm** command.
  After you run the SQL administration API **task()** or **admin()** function with the **-wf HA_FOC_ORDER=***value* or **-wm HA_FOC_ORDER=***value* arguments.

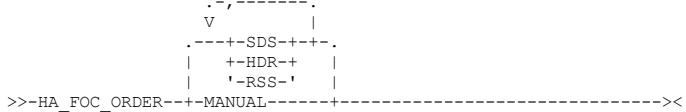
Usage

If the HA_FOC_ORDER configuration parameter is set on the primary database server of a high-availability cluster, every Connection Manager that connects to the primary server adopts the setting. The value replaces the connection unit's `ORDER=rule` failover-sequence rule. Each database server in the high-availability cluster then adopts the primary server's HA_FOC_ORDER configuration parameter value for its own HA_FOC_ORDER configuration parameter.

If the HA_FOC_ORDER configuration parameter on the primary server is set to `MANUAL`, automated failover is disabled on all Connection Managers that manager the primary server's cluster.

If the FOC ORDER value for a connection unit in a Connection Manager's configuration file is set to `DISABLED` the Connection Manager does not perform failover for that connection unit.

Syntax for the HA_FOC_ORDER configuration parameter



Example

In the following example, you have two Connection Managers that are configured to manage a cluster of three servers.

The three servers are:

- **server_1** (primary server)
- **server_2** (SD secondary server)
- **server_3** (HDR secondary server)

The first Connection Manager has the following configuration file:

```
NAME connection_manger_1

CLUSTER cluster_1
{
   INFORMIXSERVER servers_1
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=1
}
```

The second Connection Manager has the following configuration file:

```
NAME connection_manger_2

CLUSTER cluster_1
{
   INFORMIXSERVER servers_1
   SLA sla_2 DBSERVERS=ANY
   FOC ORDER=ENABLED \
       PRIORITY=2
}
```

The onconfig file of **server_1** has the following value:

```
HA_FOC_ORDER SDS,HDR
```

When **connection_manger_1** and **connection_manger_2** connect with **server_1**, their configurations become:

```
NAME connection_manger_1

CLUSTER cluster_1
{
   INFORMIXSERVER servers_1
   SLA sla_1 DBSERVERS=ANY
   FOC ORDER=SDS,HDR \
       PRIORITY=1
}

NAME connection_manger_2

CLUSTER cluster_1
{
   INFORMIXSERVER servers_1
   SLA sla_2 DBSERVERS=ANY
   FOC ORDER=SDS,HDR \
       PRIORITY=2
}
```

The values of the HA_FOC_ORDER entries in the onconfig files of **server_2** and **server_3** are updated to `SDS,HDR`.

Related information:
Example of configuring connection management for a high-availability cluster
FOC Connection Manager configuration parameter

HDR_TXN_SCOPE configuration parameter

The HDR_TXN_SCOPE configuration parameter is used with the DRINTERVAL configuration parameter to specify the synchronization mode for HDR replication in a high-availability cluster.

onconfig.std value
: HDR_TXN_SCOPE NEAR_SYNC

values
: FULL_SYNC = HDR replication if fully synchronous. Transactions require acknowledgement of completion on the HDR secondary server before they can complete.
NEAR_SYNC = HDR replication if nearly synchronous. Transactions require acknowledgement of being received on the HDR secondary server before they can complete. If used with unbuffered logging, SYNC mode, which is turned on when DRINTERVAL is set to -1, is the same as nearly synchronous mode.

: ASYNC = HDR replication if fully asynchronous. Transactions do not require acknowledgement of being received or completed on the HDR secondary server before they can complete.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
After you run the SQL administration API **task()** or **admin()** function with the "`onmode`","`-wf HDR_TXN_SCOPE=`*`value`*" or "`onmode`","`-wm HDR_TXN_SCOPE=`*`value`*" argument.

Usage

When the DRINTERVAL configuration parameter is set to 0, the value of the HDR_TXN_SCOPE parameter determines the synchronization mode for HDR replication.

If unbuffered logging is used, HDR SYNC mode is the same as the nearly synchronous mode that is set through the HDR_TXN_SCOPE configuration parameter.

Table 1. Matrix of DRINTERVAL, HDR_TXN_SCOPE, and logging settings, and their resulting HDR replication modes.

DRINTERVAL	HDR_TXN_SCOPE	Logging	Result
-1	n/a	buffered	Asynchronous replication
-1	n/a	unbuffered	Nearly synchronous replication
0	FULL_SYNC	buffered	Fully synchronous replication
0	FULL_SYNC	unbuffered	Fully synchronous replication
0	ASYNC	buffered	Asynchronous replication
0	ASYNC	unbuffered	Asynchronous replication
0	NEAR_SYNC	buffered	Nearly synchronous replication
0	NEAR_SYNC	unbuffered	Nearly synchronous replication
positive integer	n/a	buffered	Asynchronous replication
positive integer	n/a	unbuffered	Asynchronous replication

Related reference:
DRINTERVAL configuration parameter
FAILOVER_TX_TIMEOUT configuration parameter
Related information:
HDR_TXN_SCOPE session environment option

HETERO_COMMIT configuration parameter

Use the HETERO_COMMIT configuration parameter to control whether the database server participates in heterogeneous commit transactions.

onconfig.std value
HETERO_COMMIT 0
values
1 = Enable heterogeneous commit.
0 = Disable heterogeneous commit.

takes effect
After you edit your onconfig file and restart the database server.

Usage

The HETERO_COMMIT configuration parameter specifies whether or not the database server is prepared to participate with IBM® Informix® Gateway products in heterogeneous commit transactions. Setting HETERO_COMMIT to 1 allows a single transaction to update one non-Informix database (accessed with any of the Gateway products) and one or more databases.

If HETERO_COMMIT is 0, a single transaction can update databases as follows:

- One or more databases and no non-Informix databases
- One non-Informix database and no databases

You can read data from any number of and non-Informix databases, regardless of the setting of HETERO_COMMIT.

Related information:
Heterogeneous commit protocol

IFX_EXTEND_ROLE configuration parameter

Your database system administrator (DBSA), by default user **informix**, can use the IFX_EXTEND_ROLE parameter to control which users are authorized to register DataBlade modules or external user-defined routines (UDRs).

onconfig.std value
IFX_EXTEND_ROLE 1
values
1 or On (default) = Enables the requirement for the EXTEND role so that administrators can grant privileges to a user to create or drop a UDR that includes the EXTERNAL clause.
0 or Off = Disables the requirement for the EXTEND role, so that any user who holds the USAGE ON LANGUAGE privilege for the appropriate external language (C or JAVA) can register or drop an external routine that was written in that language.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Security for external routines (UDRs)

IFX_FOLDVIEW configuration parameter

Use the IFX_FOLDVIEW configuration parameter to enable or disable view folding. For certain situations where a view is involved in a query, view folding can significantly improve the performance of the query. In these cases, views are folded into a parent query instead of the query results being put into a temporary table.

onconfig.std value
IFX_FOLDVIEW 1
values
`0` or `Off` = Disables view folding.
`1` or `On` = Default. Enables view folding.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The following types of queries can take advantage of view folding:

- Views that contain a UNION ALL clause and the parent query includes a regular join, IBM® Informix® join, ANSI join, or an ORDER BY clause

A temporary table is created and view folding is not performed for the following types of queries that perform a UNION ALL operation involving a view:

- The view has one of the following clauses: AGGREGATE, GROUP BY, ORDER BY, UNION, DISTINCT, or OUTER JOIN (either or ANSI type).
- The parent query has a UNION or UNION ALL clause.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Enable view folding to improve query performance

IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter

Use the IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter to enable the transaction manager to use the same XID to represent global transactions on different databases in the same database server instance.

onconfig.std value
None
default value
0
values
`0` = disabled
`1` = enabled
takes effect
After you edit your onconfig file and restart the database server.

Usage

An XID is a global transaction ID for a distributed XA transaction.

If you set the IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter to 1, the database server allows the transaction manager to use the same XID to represent global transactions on different databases in the same database server instance. Thus, the database can be the domain instead of the server.

Related reference:
onstat -G command: Print TP/XA transaction information
Related information:
XA-compliant external data sources

INFORMIXCONRETRY configuration parameter

Use the INFORMIXCONRETRY configuration parameter to specify the maximum number of connection attempts that can be made to each database server after the initial connection attempt fails. These attempts are made within the time limit that the INFORMIXCONTIME configuration parameter specifies.

onconfig.std value
: INFORMIXCONRETRY 1

values
: Positive integers

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The INFORMIXCONTIME setting takes precedence over the INFORMIXCONRETRY setting. Connection attempts can end after the INFORMIXCONTIME value is exceeded, but before the INFORMIXCONRETRY value is reached.

To override the value of the INFORMIXCONRETRY configuration parameter for the current session, you can set either the INFORMIXCONRETRY environment option of the SET ENVIRONMENT statement or the client's INFORMIXCONRETRY environment variable.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
INFORMIXCONRETRY session environment option
INFORMIXCONRETRY environment variable

INFORMIXCONTIME configuration parameter

Use the INFORMIXCONTIME configuration parameter to specify the number of seconds that the CONNECT statement attempts to establish a connection to a database server.

onconfig.std value
: INFORMIXCONTIME 60

values
: Positive integers

units
: Seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

To set the optimal value for the INFORMIXCONTIME configuration parameter, take into account the total distance between nodes, the hardware speed, the volume of traffic, and the concurrency level of the network.

The INFORMIXCONTIME value is divided by the INFORMIXCONRETRY value to determine the number of seconds between connection attempts. If you set the INFORMIXCONTIME configuration parameter to zero, the database server uses the default value of 60 seconds.

To override the value of the INFORMIXCONTIME configuration parameter for the current session, you can set either the INFORMIXCONTIME environment option of the SET ENVIRONMENT statement or the client's INFORMIXCONTIME environment variable.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
INFORMIXCONTIME session environment option
INFORMIXCONTIME environment variable

LIMITNUMSESSIONS configuration parameter

Use the LIMITNUMSESSIONS configuration parameter to define the maximum number of sessions that you want connected to IBM® Informix®.

If you specify a maximum number, you can also specify whether you want to print messages to the online.log file when the number of sessions approaches the maximum number.

If the LIMITNUMSESSIONS configuration parameter is enabled and sessions are restricted because of this limit, both regular user threads and DBSA user threads connecting to any database count against the limit. However, a DBSA user is allowed to connect to the server even after the limit has been reached.

Distributed queries against a server are also counted against the limit.

The LIMITNUMSESSIONS configuration parameter is not intended to be used as a means to adhere to license agreements.

onconfig.std value
: Not set in the onconfig.std file

values
: maximum_number_of_sessions = 0 to 2,097,152 (2*1024*1024). The default is 0.
: print_warning = 0 (off) or 1 (on). The default for this optional value is 0.

separators
: Comma

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

If the print_warning is set to 1, a warning is triggered when the number of sessions is greater than or equal to 95 percent of the maximum_number_of_sessions value. If print_warning is set to zero, or if it is not set, no warning is issued No new user sessions can be opened after the maximum_number_of_sessions limit is reached.

If the maximum_number_of_sessions value for the LIMITNUMSESSIONS configuration parameter is set to 0, or if it is not set, there is no limit to the number of sessions that can connect to the server.

The following example specifies that you want a maximum of 100 sessions to connect to the server and you want to print a warning message when the number of connected sessions approaches 100.

```
LIMITNUMSESSIONS 100,1
```

The settings in this example cause a warning to be printed when more than 94 sessions are concurrently connected. Only a member of the DBSA group can start a new session when 100 sessions are already connected.

Use **onmode -wf** or **onmode -wm**, or the equivalent SQL administration API ONMODE commands, to dynamically increase or temporarily disable the LIMITNUMSESSIONS setting. Use this configuration parameter to allow administrative utilities to run if the database server is reaching the maximum_number_of_sessions limit.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

LISTEN_TIMEOUT configuration parameter

Use the LISTEN_TIMEOUT configuration parameter to specify the number of seconds in which the server waits for a connection.

onconfig.std value
: LISTEN_TIMEOUT 60

units
: Seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

You can set LISTEN_TIMEOUT to a lower number to guard against faulty connection requests that might indicate a Denial of Service attack.

Depending on the machine capability of holding the threads (in number), you can configure MAX_INCOMPLETE_CONNECTIONS to a higher value and depending on the network traffic, you can set LISTEN_TIMEOUT to a lower value to reduce the chance that an attack can reach the maximum limit.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
MAX_INCOMPLETE_CONNECTIONS configuration parameter

LOCKS configuration parameter

The LOCKS configuration parameter specifies the initial size of the lock table.

The lock table holds an entry for each lock. If the number of locks allocated exceeds the value of the LOCKS configuration parameter, the database server increases the size of the lock table. The lock table can be increased a maximum of 99 times.

onconfig.std value

LOCKS 20000

values
: 2,000 through 8,000,000 for 32-bit database servers 2,000 through 500,000,000 for 64-bit database servers

units
: Number of locks in the internal lock table

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The database server increases the size of the lock table by attempting to double the lock table on each increase. However, the amount added during each increase is limited to a maximum value. For 32-bit platforms, a maximum of 100,000 locks can be added during each increase. Therefore, the total maximum locks allowed for 32-bit platforms is 8,000,000 (maximum number of starting locks) + (99 (maximum number of dynamic lock table extensions) x 100,000 (maximum number of locks added per lock table extension). For 64-bit platforms, a maximum of 1,000,000 locks can be added during each increase. Therefore, the total maximum locks allowed is 500,000,000 (maximum number of starting locks) + (99 (maximum number of dynamic lock table extensions) x 1,000,000 (maximum number of locks added per lock table extension).

With the initial lock table stored in resident memory and each additional lock stored in virtual memory, locks can become a resource drain if you have a limited amount of shared memory. The amount of storage occupied by a single lock depends on the word size and operating system, and is subject to change. Currently, the amount of storage ranges from approximately 100 to 200 bytes. You can see the amount of storage required to support additional locks by restarting the server with a different value of the LOCKS configuration parameter (without making other changes), and observing the increase in memory used as shown by "onstat -g mem" for the resident pool.

Tip: When you drop a database, a lock is acquired and held on each table in the database until the database is dropped.

Related reference:
onstat -k command: Print active lock information

Related information:
The LOCKS configuration parameter and memory utilization
Locking
Shared memory
DROP DATABASE statement

LOGBUFF configuration parameter

Use the LOGBUFF configuration parameter to specify the size in kilobytes for the three logical-log buffers in shared memory.

onconfig.std value
: LOGBUFF 64

units
: Kilobytes

values
: An integer in the range of 32 - (32767 * *pagesize* / 1024), where *pagesize* is the default system page size. The value must be evenly divisible by the default system page size. If the value is not evenly divisible by the page size, the database server rounds down the size to the nearest value that is evenly divisible by the page size.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The three logical log buffers permit user threads to write to the active buffer while one of the other buffers is being flushed to disk. If flushing is not complete by the time the active buffer fills, the user thread begins writing to the third buffer.

If the RTO_SERVER_RESTART configuration parameter is enabled, set the value of the LOGBUFF configuration parameter to 256 kilobytes. If the value of the LOGBUFF configuration parameter is less than 256 kilobytes, a warning message displays when you restart the server.

Otherwise, set the value of the LOGBUFF configuration parameter to 32 kilobytes for standard workloads or 64 kilobytes for heavy workloads. The database server uses the LOGBUFF parameter to set the size of internal buffers that are used during recovery. If you set LOGBUFF too high, the database server can run out of memory and shut down during recovery.

If you log user data in smart large objects, increase the size of the log buffer to make the system more efficient. The database server logs only the portion of a smart-large-object page that changed.

You can view information about the logical log buffers by running the **onstat -l** command.

Related reference:
onstat -l command: Print physical and logical log information

Related information:
Determine database server page size
Logical-log buffer

LOGBUF_INTVL configuration parameter

Use the LOGBUF_INTVL configuration parameter to ensure the logical log buffer is flushed periodically when only buffered logging is used.

onconfig.std value
LOGBUF_INTVL 0
units
Seconds

values
The integer value of the parameter is the maximum number of seconds between logical log buffer flushes.
Default value: 0 – The feature is disabled by default. The time since the last logical log buffer flush will not be used as a criterion for flushing the buffer.

Minimum value = 1

Maximum value = 2147483647

takes effect
After you edit your onconfig file and restart the database server.

Usage

Setting this parameter will also positively affect latency in a replication environment when buffered logging is used. During relatively quiet periods on the primary, log records may be pushed to secondaries more frequently.

LOGFILES configuration parameter

Use the LOGFILES configuration parameter to specify the number of logical-log files that the database server creates during disk initialization.

onconfig.std value
LOGFILES 6
values
3 - 32,767 (integers only)
units
Number of logical-log files
takes effect
During disk initialization and when you add a new log file. You add a new log with one of the **onparms** utilities.

Usage

To change the number of logical-log files, add or drop logical-log files.

If you use **onparams** to add or drop log files, the database server automatically updates LOGFILES.

Related reference:
The onparams Utility
Related information:
Size of the logical-log file
Adding logical-log files manually
Dropping logical-log files

LOG_INDEX_BUILDS configuration parameter

Use the LOG_INDEX_BUILDS configuration parameter to enable or disable index page logging.

onconfig.std value
Not set.
values
0 = Disable
1 = Enable

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If LOG_INDEX_BUILDS is enabled, logical log file space consumption will increase, depending on the size of the indexes. This might lead to logical log file backups being required more frequently. Messages are written to the online.log file when index page logging status changes.

Tip for RS secondary servers: Using **onmode -wm** enables or disables index page logging for the current session only, and does not affect the setting in the onconfig file. If the server is stopped and restarted, the setting in the onconfig file determines whether index page logging is enabled. Therefore, enabling index page logging using **onmodem -wm** is not recommended when using RS secondary servers; instead, use **onmode -wf** to update the onconfig file, so that index page logging is enabled after restarting the server. Index page logging is a requirement when using RS secondary servers.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

LOG_STAGING_DIR configuration parameter

Use the LOG_STAGING_DIR configuration parameter to specify the location of log files received from the primary server when configuring delayed application of log files on RS secondary servers.

onconfig.std value
: Not set.

values (first parameter)
: Any valid, secure directory.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The LOG_STAGING_DIR configuration parameter specifies the directory where log files sent from the primary are stored in the following circumstances:

- The DELAY_APPLY configuration parameter is set on an RS secondary server to delay the application of logs
- The STOP_APPLY configuration parameter is set on an RS secondary server to stop the application of logs
- An RS secondary server must temporarily buffer logs
- The LOG_INDEX_BUILDS parameter is set on the HDR secondary server, and the HDR secondary server is processing checkpoints

Delaying the application of log files allows you to recover quickly from erroneous database modifications by restoring the data from the RS secondary server.
The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user **informix**, must belong to group **informix**, and must not have public read, write, or execute permission.

The directory should have enough space to hold all the logical logs that are staged. Choose a directory capable of storing at least twice the total logical logs on the primary server. To estimate the storage size, multiply the value of the LOGBUFF configuration parameter with the value of the LOGFILES configuration parameter, and then double that value.

To see information about the data being sent to the log-staging directory set for a RS secondary server, run the **onstat -g rss verbose** command on the RS secondary server.

If the write to the staging file fails, the RS secondary server raises event alarm 40007.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DELAY_APPLY Configuration Parameter
STOP_APPLY configuration parameter
Related information:
Delayed application of log records
Remote standalone secondary servers

LOGSIZE configuration parameter

Use the LOGSIZE configuration parameter to specify the size that is used when logical-log files are created.

onconfig.std value
: LOGSIZE 10000

units
: Kilobytes

values
: An integer value.
: Minimum value = `200`

: Maximum value when the database server is first initialized = `(ROOTSIZE - PHYSFILE - 512 - (63 * pagesize/1024)) / LOGFILES`

: The *pagesize* value is the default system page size for the operating system.

: If you expand the root dbspace or move logical logs to a different dbspace, the maximum size of logical log files cannot exceed the following page size-dependent value:

  - 1 GiB for page size = 2 KiB
  - 2 GiB for page size = 4 KiB

: This limit is the maximum number of pages that the log position can describe for those page sizes.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

When you reset the value in memory by running the **onmode -wm** command.

Usage

When you change the value of the LOGSIZE configuration parameter, only new log files are affected. The size of existing log files does not change. The total logical-log size is the product of the LOGSIZE configuration parameter setting multiplied by the value of the LOGFILES configuration parameter. However, if you change the value of the LOGSIZE configuration parameter, the total size of all logical log files depends on the number of log files of each size.

If the AUTO_LLOG configuration parameter is enabled, logical log files are added automatically as needed to improve performance, up to a configurable maximum total logical-log size.

To verify the page size that the database server uses on your platform, run the **onstat -b** command.

If you declare logging for a smart-large-object column, you must ensure that the logical log is considerably larger than the amount of data that is logged during inserts or updates. The database server cannot back up open transactions. If many transactions are active, the total logging activity must not force open transactions to the log backup files. For example, if your log size is 1000 KB and the high-watermark is 60 percent, do not use more than 600 KB of the logical log for the smart-large-object updates. The database server starts rolling back the transaction when it reaches the high-watermark of 600 KB.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
LTXHWM configuration parameter
onparams -p: Change physical-log parameters
Related information:
Determine database server page size
Size of the logical-log file
Estimate the size and number of log files

LOW_MEMORY_MGR configuration parameter

Use the LOW_MEMORY_MGR configuration parameter to enable automatic low memory management, which you can use to change the default behavior of a primary or standard server when it reaches its memory limit.

onconfig.std value
LOW_MEMORY_MGR 0
values
1 = Enables automatic low memory management when the database server starts.
0 = Disables automatic low memory management.

takes effect
After you edit your onconfig file and restart the database server.

Usage

If you configure a primary or standard server to use a percentage of the SHMTOTAL configuration parameter value for automatic low memory management start and stop thresholds, the SHMTOTAL configuration parameter must be set to a positive integer value.

Attention: Changing the value of the SHMTOTAL configuration parameter can cause the configuration of automatic low memory management to become invalid, forcing the database server to use default settings.
To enable automatic low memory management, specify:

```
LOW_MEMORY_MGR 1
```

Related reference:
SHMTOTAL configuration parameter
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)
onstat -g lmm command: Print low memory management information
Related information:
Reserve memory for critical activities

LOW_MEMORY_RESERVE configuration parameter

Use the LOW_MEMORY_RESERVE configuration parameter to reserve a specific amount of memory for use when critical activities are needed and the server has limited free memory.

If you enable the new LOW_MEMORY_RESERVE configuration parameter by setting it to a specified value in kilobytes, critical activities, such as rollback activities, can complete even when you receive out-of-memory errors.

onconfig.std value
LOW_MEMORY_RESERVE 0
values
0 or 128 - 2147483648, although the maximum value cannot be higher than 20 percent of the value of the SHMVIRTSIZE configuration parameter

units
kilobytes
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

No matter how the LOW_MEMORY_RESERVE configuration parameter is set, the maximum size of reserved memory is 20 percent of the value of the SHMVIRTSIZE configuration parameter.

For example, to reserve 512 kilobytes of memory, specify:

```
LOW_MEMORY_RESERVE 512
```

You can use the **onstat -g seg** command to view low-memory reserve information. The output includes lines that show the size of reserved memory, the number of times that the server has used the reserved memory, and the maximum memory needed.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g seg command: Print shared memory segment statistics
SHMVIRTSIZE configuration parameter

LTXEHWM configuration parameter

Use the LTXEHWM configuration parameter to specify the *long-transaction, exclusive-access, high-watermark*. When the logical-log space reaches the LTXEHWM threshold, the long transaction currently being rolled back is given *exclusive* access to the logical log.

onconfig.std value
LTXEHWM 80
if not present
90 (if DYNAMIC_LOGS is set to 1 or 2) 60 (if DYNAMIC_LOGS is set to 0)
range of values
LTXHWM through 100
units
Percent
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

A *transaction* is *long* if it is not committed or rolled back when it reaches the long-transaction high-watermark.

If your system runs out of log space before the rollback completes, lower the LTXEHWM value.

If you do not want too many logical logs to be added, LTXEHWM should be set to a smaller value (around 60). If dynamic logging is turned off (DYNAMIC_LOGS = 0), LTXEHWM should be set lower (around 50) to avoid running out of logical space.
Tip: To allow users to continue to access the logical logs, even during a long transaction rollback, set LTXEHWM to 100. Set DYNAMIC_LOGS to 1 or 2 so that the database server can add a sufficient number of log files to prevent long transactions from hanging and to allow long transactions to roll back.
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DYNAMIC_LOGS configuration parameter
LTXHWM configuration parameter
Related information:
Controlling long transactions

LTXHWM configuration parameter

Use the LTXHWM configuration parameter to specify the long-transaction high-watermark. The *long-transaction high-watermark* is the percentage of available log space that, when filled, triggers the database server to check for a long transaction.

onconfig.std value
LTXHWM 70
if not present
80 (if DYNAMIC_LOGS is set to 1 or 2) 50 (if DYNAMIC_LOGS is set to 0)
values
1 - 100
units

Percent

takes effect

After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When the logical-log space reaches the LTXHWM threshold, the database server starts rolling back the transaction. If you decrease the LTXHWM value, increase the size or number of log files to make rollbacks less likely.

If DYNAMIC_LOGS is set to 1 or 2, the database server can add a sufficient number of log files to complete the transactions or to prevent rollbacks from hanging when you have long transactions.

If you do not want too many logical logs to be added, LTXHWM should be set to a smaller value (around 60). If dynamic logging is turned off (DYNAMIC_LOGS = 0), LTXHWM should be set lower (around 50) to avoid running out of logical space.
Warning: If you set both LTXHWM and LTXEHWM to 100, long transactions are never aborted. Although you can use this configuration to your advantage, you should set LTXHWM to below 100 for normal database server operations.
If you set LTXHWM to 100, the database server issues a warning message:

```
LTXHWM is set to 100%. This long transaction high water mark
will never be reached. Transactions will not be aborted automatically
by the server, regardless of their length.
```

If the transaction hangs, follow the instructions for recovering from a long transaction hang, in the chapter on managing logical-log files in the *IBM® Informix Administrator's Guide*.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DYNAMIC_LOGS configuration parameter
LTXEHWM configuration parameter
LOGSIZE configuration parameter
Related information:
Controlling long transactions

MAX_FILL_DATA_PAGES configuration parameter

Use the MAX_FILL_DATA_PAGES configuration parameter to control inserting more rows to pages that have variable-length rows.

onconfig.std value

MAX_FILL_DATA_PAGES 0

values

0 or 1

units

Integer

takes effect

After you edit your onconfig file and restart the database server.

Usage

Set the MAX_FILL_DATA_PAGES value to 1 to allow more rows to be inserted per page in tables that have variable-length rows. This setting can reduce disk space, make more efficient use of the buffer pool, and reduce table scan times.

If MAX_FILL_DATA_PAGES is enabled, the server will add a new row to a recently modified page with existing rows if adding the row leaves at least 10 percent of the page free for future expansion of all the rows in the page. If MAX_FILL_DATA_PAGES is not set, the server will add the row only if there is sufficient room on the page to allow the new row to grow to its maximum length.

A possible disadvantage of enabling MAX_FILL_DATA_PAGES and allowing more variable-length rows per page is that the server might store rows in a different physical order. Also, as the page fills, updates made to the variable-length columns in a row could cause the row to expand so it no longer completely fits on the page. This causes the server to split the row onto two pages, increasing the access time for the row.

To take advantage of this setting, existing tables with variable-length rows must be reloaded or existing pages must be modified, followed by further inserts.

Related information:
Reduce disk space in tables with variable length rows

MAX_INCOMPLETE_CONNECTIONS configuration parameter

Use the MAX_INCOMPLETE_CONNECTIONS configuration parameter to specify the maximum number of incomplete connections in a session.

onconfig.std value

MAX_INCOMPLETE_CONNECTIONS 1024

units

Number of incomplete connections

takes effect

After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

After the number specified in the MAX_INCOMPLETE_CONNECTIONS configuration parameter is reached, an error message is written in the online message log stating that the server might be under a Denial of Service attack. See also information about the LISTEN_TIMEOUT configuration parameter, which specifies the number of seconds the server waits for a connection. .

Depending on the machine capability of holding the threads (in number), you can configure MAX_INCOMPLETE_CONNECTIONS to a higher value. Depending on the network traffic, you can also set the LISTEN_TIMEOUT configuration parameter, which specifies the number of seconds the server waits for a connection, to a lower value to reduce the chance that an attack can reach the maximum limit.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
LISTEN_TIMEOUT configuration parameter

MAX_PDQPRIORITY configuration parameter

Use the MAX_PDQPRIORITY configuration parameter to limit the PDQ resources that the database server can allocate to any one DSS query.

onconfig.std value

MAX_PDQPRIORITY 100

values

0 = Turns off PDQ. DSS queries use no parallelism.
1 = Fetches data from fragmented tables in parallel (parallel scans) but uses no other form of parallelism.

2 - 100 = Sets the percentage of the user-requested PDQ resources actually allocated to the query. 100 uses all available resources for processing queries in parallel.

takes effect

On all user sessions after you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

MAX_PDQPRIORITY is a factor that is used to scale the value of PDQ priority set by users. For example, suppose that the database administrator sets MAX_PDQPRIORITY to `80`. If a user sets the PDQPRIORITY environment variable to `50` and then issues a query, the database server silently processes the query with a PDQ priority of `40`.

You can use the **onmode** utility to change the value of MAX_PDQPRIORITY while the database server is online.

In IBM® Informix®, PDQ resources include memory, CPU, disk I/O, and scan threads. MAX_PDQPRIORITY lets the database administrator run decision support concurrently with OLTP, without a deterioration of OLTP performance. However, if MAX_PDQPRIORITY is too low, the performance of decision-support queries can degrade.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -D, -M, -Q, -S: Change decision-support parameters
onstat -g mgm command: Print MGM resource information
Related information:
Parallel database query (PDQ)
PDQPRIORITY environment variable

MIRROR configuration parameter

Use the MIRROR configuration parameter to enable or disable mirroring for the database server.

onconfig.std value

MIRROR 0

values

`0` = Disable mirroring
`1` = Enable mirroring

takes effect

After you edit your onconfig file and restart the database server.

Usage

It is recommended that you mirror the root dbspaces and the critical data as part of initialization. Otherwise, leave mirroring disabled. If you later decide to add mirroring, you can edit your configuration file to change the parameter value.

You do not have to set the MIRROR configuration parameter to the same value on both database servers in the high-availability data-replication pair. You can enable or disable mirroring on either the primary or the secondary database server independently. Do not set the MIRROR configuration parameter to 1 unless you are using mirroring.

Related reference:
onstat -d command: Print chunk information
Related information:
Mirroring

MIRROROFFSET configuration parameter

In IBM® Informix®, MIRROROFFSET specifies the offset into the disk partition or into the device to reach the chunk that serves as the mirror for the initial chunk of the root dbspace.

onconfig.std value
: MIRROROFFSET 0

values
: Any value greater than or equal to 0

units
: Kilobytes

takes effect
: After you edit your onconfig file and restart the database server.

Related information:
Mirroring the root dbspace during initialization

MIRRORPATH configuration parameter

Use the MIRRORPATH configuration parameter to specify the full path name of the mirrored chunk for the initial chunk of the root dbspace.

onconfig.std value
: On UNIX: $INFORMIXDIR/tmp/demo_on.root_mirror
On Windows: None

values
: 65 or fewer characters

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The MIRRORPATH should be a link to the chunk path name of the actual mirrored chunk for the same reasons that ROOTPATH is specified as a link. Similarly, select a short path name for the mirrored chunk.

You must set the permissions of the file that MIRRORPATH specifies to `660`. The owner and group must both be **informix**.

If you use raw disk space for your mirror chunk on a UNIX platform, it is recommended that you define MIRRORPATH as a path name that is a link to the initial chunk of the mirror dbspace, instead of entering the actual device name for the initial chunk.

To start mirroring data on a database server that is not running with the mirroring function enabled:

1. Take the database server offline.
2. Change the MIRROR configuration parameter to `1` and leave the MIRRORPATH configuration parameter blank.
3. Bring the database server online.
4. Allocate disk space for the mirror chunks. You can allocate this disk space at any time, however, the disk space must be available when you specify mirror chunks in the next step. The mirror chunks must be on a different disk than the corresponding primary chunks.
5. Specify the **onspaces -m** option to start mirroring for a dbspace, blobspace, or sbspace. You must begin with the root dbspace. After the root dbspace command is successfully run, the MIRRORPATH value is set automatically by the server.

Related information:
Mirroring the root dbspace during initialization
Manage disk space

MSG_DATE configuration parameter

Use the MSG_DATE configuration parameter to enable the insertion of a date in MM/DD/YY format at the beginning of each message printed to the online log.

onconfig.std value
Not in the onconfig.std file.
values
0 = OFF (the default)
1 = ON

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

In the following example MSG_DATE is set to 1 (ON).

```
04/10/11 10:26:06  Value of MSG_DATE has been changed to 1.
04/10/11 10:27:35  Value of MSG_DATE has been changed to 1.
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

MSGPATH configuration parameter

Use the MSGPATH configuration parameter to specify the full path name of the message-log file. The database server writes status messages and diagnostic messages to this file during operation.

onconfig.std value
On UNIX: $INFORMIXDIR/tmp/online.log
On Windows: %INFORMIXDIR%\online.log

On Windows, if you create a server instance during installation: %INFORMIXDIR%*server_name*.log. The *server_name* is the name of server in the program group and the value of the INFORMIXSERVER environment variable.

values
The path name of the online.log file.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If the file that MSGPATH specifies does not exist, the database server creates the file in the specified directory. If the directory that MSGPATH specifies does not exist, the database server sends the messages to the system console.

If the file that MSGPATH specifies does exist, the database server opens it and appends messages to it as they occur.

Related concepts:
Messages in the database server log
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

MULTIPROCESSOR configuration parameter

Use the MULTIPROCESSOR configuration parameter to specify whether the database server performs locking in a manner that is suitable for a single-processor computer or a multiprocessor computer.

If MULTIPROCESSOR is set to 0, the parameters that set processor affinity are ignored.

onconfig.std value
MULTIPROCESSOR 0
values
0 = No multiprocessor
1 = Multiprocessor available

takes effect
After you edit your onconfig file and restart the database server.

Related information:
CPU virtual processors

NET_IO_TIMEOUT_ALARM configuration parameter

Use the NET_IO_TIMEOUT_ALARM configuration parameter to control whether to be notified if network write operations have been blocked for 30 minutes or more.

Blocked network write operations usually indicate an operating system problem. Use the NET_IO_TIMEOUT_ALARM configuration parameter to enable event alarm 82 for specific types of network traffic.

onconfig.std value
: Not in onconfig.std

values
: One of the following values or a sum of one or more of the following values:

- 0 = Disabled
- 1 = Enabled for Enterprise Replication operations
- 2 = Enabled for distributed queries
- 4 = Enabled for HDR operations
- 8 = Enabled for SMX operations
- 16 = Enabled for other component operations

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

NETTYPE configuration parameter

Use the NETTYPE parameter to tune the network protocols that are defined in the sqlhosts information.

onconfig.std values
: UNIX: ipcshm,1,50,CPU
Windows: Not set.

default value
: *connection_type*,1,50,*vp_class*
The default connection type depends on the operating system:

- UNIX: The value of the **protocol** field from the sqlhosts file.
- Windows: `onsoctcp`

The default type of virtual processor class depends on the **dbservername** entry in the sqlhosts file:

- CPU, if the **dbservername** sqlhosts entry is defined by the DBSERVERNAME configuration parameter.
- NET, if the **dbservername** sqlhosts entry is defined by the DBSERVERALIASES configuration parameter.

separators
: Separate fields with commas. Do not include blank spaces. If you can omit values for fields, but you must include a comma for each field. However, you can omit trailing commas.

values
: See the Usage section.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The NETTYPE parameter provides tuning options for the protocol and interface combinations that are associated with **dbservername** entries in the sqlhosts information. Each **dbservername** entry in the sqlhosts information is defined on either the DBSERVERNAME configuration parameter or the DBSERVERALIASES configuration parameter in the onconfig file.

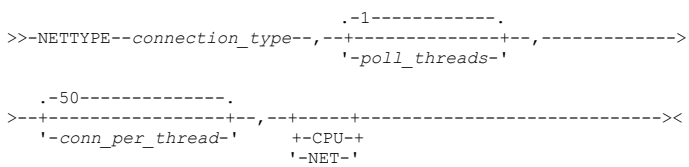


Table 1. Options for the NETTYPE configuration parameter value

Field	Values

Field	Values
connection_type	A valid protocol and interface combination, with or without the database server prefix of **on**, **ol**, or **dr**.
poll_threads	The number of poll threads that are assigned to the connection type. Default is 1. The range of values depends on the operating system and the virtual processor class: • UNIX: If the virtual processor class type is NET, an integer greater than or equal to 1. Each poll thread requires a separate virtual processor, so you indirectly specify the number of networking virtual processors when you specify the number of poll threads for an interface/protocol combination and specify that they are to be run by a network VP. • UNIX: If the virtual processor class is CPU, an integer from 1 through the number of CPU VPs. • Windows: An integer greater than or equal to 1. If your database server has many connections, you might be able to improve performance by increasing the number of poll threads. In general, each poll thread can handle approximately 200 - 250 connections. Windows: If you specify the soctcp protocol, only one poll thread is created, and instead, a socket I/O thread (soctcpio) is created in its own SOC VP for each poll thread that is specified by the NETTYPE parameter. Socket IO threads handle receive operations for all connections using I/O completion ports to receive completion notifications. These threads perform the bulk of the work of servicing network connections on Windows platforms.
conn_per_thread	An integer from 1 - 32767 that sets the maximum number of connections for each poll thread. Default is 50. For shared memory connections, the value of *conn_per_thread* is the maximum number of connections per thread. In general, specify double the number of expected connections. For network connections, the value of *conn_per_thread* can be exceeded. Poll threads dynamically reallocate resources to support more connections, as needed. Avoid setting the value for the number of concurrent connections much higher than you expect. Otherwise, you might waste system resources. If only a few connections are using a protocol concurrently, you might save memory by explicitly setting the estimated number of connections.
CPU	Specifies a CPU virtual processor. Configure shared memory connections to run in every CPU virtual processor.
NET	Specifies to use the appropriate network virtual processor: SOC, STR, SHM, or TLI. Configure network connection to run in network virtual processors.

You can specify a NETTYPE parameter for each protocol that you want the database server to use.

The following example illustrates NETTYPE parameters for two types of connections to the database server: a shared memory connection for local clients, and a network connection that uses sockets:

```
NETTYPE ipcshm,3,,CPU
NETTYPE soctcp,8,300,NET
```

The NETTYPE parameter for the shared-memory connection (ipcshm) specifies three poll threads to run in CPU virtual processors. The number of connections is not specified, so it is set to 50. For ipcshm, the number of poll threads correspond to the number of memory segments.

The NETTYPE parameter for the sockets connection (soctcp) specifies that 300 simultaneous connections are expected per thread for this protocol, and that 8 poll threads run in a network virtual processor.

UNIX: There can be a dependency between the NETTYPE and NUMFDSERVERS configuration parameter settings. When you have multiple CPU virtual processors and poll threads, and thread-status output from the **onstat -g ath** command indicates network shared file (NSF) locking, you can increase the NUMFDSERVERS value for poll threads to reduce NSF lock contention.

IBM Informix MaxConnect

If you are using IBM® Informix® MaxConnect, see the *IBM Informix MaxConnect User's Guide* for how to specify the fields in the NETTYPE parameter. The ontliimc and onsocimc protocols use TCP/IP to communicate with Informix MaxConnect. You can use these protocols to either connect Informix MaxConnect or the application clients to the database server.

Related reference:
DBSERVERNAME configuration parameter
DBSERVERALIASES configuration parameter
NUMFDSERVERS configuration parameter
VPCLASS configuration parameter
The number of configured inline poll threads exceeds the number of CPU virtual processors.
Virtual processor limit exceeded.
onstat -g nsc command: Print current shared memory connection information
onstat -g nsd command: Print poll threads shared-memory data
onstat -g nss command: Print shared memory network connections status
Related information:
Specifying the number of connections and poll threads
Run poll threads on CPU or network virtual processors
Specify the number of networking virtual processors
Connection information set in the NETTYPE configuration parameter
sqlhosts connectivity information
CPU virtual processors
Network virtual processors

NS_CACHE configuration parameter

Use the NS_CACHE configuration parameter to define the maximum retention time for entries in the Informix® name service caches: the host name/IP address cache, the service cache, the user cache, and the group cache.

onconfig.std value
: NS_CACHE host=900,service=900,user=900,group=900,sqlhosts=900

values
: Each of the fields takes an integer value equal to or greater than 0.
host = Sets the number of seconds to cache information in the host name or IP address cache.

service = Sets the number of seconds to cache information in the service cache.

user = Sets the number of seconds to cache information in the user cache.

group = Sets the number of seconds to cache information in the group cache.

sqlhosts = Sets the number of seconds to cache information in the sqlhosts cache.

0 = Caching is disabled. The server always gets information from the operating system. You can set an individual cache to 0 or set all name service caches to 0: `NS_CACHE 0`.

units
: Seconds

separators
: Separate values with a comma. Do not include blank spaces.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

For looking up and resolving host names (or IP addresses), service names, users (and passwords) or groups, the database server queries the operating system using appropriate system calls. Similarly, the information from the sqlhosts file is read every time it is needed. You can avoid many of these lookups and file reads by using the Informix name service caching mechanism, which can keep and reuse each retrieved piece of information for a configurable amount of time. You should set the NS_CACHE configuration parameter if your operating system does not provide its own caching.

For the sqlhosts cache, the file cache of the operating system can be an advantage, but the database server should benefit from the sqlhosts cache more as the open()/read()/close() can be a load for the operating system in a highly concurrent environment.

The server can get information from the cache faster than it does when querying the operating system. However, if you disable one or more of these caches by setting the retention time to 0, the database server queries the operating system for the host, service, user or group information and uses direct access to the sqlhosts file.

Changes that are made to name services at the operating system level are not immediately reflected in the Informix name server caches: for example, the change of an IP address, a user added to or removed from a group, or a new password. However, you can use the **onmode -wf** or **onmode -wm** command to change NS_CACHE information immediately. When you change the value for a particular cache with the **onmode -wf** or **onmode -wm** command, the server immediately expires all existing entries in that cache.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Name service maximum retention time set in the NS_CACHE configuration parameter
Improve connection performance and scalability

NUMFDSERVERS configuration parameter

For network connections on UNIX, use the NUMFDSERVERS configuration parameter to specify the maximum number of poll threads to handle network connections migrating between IBM® Informix® virtual processors (VPs).

Specifying NUMFDSERVERS information is useful if has a high rate of new connect and disconnect requests or if you find a high amount of contention between network shared file (NSF) locks. You can use the **onstat -g ath** command to display information about all threads. This information includes a status, such as `mutex wait nsf.lock`, which indicates that you have a significant amount of NSF lock contention.

onconfig.std value
: NUMFDSERVERS 4 (Only the first 4 poll threads of each **nettype** are involved in managing the connection migrations.)

values
: 1 - 50
The actual number depends on the number of poll threads, which you specify in the NETTYPE configuration parameter.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The specified value of NUMFDSERVERS has no effect on shared-memory (SHM) connections.

If you use the NUMFDSERVERS configuration parameter, also review, and if necessary, change the number of poll threads in the NETTYPE configuration parameter. For example, if you have multiple CPU VPs and poll threads and this results in NSF locking, you can increase NUMFDSERVERS and poll threads to reduce NSF lock contention.

Related reference:
NETTYPE configuration parameter
DBSERVERNAME configuration parameter
DBSERVERALIASES configuration parameter
onstat -g ath command: Print information about all threads
Related information:
Improve connection performance and scalability

OFF_RECVRY_THREADS configuration parameter

Use the OFF_RECVRY_THREADS configuration parameter to specify the number of recovery threads that are used for logical recovery during a cold restore or fast recovery.

onconfig.std value
: OFF_RECVRY_THREADS 10

values
: Positive integers

units
: Number of recovery threads that run in parallel

takes effect
: After you edit your onconfig file and restart the database server.

Usage

Before you perform a cold restore, you can set the value of this parameter to approximately the number of tables that have many transactions against them in the logical log. For single-processor computers or nodes, more than 30 to 40 threads might be too many because the cost of thread management and memory offsets the increase in parallel processing.

Whenever logical recovery begins, the database server creates an LGR memory pool for the recovery threads. The size of the LGR memory pool is approximately equal to the value of OFF_RECVRY_THREADS * 100 KB. This pool is used during fast recovery and during cold restores. Do not set the OFF_RECVRY_THREADS configuration parameter to a value that results in the database server attempting to allocate more memory for the LGR memory pool than is available on your system.

In a high-availability cluster, a secondary server is almost always in fast recovery mode. On secondary servers, set the OFF_RECVRY_THREADS configuration parameter to a value that takes both roll-forward performance and memory usage into account.

Related information:
OFF_RECVRY_THREADS and ON_RECVRY_THREADS and their effect on fast recovery
onbar -r syntax: Restoring data

ON_RECVRY_THREADS configuration parameter

The ON_RECVRY_THREADS configuration parameter is the maximum number of recovery threads that the database server uses for logical recovery when the database server is online (during a warm restore).

onconfig.std value
: ON_RECVRY_THREADS 1

values
: Positive integers

units
: Number of recovery threads that run in parallel

takes effect
: After you edit your onconfig file and restart the database server.

refer to

- *IBM® Informix Backup and Restore Guide*
- *IBM Informix Performance Guide*

Usage

You can tune ON_RECVRY_THREADS to the number of tables that are likely to be recovered, because the logical-log records that are processed during recovery are assigned threads by table number. The maximum degree of parallel processing occurs when the number of recovery threads matches the number of tables being recovered.

To improve the performance of warm restores, increase the number of fast-recovery threads with the ON_RECVRY_THREADS parameter.

Related information:
OFF_RECVRY_THREADS and ON_RECVRY_THREADS and their effect on fast recovery
onbar -r syntax: Restoring data

ONDBSPACEDOWN configuration parameter

Use the ONDBSPACEDOWN configuration parameter to define the action that the database server takes when any disabling event occurs on a primary chunk within a noncritical dbspace.

onconfig.std value
ONDBSPACEDOWN 2
values
0 = The database server marks the dbspace as offline and continues.
1 = The database server aborts.

2 = The database server writes the status of the chunk to the logs and waits for user input. If you set this option, but you want the database server to mark a disabled dbspace as down and continue processing, use **onmode -O** to override this ONDBSPACEDOWN setting.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

- Database Server Behavior When ONDBSPACEDOWN Does Not Apply

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -O: Override ONDBSPACEDOWN WAIT mode
Related information:
Monitor the database server for disabling I/O errors

Database Server Behavior When ONDBSPACEDOWN Does Not Apply

The database server will not come online if a chunk within any **critical** dbspace (for example, rootdbs or logsdbs) is missing.

The value of ONDBSPACEDOWN has no effect on temporary dbspaces. For temporary dbspaces, the database server continues processing regardless of the ONDBSPACEDOWN setting. If a temporary dbspace requires fixing, you should drop and recreate it.

For a non-primary chunk within a noncritical dbspace, the behavior of the database server depends on the transaction status of the chunk when the disabling event occurs:

- **No transaction**: If no transactions are detected against that chunk, the chunk is individually marked as down. In this case, subsequent attempts to write to that chunk fail, rolling back the associated transaction. You can safely put the chunk back and then use the **onspaces -s** utility to mark the chunk as back online.
- **Transaction detected**: If there are transactions to roll forward or back, then the database server aborts with an appropriate fast recovery error. In this case, you should put the chunk back and restart the database server.

ONLIDX_MAXMEM configuration parameter

Use the ONLIDX_MAXMEM configuration parameter to limit the amount of memory that is allocated to a single *preimage* pool and a single *updator* log pool.

onconfig.std value
ONLIDX_MAXMEM 5120
values
16 - 4294967295
units
Kilobytes
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The preimage and updator log pools, **pimage**_*partnum* and **ulog**_*partnum*, are shared memory pools that are created when a CREATE INDEX ONLINE statement is executed. The pools are freed when the execution of the statement is completed.

If you specify a value for this parameter and then create a table, add rows to the table, and start to execute a CREATE INDEX ONLINE statement on a column, you can also perform other operations on the column, such as running UPDATE STATISTICS HIGH, without having memory problems.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

OPTCOMPIND configuration parameter

Use the OPTCOMPIND to specify information that helps the optimizer choose an appropriate query plan for your application.

Tip: You can think of the name of the variable as arising from “OPTimizer COMPare (the cost of using) INDexes (with other methods).”

onconfig.std value
: OPTCOMPIND 2

values
: 0 = When appropriate indexes exist for each ordered pair of tables, the optimizer chooses index scans (nested-loop joins), without consideration of the cost, over table scans (hash joins). This value ensures compatibility with previous versions of the database server.
1 = The optimizer uses costs to determine an execution path if the isolation level is not Repeatable Read. Otherwise, the optimizer chooses index scans (it behaves as it does for the value 0). This setting is recommended for optimal performance.

2 = The optimizer uses cost to determine an execution path for any isolation level. Index scans are not given preference over table scans; the optimizer bases its decision purely on cost. This value is the default if the variable is not set.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Because of the nature of *hash joins*, an application with isolation mode set to Repeatable Read might *temporarily* lock all records in tables that are involved in the join (even those records that fail to qualify the join) for each ordered set of tables. This situation leads to higher contention among connections. Conversely, nested-loop joins lock fewer records but provide inferior performance when the database server retrieves a large number of rows. Thus, both join methods offer advantages and disadvantages. A client application can also influence the optimizer in its choice of a join method.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
OPTCOMPIND environment variable
OPTCOMPIND session environment option

OPT_GOAL configuration parameter

Use the OPT_GOAL configuration parameter to specify an optimization goal for queries.

onconfig.std value
: OPT_GOAL -1

values
: 0 or -1

takes effect
: After you edit your onconfig file and restart the database server.

Usage

A value of 0 sets the optimization goal to FIRST_ROWS. A value of -1 sets the optimization goal to ALL_ROWS, which is the default.

When you set the optimization goal to optimize for FIRST ROWS, you specify that you want the database server to optimize queries for perceived response time. In other words, users of interactive applications perceive response time as the time that it takes to display data on the screen. Setting the optimization goal to FIRST ROWS configures the database server to return the first rows of data that satisfy the query.

When you set the optimization goal to optimize for ALL ROWS, you specify that you want the database server to optimize for the total execution time of the query. Making ALL ROWS the optimization goal instructs the database server to process the total query as quickly as possible, regardless of how long it takes to return the first rows to the application.

You can specify the optimization goal in one of four ways:

- By query (SELECT statement)
  Use the ALL_ROWS and FIRST_ROWS directives.
- By session
  Use the SET OPTIMIZATION statement.
- By environment
  Set the OPT_GOAL environment variable.
- By database server
  Set the OPT_GOAL configuration parameter.

The list above lists the mechanisms for setting this goal in descending order of precedence. To determine the optimization goal, the database server examines the settings in the order above. The first setting encountered determines the optimization goal. For example, if a query includes the ALL_ROWS directive but the OPT_GOAL configuration parameter is set to FIRST_ROWS, the database server optimizes for ALL_ROWS, as the query specifies.

Related information:
OPT_GOAL environment variable (UNIX)
Optimization-Goal Directives
Optimization-goal directives

PC_HASHSIZE configuration parameter

Use PC_HASHSIZE to specify the number of hash buckets in the caches that the database server uses. PC_HASHSIZE applies to UDR cache only.

onconfig.std value
: PC_HASHSIZE 31

values
: Any positive integer, a prime number is recommended.

takes effect
: After you edit your onconfig file and restart the database server.

PC_POOLSIZE configuration parameter

Use the PC_POOLSIZE configuration parameter to specify the maximum number of user-defined routines that are stored in the UDR cache.

onconfig.std value
: PC_POOLSIZE 127

values
: A positive value 127 or greater that represents half of the initial maximum number of entries in the cache. The maximum value is dependent upon the shared memory configuration and available shared memory for the server instance.

takes effect
: After you edit your onconfig file and restart the database server.
: When you increase the value in memory by running the **onmode -wm** command.
: When you reset the value in memory by running the **onmode -wm** command.

The initial number of entries in the cache is twice the value of the PC_POOLSIZE configuration parameter. For example, if the PC_POOLSIZE configuration parameter is set to 127, 254 entries are allowed in the cache. If all entries in the cache are full, the cache size automatically grows by 10%. To reduce the size of the cache, decrease the value of the PC_POOLSIZE configuration parameter in the onconfig file and restart the server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

PFSC_BOOST configuration parameter

Use the PFSC_BOOST configuration parameter to enable or disable the boosted partition free space cache feature.

onconfig.std value
: PFSC_BOOST 1

values
: 0 = Boosted Partition Free Space Caches disabled
: 1 = Boosted Partition Free Space Caches enabled

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

Set the PFSC_BOOST configuration parameter to 0 to prevent boosted PFSCs from being created.

For more information, see Boosted Partition Free Space Caches (PFSC).

PHYSBUFF configuration parameter

Use the PHYSBUFF configuration parameter to specify the size in kilobytes of the two physical-log buffers in shared memory.

onconfig.std value
PHYSBUFF 128
units
Kilobytes
values
An integer in the range of 4 - (32767 * *pagesize* / 1024), where *pagesize* is the default system page size. The value must be evenly divisible by the default system page size. If the value is not evenly divisible by the page size, the database server rounds down the size to the nearest value that is evenly divisible by the page size.
takes effect
After you edit your onconfig file and restart the database server.

Usage

Double buffering permits user threads to write to the active physical-log buffer while the other buffer is being flushed to the physical log on disk. A write to the physical-log buffer is exactly one page in length. The value of the PHYSBUFF parameter determines how frequently the database server needs to flush the physical-log buffer to the physical-log file.

If the RTO_SERVER_RESTART configuration parameter is enabled, use the 512 kilobyte default value for PHYSBUFF. If the value of the PHYSBUFF configuration parameter is less than 512 kilobytes when the RTO_SERVER_RESTART configuration parameter is enabled, a warning message displays when you restart the server.

The user-data portion of a smart large object does not pass through the physical-log buffers.

Related reference:
onstat -l command: Print physical and logical log information
Related information:
Physical-log buffer

PHYSFILE configuration parameter

Use the PHYSFILE configuration parameter to specify the size of the physical log file when you first initialize the disk space and bring the database server online.

onconfig.std value
PHYSFILE 50000
if not present
200
values
An integer 200 or greater
units
KB
takes effect
After you edit the onconfig file and initialize disk space by running the **oninit -i** command.
After you run the **onparams -p -s** command.

Usage

You cannot change the value of the PHYSFILE configuration parameter by editing the onconfig file after you start the server for the first time.

The database server updates the value of the PHYSFILE configuration parameter in the onconfig file under the following circumstances:

- You change the size of the physical log file by running the **onparams -p -s** command.
- The plogspace is automatically expanded. If the physical log is stored in a plogspace, the database server expands the size of the physical log as needed to improve performance.

When the RTO_SERVER_RESTART or SEC_NONBLOCKING_CKPT configuration parameter is enabled, ensure that the size of the physical log is equal to at least 110% of the buffer pool size.

A warning message prints to the message log when:

- The value for the PHYSFILE configuration parameter is changed to less than 110% of all of the buffer pools
- The server is restarted
- A new buffer pool is added

Related reference:
onparams -p: Change physical-log parameters
RESTARTABLE_RESTORE configuration parameter
SDS_PAGING configuration parameter
Related information:
Strategy for estimating the size of the physical log
Change the physical-log location and size

PLOG_OVERFLOW_PATH configuration parameter

The PLOG_OVERFLOW_PATH parameter specifies the location of the file that is used during fast recovery if the physical log file overflows.

The file is plog_extend.*servernum* and by default located in $INFORMIXDIR/tmp. Use the full path name to specify a different location for the file with the PLOG_OVERFLOW_PATH parameter.

onconfig.std values
: On UNIX: $INFORMIXDIR/tmp
On Windows: None

takes effect
: When the database server is brought up (shared memory is initialized)

Related information:
Possible physical log overflow during fast recovery

PLCY_HASHSIZE configuration parameter

The PLCY_HASHSIZE configuration parameter specifies the number of hash buckets in the security policy information cache.

onconfig.std value
: PLCY_HASHSIZE 31

values
: Any positive integer

units
: KB

takes effect
: After you edit your onconfig file and restart the database server.

Related information:
Maintaining a label-based access-control implementation

PLCY_POOLSIZE configuration parameter

Use the PLCY_POOLSIZE configuration parameter to specify the maximum number of entries in each hash bucket of the security policy information cache.

onconfig.std value
: PLCY_POOLSIZE 127

values
: A positive value 127 or greater that represents half of the initial maximum number of entries in the cache. The maximum value is dependent upon the shared memory configuration and available shared memory for the server instance.

takes effect
: After you edit your onconfig file and restart the database server.
When you increase the value in memory by running the **onmode -wm** command.
When you reset the value in memory by running the **onmode -wm** command.

The initial number of entries in the cache is twice the value of the PLCY_POOLSIZE configuration parameter. For example, if the PLCY_POOLSIZE configuration parameter is set to 127, 254 entries are allowed in the cache. If all entries in a cache are full, the cache size automatically grows by 10%. To reduce the size of the cache, decrease the value of the PLCY_POOLSIZE configuration parameter in the onconfig file and restart the server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Maintaining a label-based access-control implementation

PN_STAGEBLOB_THRESHOLD configuration parameter

Use the PN_STAGEBLOB_THRESHOLD configuration parameter to reserve space for BYTE and TEXT data in round-robin fragments.

onconfig.std value
: Not set.

if not present
: 0

values
: 0 - 1000000

units
: Kilobytes

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Set this configuration parameter to the typical or average size of the BYTE or TEXT data that is stored in the table.

Restriction: The PN_STAGEBLOB_THRESHOLD configuration parameter has no effect if the number of extents has reached the maximum extents allowed or if the dbspace is full.
When a table reaches the maximum number of pages for a fragment, more pages can be added to the table by adding a new fragment. However, if a table contains BYTE or TEXT columns and that table is fragmented by the round-robin distribution scheme, adding a new fragment does not automatically enable new rows to be inserted into the new fragment.

For example, if one of the fragments in the table reaches the maximum number of pages, adding a new fragment does not extend the table to store more rows. Because BYTE and TEXT data tend to be large in size, the data is *staged* in one of the fragments before being distributed evenly in all of the fragments. The staging fragment must have sufficient space to store the BYTE or TEXT data. Use the PN_STAGEBLOB_THRESHOLD configuration parameter so that the database server can stage the BYTE or TEXT data temporarily in a staging fragment until the INSERT operation is completed and the data is permanently stored in the table.

During a UPDATE operation if the fragment does not have the space that is specified in PN_STAGEBLOB_THRESHOLD configuration parameter the table row that is impacted by the updated is moved into another fragment.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Fragmentation by ROUND ROBIN

PRELOAD_DLL_FILE configuration parameter

The PRELOAD_DLL_FILE configuration parameter specifies the path name for a shared library file that is preloaded when the database server is started.

onconfig.std value
: Not set. No shared library files are preloaded.

value
: *pathname* = Full path name for the shared library file. Can include $INFORMIXDIR.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

Use this parameter to preload the shared library files for DataBlade modules, built-in extensions, or user-defined routines that are created in the C programming language (C UDRs). Otherwise, the shared libraries are loaded when they are first used after the server starts, which affects performance. Add a separate entry of this parameter for each library file that you want to preload. A preloaded shared library remains active until the server is stopped.

Restriction: You cannot use the **onmode -wm** or **onmode -wf** commands to set the PRELOAD_DLL_FILE configuration parameter.

Examples

The following examples preload the built-in basic text search, spatial, and time series extensions:

```
PRELOAD_DLL_FILE $INFORMIXDIR/extend/bts.version/bts.bld

PRELOAD_DLL_FILE $INFORMIXDIR/extend/TimeSeries.version/TimeSeries.bld
```

The *version* is the specific version number for the extension. To find the correct version number, run the appropriate function to return the release number for the extension or check the directory name in your installation directory.

Important: The version numbers of built-in extensions can change in any fix pack or release. After you upgrade, you must update the value of the PRELOAD_DLL_FILE configuration parameter if the version number of an extension changed.
Related reference:
onstat -g dll command: Print dynamic link library file list

QSTATS configuration parameter

The QSTATS configuration parameter specifies the ability of **onstat -g qst** to print queue statistics.

onconfig.std value
: QSTATS 0

values
: 0 = Disable queue statistics
: 1 = Enable queue statistics

takes effect
: After you edit your onconfig file and restart the database server.

Related reference:

onstat -g qst command: Print wait options for mutex and condition queues

REMOTE_SERVER_CFG configuration parameter

Use the REMOTE_SERVER_CFG configuration parameter to specify the file that lists trusted remote hosts.

onconfig.std value
: Not set. The system hosts.equiv file is used.

values
: File name. The path is assumed to be $INFORMIXDIR/etc. Consider using the following naming convention:

  authfile.*server_name*

  The file that is specified by the REMOTE_SERVER_CFG configuration parameter must be in $INFORMIXDIR/etc.

takes effect
: After you edit your onconfig file and restart the database server.
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
  When you reset the value in memory by running the **onmode -wm** command.

Usage

For applications that connect to the database server as the **root** user, the REMOTE_SERVER_CFG configuration parameter is not applicable.

The format of the file that is specified by the REMOTE_SERVER_CFG configuration parameter is the same as the format of the system hosts.equiv file.

If the REMOTE_SERVER_CFG configuration parameter is not set, and you run the SQL administration API **task()** or **admin()** function with the **cdr add trustedhost** argument, the database server performs the following actions:

1. The REMOTE_SERVER_CFG configuration parameter is set to authfile.*DBSERVER*.
2. The authfile.*DBSERVER* file is created in $INFORMIXDIR/etc.
3. The specified trusted-host information is added to $INFORMIXDIR/etc/authfile.*DBSERVER*.
4. If the database server is part of a high-availability cluster, the trusted-host information is propagated to the trusted-host files of the other cluster servers.

Note: If the sqlhosts file of the database server uses the `s=6` option, you must also set the S6_USE_REMOTE_SERVER_CFG configuration parameter to `1` to use the file specified REMOTE_SERVER_CFG configuration parameter. Otherwise, the database server uses the system hosts.equiv file instead of the file specified REMOTE_SERVER_CFG configuration parameter.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
S6_USE_REMOTE_SERVER_CFG configuration parameter
cdr add trustedhost argument: Add trusted hosts (SQL administration API)
cdr remove trustedhost argument: Remove trusted hosts (SQL administration API)
cdr list trustedhost argument: List trusted hosts (SQL administration API)

Related information:
Trusted-host information
sqlhosts file and SQLHOSTS registry key options

REMOTE_USERS_CFG configuration parameter

Use the REMOTE_USERS_CFG configuration parameter to specify the file that lists the names of trusted users that exist on remote hosts.

onconfig.std value
: Not set.

values
: File name. The path is assumed to be $INFORMIX/etc.

takes effect
: After you edit your onconfig file and restart the database server.
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
  When you reset the value in memory by running the **onmode -wm** command.

Usage

The file specified by the REMOTE_USERS_CFG configuration parameter must be located in $INFORMIXDIR/etc. If the configuration parameter is set then the file specified is used instead of the ~/.rhosts file. If the specified file does not exist in $INFORMIXDIR/etc, then authentication will fail.

The format of the file specified by the REMOTE_USERS_CFG configuration parameter is the same as the format of the ~/.rhosts file.

Consider using the following naming convention for the file specified by the REMOTE_USERS_CFG configuration parameter:

users.*server_name*

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
REMOTE_USERS_CFG configuration parameter

Related information:
Trusted-user information

RESIDENT configuration parameter

Use the RESIDENT configuration parameter to specify whether resident and virtual segments of shared memory remain resident in operating-system physical memory.

onconfig.std value
RESIDENT 0
values
-1 - 99
0 = off

1 = lock the resident segment only

-1 = lock all resident and virtual segments

n = lock the resident segment and the next *n* -1 virtual segments. For example, if you specify 99 as the value, the resident segment is locked and the next 98 virtual segments are locked.

Certain platforms have different values. For information, see your machine notes.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Some systems allow you to specify that the resident portion of shared memory must stay (be resident) in memory at all times. If your operating system supports forced residency, you can specify that resident and virtual segments of shared memory not be swapped to disk.
Warning: Before you decide to enforce residency, verify that the amount of physical memory available is sufficient to execute all required operating-system and application processes. If insufficient memory is available, a system hang could result that requires a reboot.
On AIX®, Solaris, or Linux systems that support large pages of memory, the DBSA can use operating system commands to configure a pool of large pages.

IBM® Informix® can store non-message virtual memory segments on these large pages if you take the following steps:

- Enable large page sizes by setting the IFX_LARGE_PAGES environment variable.
- For virtual memory segments that you intend to store on large pages, set the RESIDENT parameter to lock those segments in physical memory, so that they cannot be swapped to disk

Storing virtual memory segments on large pages can offer significant performance benefits in large memory configurations.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -n, -r: Change shared-memory residency
Related information:
Resident portion of shared memory
Set database server shared-memory configuration parameters
IFX_LARGE_PAGES environment variable

RESTARTABLE_RESTORE configuration parameter

Use the RESTARTABLE_RESTORE configuration parameter to control whether the database server performs restartable restores.

onconfig.std value
RESTARTABLE_RESTORE ON
values
ON = Restartable restore is enabled
OFF = Restartable restore is disabled

takes effect
After you edit your onconfig file and restart the database server.

If you set RESTARTABLE_RESTORE to ON, you enable the database server to restart a failed physical or cold logical restore at the point at which the failure occurred. To perform a restartable restore with ON-Bar, use the **onbar -RESTART** command.

Increase the size of your physical log if you plan to use restartable restore. Although a restartable restore slows down the logical restore if many logs need to be restored, you save a lot of time from not having to repeat the entire restore.
Important: If the database server fails during a warm logical restore, you must repeat the entire restore. If the database server is still running, use **onbar -r -l** to complete the restore.
If you do a cold restore on systems that are not identical, you can assign new pathnames to chunks, and you can rename devices for critical chunks during the restore. You must perform a level-0 archive after the rename and restore operation completes.

The database server uses physical recovery and logical recovery to restore data as follows:

- **Physical recovery.** The database server writes data pages from the backup media to disk. This action leaves the storage spaces consistent to the point at which it was originally backed up. However, the backup times for each storage space are usually different. A restartable restore is restartable to the level of a storage space. If only some chunks of a storage space are restored when the restore fails, the entire storage space needs to be recovered again when you restart the restore.
- **Logical recovery.** The database server replays logical-log records on media to bring all the storage spaces up to date. At the end of logical recovery, all storage spaces are consistent to the same point.

Related reference:
PHYSFILE configuration parameter
Related information:
onbar -RESTART syntax: Restarting a failed restore

RESTORE_POINT_DIR configuration parameter

Use the RESTORE_POINT_DIR configuration parameter to change the path name of the directory where restore point files will be placed during a failed upgrade to a new version of the server. IBM® Informix® will store restore point files in a subdirectory of the specified directory, with the server number as the subdirectory name, only if the CONVERSION_GUARD configuration parameter is enabled.

onconfig.std value
: $INFORMIXDIR/tmp

value
: Complete path name for a directory

takes effect
: After you edit your onconfig file and restart the database server.

Usage

You can change the directory, for example, if you think that the $INFORMIXDIR/tmp directory does not have enough space for restore point data. If you want to change the directory, you must change it before you initiate an upgrade to a new version of the server. You cannot change the directory during an upgrade.

The directory specified in the RESTORE_POINT_DIR configuration parameter must be empty when an upgrade begins. If the directory contains any restore point files from a previous upgrade, you must remove the files before a new upgrade begins a new restore point.

Important:
The empty directory is a prerequisite before doing the upgrade, not when recovering from a failed upgrade. After a failed upgrade, do not empty the RESTORE_POINT_DIR directory before you attempt to run the **onrestorept** utility.

Related reference:
CONVERSION_GUARD configuration parameter
Related information:
The onrestorept utility

ROOTNAME configuration parameter

ROOTNAME specifies a name for the root dbspace for this database server configuration.

The name must be unique among all dbspaces that the database server manages. It is recommended that you select a name that is easily recognizable as the root dbspace.

onconfig.std value
: ROOTNAME rootdbs

values
: Up to 128 bytes. ROOTNAME must begin with a letter or underscore and must contain only letters, numbers, underscores, or **$** characters.

units
: A dbspace

takes effect
: When disk is initialized (destroys all data)

Related information:
Allocate disk space

ROOTOFFSET configuration parameter

ROOTOFFSET specifies the offset into an allocation of disk space (file, disk partition, or device) at which the initial chunk of the root dbspace begins.

UNIX Only:

On some UNIX platforms, it is not valid to set ROOTOFFSET to 0. When this parameter is set incorrectly, you must reinitialize disk space and reload data to resume proper operation of the database server. Before you configure the database server, always check your machine notes file for information about proper settings.

onconfig.std value
: ROOTOFFSET 0

values
: Any value greater than or equal to 0

units
: Kilobytes

takes effect
: When disk is initialized (destroys all data)

Related information:
Allocating raw disk space on UNIX

ROOTPATH configuration parameter

Use the ROOTPATH configuration parameter to specify the full path name, including the device or file name, of the initial chunk of the root dbspace. The ROOTPATH configuration parameter is stored in the reserved pages as a chunk name.

onconfig.std value

- On UNIX: $INFORMIXDIR/tmp/demo_on.rootdbs
- On Windows: None

values
: *pathname*

takes effect
: When disk is initialized (destroys all data)

refer to
: The following material in the chapter on managing disk space in the *IBM® Informix Administrator's Guide*

- Allocating disk space
- Creating links for raw devices

Usage

On UNIX, you must set the permissions of the file that you specify with the ROOTPATH configuration parameter to 660, and the owner and group must both be **informix**. On Windows, a member of the **Informix-Admin** group must own the file that you specify with the ROOTPATH configuration parameter.

UNIX Only:
If you use unbuffered disk space for your initial chunk on UNIX, you should define the ROOTPATH configuration parameter as a pathname that is a link to the initial chunk of the root dbspace instead of entering the actual device name for the initial chunk.

Related information:
Allocate disk space
Create symbolic links to raw devices (UNIX)

ROOTSIZE configuration parameter

Use the ROOTSIZE configuration parameter to specify the size in kilobytes of the initial chunk of the root dbspace. The size that you select depends on your immediate plans for your database server.

The database server uses the value of the ROOTSIZE configuration parameter only during a complete disk initialization. Changing the ROOTSIZE value after the initial chunk of the root dbspace has been created will have no effect.

onconfig.std value
: ROOTSIZE 300000

if not present
: 0

values
: 50,000 through maximum capacity of the storage device

units
: Kilobytes

takes effect
: When disk is initialized (destroys all data)

Related information:
Size of the root dbspace

RSS_FLOW_CONTROL configuration parameter

Specifies when flow control occurs in a high-availability cluster that contains at least one remote standalone (RS) secondary server.

onconfig.std value
RSS_FLOW_CONTROL 0

values
`0` = Flow control is activated when the difference between the current log position and the most recent acknowledged log exceeds 12 times the size of the log buffer.
`-1` = Flow control is disabled. Disabling flow control might lead to wrapping of the log files and the loss of data.

`start_value``,`*`end_value`* = The *start_value* and *end_value* determine the amount of lag between the current log position and the last acknowledged log page. The *start_value* must be greater than the *end_value*. Values must include one of the following units:

- `K` (Kilobytes)
- `M` (Megabytes)
- `G` (Gigabytes)

For example, setting `RSS_FLOW_CONTROL 128M,100M` starts flow control when the lag between the logs is 128 MB, and stops flow control when the lag drops to 100 MB.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Flow control provides a way to limit log activity on the primary server so that RS secondary servers in the cluster do not fall too far behind on processing transactions. Enabling flow control ensures that logs on RS secondary servers remain current if the servers are on a busy or intermittent network. When flow control is enabled, and when the difference in log size between the current log position and the last acknowledged log page exceeds the*start_value*, then log activity on the primary server becomes restricted. Users connected to the primary server may experience slower response time when flow control is active. Flow control is started when the lag between the logs is greater than the *start_value* and stops flow control when the log lag has dropped to the*stop_value*.

You set the RSS_FLOW_CONTROL configuration parameter on the primary server only. All RS secondary servers in the cluster are affected by the RSS_FLOW_CONTROL configuration parameter. Logs are always sent to the RS secondary server in the order in which they were received.

To check if flow control is active for a RS secondary server, use the **onstat -g rss verbose** command, and compare the `RSS flow control` value to the `Approximate Log Page Backlog` value. If the `Approximate Log Page Backlog` is higher than the first value of `RSS flow control`, flow control is active. If the `Approximate Log Page Backlog` is lower than the second value of `RSS flow control`, flow control is disabled.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
SDS_FLOW_CONTROL configuration parameter
Related information:
Flow control for shared-disk secondary servers
Flow control for remote standalone secondary servers

RSS_NONBLOCKING_CKPT configuration parameter

Use the RSS_NONBLOCKING_CKPT configuration parameter to enable non-blocking checkpoint at RS secondary server.

onconfig.std value
RSS_NONBLOCKING_CKPT 0

values

- 1 - Enable non-blocking checkpoint at RS secondary server
- 0 – (Default) Disable non-blocking checkpoints

units
Not applicable

takes effect
After you edit your onconfig file and restart the database server.

When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

When you reset the value in memory by running the **onmode -wm** command.

Usage

RSS_NONBLOCKING_CKPT configuration parameter controls the checkpoint behavior at RS secondary server.

RSS_NONBLOCKING_CKPT configuration parameter will be deprecated in future releases. User should use SEC_NONBLOCKING_CKPT instead.

Related reference:

SEC_NONBLOCKING_CKPT configuration parameter

RTO_SERVER_RESTART configuration parameter

Use the RTO_SERVER_RESTART configuration parameter to specify recovery time objective (RTO) standards for the amount of time, in seconds, that IBM® Informix® has to recover from a problem after you restart the server and bring it into online or quiescent mode.

onconfig.std value
: RTO_SERVER_RESTART 0 (disabled)

range of values
: 0 = disabled
: 60 - 1800

units
: seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
CKPTINTVL configuration parameter
The onparams Utility
Related information:
The oncheck -pr command
Checkpoints
Effect of configuration on I/O activity

S6_USE_REMOTE_SERVER_CFG configuration parameter

Use the S6_USE_REMOTE_SERVER_CFG configuration parameter to control whether the file specified by the REMOTE_SERVER_CFG configuration parameter is used to authenticate secure connections for server clusters and Enterprise Replication.

onconfig.std value
: S6_USE_REMOTE_SERVER_CFG 0

default value
: 0

values
: 0 = The system hosts.equiv file is used to authenticate servers connecting through a secure port.
: 1 = The file specified by the REMOTE_SERVER_CFG configuration parameter is used to authenticate servers connecting through a secure port.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The REMOTE_SERVER_CFG configuration parameter is used to specify a file that lists the remote server hosts that are trusted by the computer housing the database server. If one or more of the listed servers are configured using the sqlhosts file connection-security option `s=6`, then you must set the S6_USE_REMOTE_SERVER_CFG configuration parameter to `1`.

If S6_USE_REMOTE_SERVER_CFG is unset or set to `0`, the system hosts.equiv file, rather than the file specified by the REMOTE_SERVER_CFG configuration parameter, is used to authenticate servers connecting through a secure port.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
REMOTE_SERVER_CFG configuration parameter
Related information:
sqlhosts file and SQLHOSTS registry key options

SB_CHECK_FOR_TEMP configuration parameter

Use the SB_CHECK_FOR_TEMP configuration parameter to prevent the copying of a temporary smart large object into a permanent table.

onconfig.std value

Not set.

if value not present
The copying of temporary smart large objects into permanent tables is permitted.

values
0 = Permit the copying of temporary smart large objects into permanent tables. Equivalent to the configuration parameter not being set in the onconfig file.
1 = Prevent the copying of temporary smart large objects into permanent tables. The database server returns the following error messages instead of copying the handle of a temporary smart large object:

- -9810: Smart-large-object error.
- -12246: Smart large objects: You cannot put a temporary smart large object into a permanent table

takes effect
After you edit your onconfig file and restart the database server.

Usage

By default, you can copy temporary smart large objects into permanent tables. Smart large object data types, BLOB and CLOB, consist of two parts: the data, which is stored in an sbspace, and the handle, which is stored in a table. When you copy a temporary smart large object into a permanent table, only the BLOB or CLOB handle is copied into the permanent table. If you subsequently drop the temporary smart large object, the permanent table contains a handle that is no longer valid.

To prevent the copying of a temporary smart large object into a permanent table, set the SB_CHECK_FOR_TEMP configuration parameter to 1 in the onconfig file. For example, if the SB_CHECK_FOR_TEMP configuration parameter is set to 1, an INSERT INTO . . . SELECT FROM . . . statement that copies a temporary smart large object into a permanent table fails.

SBSPACENAME configuration parameter

Use the SBSPACENAME configuration parameter specifies the name of the default sbspace.

onconfig.std value
Not set.

if not present
0

values
Up to 128 bytes.
SBSPACENAME must be unique, begin with a letter or underscore, and contain only letters, digits, underscores, or **$** characters.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If your database tables include smart-large-object columns that do not explicitly specify a storage space, that data is stored in the sbspace that SBSPACENAME specifies.

The default sbspace is also used by the built-in encryption and decryption functions to store BLOB or CLOB values. If DECRYPT_BINARY or an encryption function cannot find an sbspace in which to store a BLOB or CLOB argument or returned value, the function fails with the following error message:

```
Fatal error in server row processing - SQL error -9810 ISAM error -12053
```

If you see this error message after you invoke an encryption or decryption function that has a CLOB or BLOB argument, configure a default sbspace using the SBSPACENAME configuration parameter, and then repeat the function call.
You must create the default sbspace with the **onspaces -c -S** utility before you can use it. The database server validates the name of the default sbspace when one of the following occurs:

- You specify the default sbspace as the storage option for a CLOB or BLOB column in the PUT clause of the CREATE TABLE or ALTER TABLE statement.
- The database server attempts to write a smart large object to the default sbspace when no sbspace was specified for the column.
- You store multirepresentational data in the default sbspace.

JAVA Language Support:
If you are using J/Foundation, you must provide a smart large object where the database server can store the Java™ archive (JAR) files. These JAR files contain your Java user-defined routines (UDRs). It is suggested that when you use Java UDRs, you create separate sbspaces for storing smart large objects.

Warning: When you use Enterprise Replication, you must set the CDR_QDATA_SBSPACE parameter and create the sbspace before you define the replication server.

Automatic creation of the default sbspace

Under certain circumstances, a default sbspace is created even if the SBSPACENAME configuration parameter is not set:

- If you create a **bts** index and do not explicitly specify an sbspace name
- If you create a table with a spatial data type column and do not explicitly specify an sbspace name

The default sbspace is created in the root dbspace for the database server with a size of 10 000 KB. You must manually increase the size of the default sbspace when it fills.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

SBSPACETEMP configuration parameter
SYSSBSPACENAME configuration parameter
Sbspace Structure
onspaces -c -S: Create an sbspace
Related information:
Sbspaces
Alter storage characteristics of smart large objects
PUT Clause
Row Data sbspaces

SBSPACETEMP configuration parameter

Use the SBSPACETEMP configuration parameter to specify a list of default temporary sbspace for storing temporary smart large objects without metadata or user-data logging. If you store temporary smart large objects in a standard sbspace, the metadata is logged.

onconfig.std value
: Not set. Temporary smart large objects are stored in the default sbspace, which is specified by the SBSPACENAME configuration parameter.

separators
: Commas

values
: One or more sbspace names. Separate names with a comma. The length of the list cannot exceed 128 bytes.
Each sbspace name must be unique, begin with a letter or underscore, and contain only letters, digits, underscores, or **$** characters.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
SBSPACENAME configuration parameter
onspaces -c -S: Create an sbspace
Related information:
Temporary sbspaces
Creating a temporary sbspace
Temporary smart large objects

SDS_ALTERNATE configuration parameter

Use the SDS_ALTERNATE configuration parameter to define an alternate means of communication between the primary server and SD secondary servers in a high-availability cluster.

onconfig.std value
: NONE (No SD secondary server alternate communication path is configured.)

values
: The name of the blobspace that is to be used as the alternate communication path between the primary server and SD secondary servers.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

You set the SDS_ALTERNATE configuration parameter and create a shared blobspace to allow the primary server and all SD secondary servers in a high-availability cluster to use an alternate communication path in the event the network is unavailable between the primary server and the SD secondary servers. When an SD secondary server is about to failover and become the primary server, but TCP/IP communication is unavailable, the shared blobspace set by the SDS_ALTERNATE configuration parameter is used communicate the shut-down procedure to the original primary.

Set the SDS_ALTERNATE configuration parameter to the same value on the primary server and on all SD secondary servers.

Before setting the SDS_ALTERNATE configuration parameter, you must create the shared blobspace on the primary server. For example, to create a blobspace named **sds_alt_comm** enter the following command on the primary server:

```
onspaces -c -b sds_alt_comm -g <pagesize> -p <path> -o <offset> -s <size>
```

Run the following command to switch to the next logical log file so that the newly created blobspace is usable:

```
onmode -l
```

On each of the SD secondary servers in the high-availability cluster, set the SDS_ALTERNATE configuration parameter to point to the blobspace on the primary server.

```
SDS_ALTERNATE sds_alt_comm
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
The oninit utility
Related information:
SD secondary server

SDS_ENABLE configuration parameter

Use the SDS_ENABLE configuration parameter to enable SD secondary server functionality.

onconfig.std value
Not set.
if not present
0
values
0 = Disable
1 = Enable

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

You must set SDS_ENABLE to 1 (enable) on the SD secondary server to enable SD secondary server functionality.

SDS_ENABLE is set to 1 (enabled) automatically when you run the following command:

```
onmode -d set SDS primary
```

SDS_ENABLE is set to 0 (disabled) when you run the following command:

```
onmode -d clear SDS primary
```

To prevent data corruption, you cannot use the **oninit -i** or **oninit -iy** command to initialize disk space on a server if SDS_ENABLE is set to 1 (enabled). To initialize an SD secondary server, initialize only the shared memory by using **oninit** with no parameters. To initialize a primary server to which one or more SD secondary servers are attached, and whose disk has never been initialized, set SDS_ENABLE to 0 and initialize the server memory and disk using **oninit -i**. To initialize a primary server to which SD secondary servers are attached, and whose disk is already initialized, set SDS_ENABLE to 1 and initialize shared memory only using **oninit** with no parameters.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

SDS_FLOW_CONTROL configuration parameter

Specifies when flow control occurs in a high-availability cluster that contains at least one shared-disk (SD) secondary server.

onconfig.std value
SDS_FLOW_CONTROL 0
values
0 = Flow control is activated when the difference between the current log position and the most recent acknowledged log exceeds 12 times the size of the log buffer.
-1 = Flow control is disabled. Disabling flow control might lead to wrapping of the log files and the loss of data.

start_value,*end_value* = The *start_value* and *end_value* determine the amount of lag between the current log position and the last acknowledged log page. The *start_value* must be greater than the *end_value*. Values must include one of the following units:

- K (Kilobytes)
- M (Megabytes)
- G (Gigabytes)

For example, setting SDS_FLOW_CONTROL 128M,100M starts flow control when the lag between the logs is 128 MB, and stops flow control when the lag has dropped to 100 MB.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

Usage

Flow control provides a way to limit log activity on the primary server so that SD secondary servers in the cluster do not fall too far behind on processing transactions. When flow control is enabled, and when the difference in log size between the current log position and the last acknowledged log page exceeds the*start_value*, then log activity on the primary server becomes restricted. Users connected to the primary server may experience slower response time when flow control is active. Flow control is started when the lag between the logs is greater than the *start_value* and stops flow control when the log lag has dropped to the*stop_value*.

You set the SDS_FLOW_CONTROL configuration parameter on the primary server only. All SD secondary servers in the cluster are affected by the SDS_FLOW_CONTROL configuration parameter. Logs are always sent to the SD secondary server in the order in which they were received.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
RSS_FLOW_CONTROL configuration parameter
Related information:
Flow control for remote standalone secondary servers
Flow control for shared-disk secondary servers

SDS_LOGCHECK configuration parameter

Use the SDS_LOGCHECK configuration parameter to set the number of seconds to delay the secondary server from taking over the role of the primary server. If the secondary server detects that the primary server is generating log records during the delay period, then the failover is prevented. The delay can prevent an unnecessary failover if network communication between the primary and secondary servers is temporarily unavailable.

onconfig.std value
: SDS_LOGCHECK

  - On UNIX: 10
  - On Windows: 0

values
: 0 = Do not detect log activity; allow immediate failover.
n = Wait up to *n* seconds. If log activity is detected from the primary server, failover is prevented; otherwise, failover is allowed.

units
: Seconds

takes effect
: When shared disk functionality is enabled on the primary server

Usage

Important: You must specify the same value for the primary server and for all secondary servers. If the values that you specify are not the same, the database server automatically changes the value that is different on a secondary server to the value that is set for the primary server.
For example, if the SDS_LOGCHECK configuration parameter is set to 10, and the primary server fails, the SD secondary server waits up to 10 seconds to either detect that the primary server is generating log records (in which case failover is prevented), or the SD secondary server detects that the primary is not generating log records and failover occurs.

An unnecessary failover can result in two primary servers that are both receiving input from applications and writing to the same chunks, which can cause unrepairable data corruption.

Set the SDS_LOGCHECK configuration parameter to a value greater than zero if you do not have I/O fencing configured and your system consists of a primary server and one or more SD secondary servers.

If your system has I/O fencing configured, and if an SD secondary server becomes a primary server, the I/O fencing script must prevent the failed primary server from updating any of the shared disks. If the system does not have I/O fencing configured, the SDS_LOGCHECK configuration parameter prevents the occurrence of multiple primary servers by not failing over to the SD secondary server if the original primary server is generating log records.

Related information:
SD secondary server

SDS_PAGING configuration parameter

The SDS_PAGING configuration parameter specifies the location of two files that serve as buffer paging files.

onconfig.std value
: Not set

Values
: File paths

Separators
: A single comma

Default value
: None

Takes effect
: When SD secondary server is started

Usage

The SDS_PAGING configuration parameter must be set to a valid value to ensure that the SD secondary server starts. Because the paging files grow dynamically as needed, you should allocate enough disk space to store two times the size of the value specified by the PHYSFILE configuration parameter.

Example

In the following example, the files page1 and page2 are set as the buffer paging files for the SD secondary server.

```
SDS_PAGING /usr/informix/tmp/page1,/usr/informix/tmp/page2
```

Related reference:
PHYSFILE configuration parameter
Related information:
SD secondary server

SDS_TEMPDBS configuration parameter

Use the SDS_TEMPDBS configuration parameter to specify information that the shared disk (SD) secondary server uses to dynamically create temporary dbspaces. This configuration parameter can be specified only on the SD secondary server.

onconfig.std value
: Not set. Temporary dbspaces for shared disk secondary servers are not created.

values
: A string containing the following values in the following order, separated by commas:
dbspace = The name of the dbspace to create. Must be unique among all existing dbspaces, blobspaces, and sbspaces, including those any temporary spaces that are inherited from a primary server. The name cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character.

dbpath = The path for the dbspace, either a full path name or a relative path name. If you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server.

pagesize = An integer representing the page size of the dbspace, in kilobytes. The page size must be between 2 KB and 16 KB and must be a multiple of the default page size.

offset = An integer equal to or greater than 0 that specifies offset into the disk partition or into the device to reach the initial chunk of the dbspace. The starting offset plus the chunk size cannot exceed the maximum chunk size. The offset must be a multiple of the page size. The maximum offset is 2 or 4 terabytes, depending on the platform. By default, the value is in kilobytes. You can designate different units by appending a single character modifier to the value: `M` or `m` for megabytes, `G` or `g` for gigabytes, or `T` or `t` for terabytes.

size = A positive integer equal to or greater than 1000 kilobytes and a multiple of the page size that specifies the size of the initial chunk of the dbspace. The value of *offset* plus the value of *size* cannot exceed the maximum chunk size. The maximum size of a chunk is equal to 2 147 483 647 pages multiplied by the page size. By default, the value is in kilobytes. You can designate different units by appending a single character modifier to the value: `M` or `m` for megabytes, `G` or `g` for gigabytes, or `T` or `t` for terabytes.

separators
: Separate each value with a comma. Do not use blank spaces.

takes effect
: After you edit your onconfig file and restart the SD secondary server.

Usage

The temporary dbspaces are created, or initialized if the dbspaces existed previously, when the SD secondary server starts. The temporary dbspaces are used for creating temporary tables. There must be at least one occurrence of the SDS_TEMPDBS configuration parameter in the onconfig file of the SD secondary server for the SD secondary server to start. You can specify up to 16 SD secondary temporary dbspaces in the onconfig file by using multiple occurrences of the SDS_TEMPDBS configuration parameter.

For each occurrence of the SDS_TEMPDBS configuration parameter in the onconfig file:

- The *dbsname* value must unique for each server and not shared with any other SD secondary server or the primary server.
- The combination of *dbspath*, *pagesize*, *offset*, and *size* must not cause any overlap with existing chunks or between temporary dbspaces spaces specified by the SDS_TEMPDBS configuration parameter.
- The *pagesize* value must be the same for each SDS_TEMPDBS configuration parameter value.

The following example shows two entries for the SDS_TEMPDBS configuration parameter:

```
SDS_TEMPDBS sds_space1,/dev/raw_dev1,2,0,60M
SDS_TEMPDBS sds_space2,/dev/raw_dev2,2,0,80M
```

If the primary server in a high-availability cluster fails and an SD secondary server takes over as the primary server, then the value set for the SDS_TEMPDBS configuration parameter on the SD secondary server is used for temporary dbspaces until the server is restarted. You should ensure that the value specified for the SDS_TEMPDBS configuration parameter on the SD secondary server is different than the value specified on the primary server. After the SD secondary server is restarted, the DBSPACETEMP configuration parameter is used.

Related information:
Shared disk secondary servers

SDS_TIMEOUT configuration parameter

Use the SDS_TIMEOUT configuration parameter to specify the amount of time in seconds that the primary server in a high-availability cluster will wait for a log-position acknowledgment to be sent from a shared disk (SD) secondary server.

onconfig.std value
: SDS_TIMEOUT 20

if not present
: 10

values
: 2 - 2147483647

units
: seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the SDS_TIMEOUT value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the SDS_TIMEOUT value in memory by running the **onmode -wm** command.

Usage

If no log-position acknowledgment is received from the SD secondary server in the specified amount of time, the primary server will disconnect from the SD secondary server and continue. After waiting for the number of seconds specified in the SDS_TIMEOUT configuration parameter setting, the primary server will start removing SD secondary servers, if page flushing has timed out while waiting for an SD secondary server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Shared disk secondary servers

SEC_APPLY_POLLTIME configuration parameter

Use the SEC_APPLY_POLLTIME configuration parameter to control how long log replay thread should poll for new work before yielding.

onconfig.std value
: SEC_APPLY_POLLTIME 0

values

- Minimum value: (Default) 0
- Recommended value for smaller systems (between 1 to 8 CPUVPs): 0
- Recommended value for medium systems (between 8 and 16 CPUVPS): 10
- Recommended value for larger systems (> 16 CPUVPS): 1000
- Maximum value: 5000

units
: micro seconds.

takes effect
: After you edit your onconfig file and restart the database server.

: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

: When you reset the value in memory by running the **onmode -wm** command.

Usage

In micro seconds, controls how long log replay thread should poll for new work before yielding. Use this parameter to reduce thread context switch overhead while replaying log records. It is recommended to configure poll threads to run on NET VP if SEC_APPLY_POLLTIME value > 0. For more information see, NETTYPE configuration parameter.

SEC_DR_BUFS configuration parameter

Use the SEC_DR_BUFS configuration parameter to control the number of replication buffers to be used for replicating log records to secondary server. Buffer size is same as LOGBUFF config value.

onconfig.std value
: SEC_DR_BUFS 12

values

- Minimum value: (Default) 12
- Maximum value: 128
- Recommended value: Between 12 and 24

units
Number of replication buffers. Buffer size is same as LOGBUFF config value.

takes effect
After you edit your onconfig file and restart the database server.

Usage

SEC_DR_BUFS configuration parameter control the number of replication buffers to be used for replicating log records to secondary server. Buffer size is same as LOGBUFF config value.

SEC_LOGREC_MAXBUFS configuration parameter

Use the SEC_LOGREC_MAXBUFS configuration parameter to control the number of log buffers to be used for replaying log records at secondary server. Each log buffer is of size 16KB.

onconfig.std value
SEC_LOGREC_MAXBUFS 1000

values

- Minimum value: (Default) 5 times OFF_RECVRY_THREADS config parameter value
- Maximum value: Do not set to more than 2000 buffers
- Recommended value: 1000

units
Number of 16KB buffers

takes effect
After you edit your onconfig file and restart the database server.

Usage

SEC_LOGREC_MAXBUFS configuration parameter control the number of log buffers to be used for replaying log records at secondary server.

SEC_NONBLOCKING_CKPT configuration parameter

Use the SEC_NONBLOCKING_CKPT configuration parameter to enable non-blocking checkpoint at HDR and RS secondary server.

onconfig.std value
SEC_NONBLOCKING_CKPT 0

values

- 1 - Enable non-blocking checkpoint at HDR and RS secondary server
- 0 – (Default) Disable non-blocking checkpoints

units
Not applicable

takes effect
After you edit your onconfig file and restart the database server.

When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

When you reset the value in memory by running the **onmode -wm** command.

Usage

SEC_NONBLOCKING_CKPT configuration parameter controls the checkpoint behavior at HDR and RS secondary server.

When SEC_NONBLOCKING_CKPT configuration parameter is enabled, HDR secondary applies blocking or non-blocking checkpoint exactly like the Primary server.

When SEC_NONBLOCKING_CKPT configuration parameter is enabled, ensure that the size of the physical log is equal to at least 110% of the buffer pool size.

Related reference:
PHYSFILE configuration parameter

SECURITY_LOCALCONNECTION configuration parameter

Use the SECURITY_LOCALCONNECTION configuration parameter to verify security on local connections by verifying that the ID of the local user who is running a program is the same ID of the user who is trying to access the database.

onconfig.std value
: Not set.

values
: `0` = No security checking occurs.
`1` = IBM® Informix® checks whether the ID of the user who is running the program matches the ID of the user who is trying to connect to the database.

`2` = same as `1`, plus retrieves the peer port number from the network API and verifies that the connection is coming from the client program. You can only specify two if your system has SOCTCP or IPCSTR network protocols.

takes effect
: After you edit your onconfig file and restart the database server.

Related information:
Set database server shared-memory configuration parameters

SEQ_CACHE_SIZE configuration parameter

Use the SEQ_CACHE_SIZE configuration parameter to specify the maximum number of sequence objects that are cached in memory.

onconfig.std value
: SEQ_CACHE_SIZE 10

values
: 1 - 2147483647

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When the maximum number of sequence objects are cached, the database server attempts to remove entries for any sequence objects that are no longer referenced.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

SERVERNUM configuration parameter

The SERVERNUM configuration parameter specifies a relative location in shared memory.

onconfig.std value
: SERVERNUM 0

values
: 0 - 255

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The value that you choose must be unique for each database server on your local computer. The value does not need to be unique on your network. Because the value `0` is included in the onconfig.std file, it is suggested that you choose a value other than `0` to avoid the inadvertent duplication of the SERVERNUM configuration parameter.

Related information:
Set database server shared-memory configuration parameters

SESSION_LIMIT_LOCKS configuration parameter

The SESSION_LIMIT_LOCKS configuration parameter specifies the maximum number of locks available in a session. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

onconfig.std value

none
if not present
2147483647
values
500 - 2147483647
units
Number of locks in the internal lock table
takes effect
After you edit your onconfig file and restart the database server.

Usage

For massively lock-intensive operations, administrators can set SESSION_LIMIT_LOCKS to reduces the risk of ordinary users in concurrent sessions depleting the lock resources of the database server.

The database server terminates a transaction that exceeds the limit of the number of locks, puts a message in the database server message log, and triggers the event alarm 21014.

Important:
In repeatable read isolation level, because each row in the active set requires a lock, be careful about setting too low a limit for locks on the server. Similarly, setting too small a lock limit can interfere with Enterprise Replication tasks or with **cdr** commands issued by non-DBSA users.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
IFX_SESSION_LIMIT_LOCKS session environment option
Managing tenant databases
Limit session resources

SESSION_LIMIT_LOGSPACE configuration parameter

The SESSION_LIMIT_LOGSPACE configuration parameter specifies the maximum amount of log space that a session can use for individual transactions. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

onconfig.std value
0 (off)
if not present
0 (off)
values
5120 - 2147483648
units
KB
takes effect
After you edit your onconfig file and restart the database server.

Usage

The SESSION_LIMIT_LOGSPACE configuration parameter limits how much log space a session can use for each transaction, and can conserve system resources within a tenant-database environment.

The database server terminates a transaction that exceeds the log space limit, puts a message in the database server message log, and triggers the event alarm 21018.

The `session_limit_logspace` tenant database property set through the **tenant create** or **tenant update** SQL API command takes precedent over the SESSION_LIMIT_LOGSPACE configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
Managing tenant databases
Limit session resources

SESSION_LIMIT_MEMORY configuration parameter

The SESSION_LIMIT_MEMORY configuration parameter specifies the maximum amount of memory that a session can allocate. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

onconfig.std value
0 (off)
if not present

0 (off)

values
20480 - 2147483648

units
KB

takes effect
After you edit your onconfig file and restart the database server.

Usage

The SESSION_LIMIT_MEMORY configuration parameter limits how much memory a session can allocate, and can prevent individual sessions from monopolizing system resources.

The database server terminates a session that exceeds the memory limit, puts a message in the database server message log, and triggers the event alarm 21016.

The `session_limit_memory` tenant database property set through the **tenant create** or **tenant update** SQL API command takes precedent over the SESSION_LIMIT_MEMORY configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
Managing tenant databases
Limit session resources

SESSION_LIMIT_TEMPSPACE configuration parameter

The SESSION_LIMIT_TEMPSPACE configuration parameter specifies the maximum amount of temporary table space that a session can allocate. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

onconfig.std value
0 (off)

if not present
0 (off)

values
20480 - 2147483648

units
KB

takes effect
After you edit your onconfig file and restart the database server.

Usage

The SESSION_LIMIT_TEMPSPACE configuration parameter limits how much temporary table space a session can allocate, and can conserve system resources within a tenant-database environment.

The database server terminates a session that exceeds the space limit, puts a message in the database server message log, and triggers the event alarm 21017.

The `session_limit_tempspace` tenant database property set through the **tenant create** or **tenant update** SQL API command takes precedent over the SESSION_LIMIT_TEMPSPACE configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
Managing tenant databases
Limit session resources

SESSION_LIMIT_TXN_TIME configuration parameter

The SESSION_LIMIT_TXN_TIME configuration parameter specifies the maximum amount of time that a transaction can run in a session. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

onconfig.std value
0 (off)

if not present
0 (off)

values
1 - 2147483647

units
Seconds

takes effect
After you edit your onconfig file and restart the database server.

Usage

The SESSION_LIMIT_TXN_TIME configuration parameter limits how much time a transaction can run in a session, and can prevent individual session transactions from monopolizing the logical log.

The database server terminates a transaction that exceeds the time limit, puts a message in the database server message log, and triggers the event alarm 21019.

The `session_limit_txn_time` tenant database property set through the **tenant create** or **tenant update** SQL API command takes precedent over the SESSION_LIMIT_TXN_TIME configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
Managing tenant databases
Limit session resources

SHMADD configuration parameter

Use the SHMADD configuration parameter to specify the size of the segments that are dynamically added to the virtual portion of shared memory.

onconfig.std value
Platform dependent
values
32-bit platforms: 1024 - 524288
64-bit platforms: 1024 - 4294967296

units
KB
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The value of the SHMADD configuration parameter represents the size of the first set of segments that the database server adds to the virtual portion of shared memory when additional memory is needed. The size of the first virtual shared memory segment is set by the SHMVIRTSIZE configuration parameter. Set the values of the SHMVIRTSIZE and SHMADD configuration parameters so that a minimal number of segments are added during the normal operation of the database server. In general, more segments impair performance.

The maximum number of Informix® shared memory segments is 1024. Many shared memory segments might be required if the SHMADD value is low or the database server has unexpectedly large amounts of activity or memory use. To prevent the database server from reaching the maximum number of shared memory segments, the size of virtual segments that are added dynamically by the server doubles every 16 virtual segments. It is more efficient to add memory in large segments, but wasteful if the added memory is not used. Also, the operating system might require you to add memory in a few large segments rather than many small segments.

The following table contains recommendations for setting the initial value of SHMADD.

Table 1. Recommended SHMADD values

Amount of physical memory	Recommended SHMADD value
Less than 256 MB	8192
256 - 512 MB	16,384
Greater than 512 MB	32,768

You can view information about virtual memory segments by running the **onstat -g seg** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g seg command: Print shared memory segment statistics
EXTSHMADD configuration parameter
Related information:
Virtual portion of shared memory
Monitor shared-memory segments

SHMBASE configuration parameter

Use the SHMBASE configuration parameter to specifiy the base address where shared memory is attached to the memory space of a virtual processor.

onconfig.std value
: Platform dependent

values
: Positive integers

units
: Address

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The addresses of the shared-memory segments start at the SHMBASE value and grow until the upper-bound limit, which is platform specific.

Do not change the value of SHMBASE. The onconfig.std value for SHMBASE depends on the platform and whether the processor is 32-bit or 64-bit. For information on which SHMBASE value to use, see the machine notes.

Related reference:
onstat -g seg command: Print shared memory segment statistics
Related information:
Set operating-system shared-memory configuration parameters

SHMNOACCESS configuration parameter

The SHMNOACCESS configuration parameter specifies a virtual memory address range to not use to attach shared memory.

onconfig.std values
: On UNIX: None
On Windows: #SHMNOACCESS 0x70000000-0x7FFFFFFF, and this value is commented out in the onconfig.std template file.

values
: 1 - 10 address ranges

separators
: Comma

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The SHMNOACCESS configuration parameter is used to avoid specific range process addresses, which in turn avoids conflicts with operating system libraries.

Each address in each range must start in hexadecimal format. Each address in a range must be separated by a hyphen and each range must be separated by a comma, as the following example shows:

```
SHMNOACCESS 0x70000000-0x75000000,
0x7A000000-0x80000000
```


SHMTOTAL configuration parameter

Use the SHMTOTAL configuration parameter to specify the total amount of shared memory (resident, virtual, communications, and virtual extension portions) to be used by the database server for all memory allocations. The onconfig.std value of 0 implies that no limit on memory allocation is stipulated.

onconfig.std value
: SHMTOTAL 0

values
: 0 = (no specific limit) or any integer greater than or equal to 1

units
: Kilobytes

takes effect
: After you edit your onconfig file and restart the database server.

Usage

You can use the SHMTOTAL configuration parameter to limit the demand for memory that the database server can place on your system. However, applications might fail if the database server requires more memory than the limit imposed by SHMTOTAL. When this situation occurs, the database server writes the following message in the message log:

```
size of resident + virtual segments xx + yy > zz total allowed by
configuration parameter SHMTOTAL
```

This message includes the following values.

Value

Description

xx
: Current® size of resident segments

yy
: Current size of virtual segments

zz
: Total shared memory required

If you enabled the LOW_MEMORY_MGR configuration parameter and are configuring the server to use a percentage of the SHMTOTAL configuration parameter value for automatic low memory management start and stop thresholds, the SHMTOTAL configuration parameter must not be set to 0 (unilmited).

Attention: Changing the value of the SHMTOTAL configuration parameter value can cause the configuration of automatic low memory management to become invalid, forcing the database server to use the default settings.
UNIX Only:
Set the operating-system parameters for maximum shared-memory segment size, typically SHMMAX, SHMSIZE, or SHMALL, to the total size that your database server configuration requires. For information about the amount of shared memory that your operating system allows, see the machine notes.

If you have more physical memory than the value specified in the machine notes, and the memory is to be used by IBM® Informix®, you can increase the value of the SHMALL parameter to as much 90 percent of the physical memory that is specified for your computer. It is recommended that you do not meet or exceed the available RAM.

Related reference:
DS_TOTAL_MEMORY configuration parameter
LOW_MEMORY_MGR configuration parameter
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
Related information:
Shared memory
Shared-memory size

SHMVIRT_ALLOCSEG configuration parameter

Use the SHMVIRT_ALLOCSEG configuration parameter to specify a threshold at which Informix® should allocate a new shared memory segment and the level of the event alarm activated if the server cannot allocate the new memory segment.

onconfig.std value
: SHMVIRT_ALLOCSEG 0,3

values
: A numeric value optionally followed by a comma and another numeric value.
threshold = A number that indicates when the database server should add a shared memory segment:

- 0 = Default. The database server allocated shared memory segments when needed.
- .40 - .99 = The percentage of memory used before a segment is added.
- 256 - 10000000 = The number of kilobytes remaining before a segment is added.

alarm_level: Optional. An integer value from 1 to 5 that specifies the level of the event alarm to raise: 1 = Not noteworthy, 2 = Information, 3 = Attention (Default), 4 = Emergency, 5 = Fatal. The event alarm has a class ID of 24 and an event ID of 24003.

separator
: Separate the values with a comma.

takes effect
: After you edit your onconfig file and restart the database server.

Usage

Set the SHMVIRT_ALLOGSEG configuration parameter to proactively add shared memory segments instead of waiting until the database server automatically adds shared memory segments.

The event alarm repeats every thirty minutes if a new memory segment cannot be allocated.

Related reference:
Event Alarm Parameters
Related information:
The SHMVIRT_ALLOCSEG configuration parameter and memory utilization

SHMVIRTSIZE configuration parameter

Use the SHMVIRTSIZE configuration parameter to specify the initial size of a virtual shared-memory segment.

onconfig.std value
: Platform dependent

if not present
: If SHMADD is present: the value of the SHMADD configuration parameter.
If SHMADD is not present: 8192.

values
32-bit platforms: Positive integer with a maximum value of 2 GB
64-bit platforms: Positive integer with a maximum value of 4 TB

The maximum value might be less on some platforms due to operating-system limitations. For the actual maximum value for your UNIX platform, see the machine notes.

units
KB
takes effect
After you edit your onconfig file and restart the database server.

Usage

To determine the appropriate value for the SHMVIRTSIZE configuration parameter, use the following algorithm to determine the size of the virtual portion of shared memory:

```
shmvirtsize = fixed overhead + ((stack size + heap) * number of users)
```

Variable	Value to use
fixed overhead	This includes the size of the AIO vectors, sort memory, dbspace backup buffers, dictionary size, size of stored-procedure cache, histogram pool, other pools, and other overhead. To obtain an estimate of the fixed overhead, start the database server and see how many additional memory segments are allocated, if any. The number of users that you have on the system when you start the server, impacts the allocation of memory segments. When you start the server: • If the number of users is typical for your environment, then add the size of the memory segments to the current value for the SHMVIRTSIZE configuration parameter and restart the server. • If the number of users is far less than what is typical for your environment, you must calculate the appropriate overhead value to use for the memory segments. You can determine how many memory segments each user consumes by dividing the number of additional memory segments that are allocated when you started the server by the number of users that you had on the server then. Multiply the value for the memory segments for each user by the number of users that you typically have on the system. Add this calculated value for the memory segments to the current value for SHMVIRTSIZE configuration parameter and restart the server.
stack size	On 32-bit systems, use 32 KB for the stack size. Typically on 64-bit systems, you use 64 KB for the stack size. However, some 64-bit systems use a different value.
heap	Use 30 KB per user.
number of users	Use the maximum number of concurrent user sessions that you anticipate on the server.

If possible, create a virtual portion of shared memory of a size that is more than you require for your daily processing.

Use the **onstat -g seg** command to determine peak usage and lower the value of the SHMVIRTSIZE configuration parameter accordingly.

Related reference:
DS_TOTAL_MEMORY configuration parameter
onstat -g seg command: Print shared memory segment statistics
STACKSIZE configuration parameter
LOW_MEMORY_RESERVE configuration parameter
Related information:
Virtual portion of shared memory
Effect of configuration on memory utilization

SINGLE_CPU_VP configuration parameter

The SINGLE_CPU_VP configuration parameter specifies whether or not the database server is running with only one CPU virtual processor.

onconfig.std value
SINGLE_CPU_VP 0
values
0 = running with multiple CPU VPs
1 = running with one CPU VP

takes effect
When the database server is shut down and restarted

Usage

Disable the SINGLE_CPU_VP configuration parameter by setting it to 0 if you want the number of CPU VPs to be automatically increased when the database server starts.

Setting SINGLE_CPU_VP to nonzero allows the database server to use optimized code based on the knowledge that only one CPU virtual processor is running. It enables the database server to bypass many of the mutex calls that it must use when it runs multiple CPU virtual processors.

It is strongly recommended that you set this parameter when the database server will run only one CPU virtual processor. Depending on the application and workload, setting this parameter can improve performance by up to 10 percent.

If you set SINGLE_CPU_VP to nonzero and try to add a CPU virtual processor, you receive one of the following messages:

```
onmode: failed when trying to change the number of classname VPs by n.
onmode: failed when trying to change the number of cpu virtual processors by n.
```

If you set SINGLE_CPU_VP to nonzero and then attempt to bring up the database server with VPCLASS **cpu,** *num* set to a value greater than 1, you receive the following error message, and the database server initialization fails:

```
Cannot have SINGLE_CPU_VP non-zero and CPU VPs greater than 1.
```

- VPCLASS Values and the SINGLE_CPU_VP Configuration Parameter
  Informix treats user-defined virtual-processor classes as if they were CPU virtual processors. If you set the *SINGLE_CPU_VP* configuration parameter to a nonzero value, you cannot create any user-defined virtual-processor classes.

Related information:
Run on a single-processor computer

VPCLASS Values and the SINGLE_CPU_VP Configuration Parameter

Informix® treats user-defined virtual-processor classes as if they were CPU virtual processors. If you set the *SINGLE_CPU_VP* configuration parameter to a nonzero value, you cannot create any user-defined virtual-processor classes.

Using a user-defined VPCLASS

If you set this configuration parameter to a nonzero value and then attempt to bring up the database server with a user-defined VPCLASS, you receive the following error message, and the database server initialization fails:

```
oninit: Cannot have SINGLE_CPU_VP non-zero and user-defined VP classes
```

Using the *cpu* VPCLASS

If you set this configuration parameter to a nonzero value and then attempt to bring up the database server with the VPCLASS *cpu* value for *num* set to a value greater than 1, you receive the following error message, and the database server initialization fails:

```
Cannot have SINGLE_CPU_VP non-zero and CPU VPs greater than 1.
```


SMX_COMPRESS configuration parameter

Use the SMX_COMPRESS configuration parameter to specify the level of compression that the database server uses before sending data from the source database server to the target database server.

Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data. The SMX_COMPRESS configuration parameter values of the two servers are compared and changed to the higher compression values.

onconfig.std value
: SMX_COMPRESS 0

values
: -1 = The source database server never compresses the data, regardless of whether or not the target site uses compression.
0 = The source database server compresses the data only if the target database server expects compressed data.

1 = The database server performs a minimum amount of compression.

9 = The database server performs the maximum possible compression.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

SMX_NUMPIPES configuration parameter

The SMX_NUMPIPES configuration parameter sets the number of pipes for server multiplexer group (SMX) connections.

onconfig.std value
: SMX_NUMPIPES 1

values
: 1 - 32767 = The number of network pipes for SMX connections.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

High-availability clusters and parallel sharded queries use SMX connections. If the lag time between servers is too long, increase the number of SMX pipes.

Note:
From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.

SMX_PING_INTERVAL configuration parameter

Use the SMX_PING_INTERVAL configuration parameter to specify the number of seconds in a timeout interval, where a secondary server waits for activity from the primary server in a Server Multiplexer Group (SMX) connection.

onconfig.std value
SMX_PING_INTERVAL 10
values
`0` = Wait indefinitely.
A positive integer between `1` and `60`, inclusive. = The number of seconds in the timeout interval.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
After you run the SQL administration API **task()** or **admin()** function with the `"onmode","-wf SMX_PING_INTERVAL=value"` or `"onmode","-wm SMX_PING_INTERVAL=value"` argument.

Usage

If the secondary server does not receive any message during the length of time that is specified by the SMX_PING_INTERVAL configuration parameter and after the number of intervals that are specified by the SMX_PING_RETRY configuration parameter, the secondary server prints an error message to the online.log and closes the SMX connection. If an SMX timeout message is in the online.log, you can increase the SMX_PING_INTERVAL value, the SMX_PING_RETRY value, or both of these values.

This configuration parameter applies only to secondary servers. If you set SMX_PING_INTERVAL on the primary server, it becomes effective if the primary server becomes a secondary server.

If the onconfig file of a secondary server in a high-availability cluster has the following entries, the secondary server waits a total of 180 seconds for activity from the primary server. If there is no activity from the primary server during those 180 seconds, the secondary server closes the SMX connection and writes an error message to the online log.

```
SMX_PING_INTERVAL 30
SMX_PING_RETRY 6
```

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.
Related reference:
SMX_PING_RETRY configuration parameter
Related information:
Set the wait time for SMX activity between servers

SMX_PING_RETRY configuration parameter

Use the SMX_PING_RETRY configuration parameter to specify the maximum number of times that a secondary server repeats the timeout interval that is specified by the SMX_PING_INTERVAL configuration parameter if a response from the primary server is not received. If the maximum number is reached without a response, the secondary server prints an error message in the online.log and closes the Server Multiplexer Group (SMX) connection.

onconfig.std value
SMX_PING_RETRY 6
values
Any positive integer = The maximum number of times to repeat the timeout interval.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
After you run the SQL administration API **task()** or **admin()** function with the After you run the SQL administration API **task()** or **admin()** function with the `"onmode","-wf SMX_PING_RETRY=value"` or `"onmode","-wm SMX_PING_RETRY=value"` argument.

Usage

If the secondary server does not receive any message during the length of time that is specified by the SMX_PING_INTERVAL configuration parameter and after the number of intervals that are specified by the SMX_PING_RETRY configuration parameter, the secondary server prints an error message to the online.log and closes the SMX connection. If an SMX timeout message is in the online.log, you can increase the SMX_PING_INTERVAL value, the SMX_PING_RETRY value, or both of these values.

This configuration parameter applies only to secondary servers. If you set SMX_PING_RETRY on the primary server, it becomes effective if the primary server becomes a secondary server.

If the onconfig file of a secondary server in a high-availability cluster has the following entries, the secondary server waits a total of 60 seconds for activity from the primary server. If there is no activity from the primary server during those 60 seconds, the secondary server closes the SMX connection and writes an error message to the online log.

```
SMX_PING_INTERVAL 12
SMX_PING_RETRY 5
```

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.
Related reference:
SMX_PING_INTERVAL configuration parameter
Related information:
Set the wait time for SMX activity between servers

SP_AUTOEXPAND configuration parameter

Use the SP_AUTOEXPAND configuration parameter to enable or disable the automatic creation or extension of chunks.

onconfig.std value
SP_AUTOEXPAND 1
values
0 = The automatic creation or extension of chunks is not enabled.
1 = The automatic creation or extension of chunks is enabled.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When the SP_AUTOEXPAND configuration parameter is enabled and a storage container such as a dbspace has a defined create size or extend size that is not zero, the container is auto-expandable.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Automatic space management

SP_THRESHOLD configuration parameter

Use the SP_THRESHOLD configuration parameter to define the minimum amount of free kilobytes that can exist in a storage space before IBM® Informix® automatically runs a task to expand the space, either by extending an existing chunk in the space or by adding a new chunk.

onconfig.std value
SP_THRESHOLD 0
values
0 = No threshold. The trigger that runs the storage space monitoring (**mon_low_storage**) task for adding space when space is below the threshold is disabled.
1 - 50 = A threshold that is a percentage of free kilobytes in a storage space.

If the value is 50 or below, interprets the value as a percentage (for example, 10 = 10 percent and 2.84 = 2.84 percent).

1000 to the maximum size of a chunk = A threshold that is either 1000 kilobytes or the maximum size of the chunk on the current platform.

If the value is 1000 or higher, interprets the value as a specific number of kilobytes.

Values 50 - 1000 are not valid.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When you set the SP_THRESHOLD configuration parameter to a valid value that is greater than 0, the built-in Scheduler task, **mon_low_storage**, runs automatically when the free space in a dbspace, temporary dbspace, sbspace, temporary sbspace, or blobspace falls below the threshold.

Suppose the value of the SP_THRESHOLD configuration parameter value is 5.5, which the server interprets as 5.5 percent. If a space runs low on free pages, and the free space percentage falls below 5.5 percent and remains below that level until the **mon_low_storage** task runs next, that task will attempt to expand the space. If the SP_THRESHOLD configuration parameter is set to 50000 and a space has fewer than 50000 free kilobytes, that space will be expanded the next time **mon_low_storage** task runs.

A value of 0 turns off the **mon_low_storage** task, and prevents the server from extending any space. However, a value of 0 does not affect the ability of the server to extend a space when all free pages are depleted and more are needed.

The value specified in the SP_THRESHOLD configuration parameter applies to all spaces belonging to the server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Automatic space management

SP_WAITTIME configuration parameter

Use the SP_WAITTIME configuration parameter to specify the maximum number of seconds that a thread waits for a dbspace, temporary dbspace, plogspace, sbspace, temporary sbspace, or blobspace space to expand before returning an out-of-space error.

onconfig.std value
: SP_WAITTIME 30

values
: 0 - 2147483647

units
: seconds

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The time that the server uses to automatically add or expand a chunk can vary widely, depending on various factors such as the size of the chunk, the speed of the associated disk drives, and the load on the system. When IBM® Informix® automatically adds or expands a chunk to prevent free space from falling below the threshold specified by the SP_THRESHOLD configuration parameter, forces threads that need the space to wait until it is available. You can change the value of the SP_WAITTIME configuration parameter if you want to change the maximum amount of time that the thread will wait for more space.

A thread will wait for a storage space to expand only if the storage pool contains entries. A thread will not wait if the storage pool is empty.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Automatic space management

SQL_LOGICAL_CHAR configuration parameter

Use the SQL_LOGICAL_CHAR configuration parameter to enable or disable the expansion of size specifications in declarations of built-in character data types.

onconfig.std value
: SQL_LOGICAL_CHAR OFF (= interpret size specifications in units of bytes)

values
: `OFF` = No expansion of declared sizes.
: `1` = No expansion of declared sizes.
: `2` = Use `2` as the expansion factor for declared sizes.
: `3` = Use `3` as the expansion factor for declared sizes.
: `4` = Use `4` as the expansion factor for declared sizes.
: `ON` = Use *M* as the expansion factor, where *M* is the maximum length in bytes that any logical character requires in the code set of the current database. Depending on the DB_LOCALE setting, *M* has an integer range from `1` (in single-byte locales) up to `4`.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

For applications that are developed in single-byte locales, but deployed in multibyte locales, this feature can reduce the risk of multibyte logical characters being truncated during data entry operations.

In a multibyte code set, such as **UTF-8** or the multibyte code sets for some East Asian languages, a single logical character can require more than one byte of storage. The setting of this parameter can instruct the SQL parser to apply logical-character semantics to declarations of these built-in character data types:

- BSON
- CHAR
- CHARACTER
- CHARACTER VARYING
- JSON
- LVARCHAR
- NCHAR
- NVARCHAR
- VARCHAR
- DISTINCT types that declare any of these data types as their base types
- ROW types (named and unnamed) that include fields of these data types
- Collection types (LIST, MULTISET, or SET) that include these types as elements.

The setting that you specify for this parameter must be one of the following values:

Whether the SQL_LOGICAL_CHAR configuration parameter is set to enable or disable the expansion of declared storage sizes, its setting specifies how data type declarations are interpreted for all sessions of the IBM® Informix® instance.

Automatic Resizing of the Expansion Factor

When SQL_LOGICAL_CHAR is set to a valid digit, and the current session creates a database, compares the SQL_LOGICAL_CHAR value with the maximum number of bytes that any logical character will use for the code set of the database.

If the SQL_LOGICAL_CHAR setting is greater than that maximum number of bytes, the database uses the maximum value for the locale as the new expansion factor, overriding what the configuration file specifies. The SQL_LOGICAL_CHAR setting in the configuration file remains unchanged, and continues to act as the default expansion factor for other user databases.

Similarly, if the SQL_LOGICAL_CHAR value for a session is automatically reset to a digit, as described above, but the same session subsequently connects to another database whose locale uses a code set in which a logical character requires a larger storage size than the current expansion factor, uses the maximum number of bytes for the new code set as the new expansion factor while the user session is connected to that database, rather than using the current setting of SQL_LOGICAL_CHAR.

Automatic resetting of the expansion factor to match the largest logical character size in the code set that DB_LOCALE specifies at connection time also occurs when SQL_LOGICAL_CHAR is set to `ON`, but the effects of the `ON` setting are not identical to the database server behavior when SQL_LOGICAL_CHAR is set to a digit (1, 2, 3, or 4) in two ways:

- The expansion factor can be automatically reset to a smaller value if `ON` is the SQL_LOGICAL_CHAR setting.
- There is no difference between `SQL_LOGICAL_CHAR = 4` and `SQL_LOGICAL_CHAR = ON`.

You must set SQL_LOGICAL_CHAR to `ON`, rather than to a digit, if you want a smaller expansion factor when the current session connects to a database whose largest logical character in the DB_LOCALE code set requires a smaller number of bytes than the current SQL_LOGICAL_CHAR setting. The effective expansion factor will always be less than or equal to the maximum character size for a locale.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
SYSTABLES
Single-byte and multi-byte characters and locales
Data definition statements

SQLTRACE configuration parameter

Use the SQLTRACE parameter to control the startup environment of SQL tracing.

onconfig.std value
: On UNIX: Not set. SQL tracing is not enabled.
On Windows: #SQLTRACE level=low,ntraces=1000,size=2,mode=global

values
: See the Usage section.

takes effect
: After you edit your onconfig file and restart the database server.
After you run the SQL administration API **task()** or **admin()** function with the **set sql tracing** argument.

Usage

Remove the # symbol from the onconfig value to retain basic information, up to 2 KB in size, about the last 1000 SQL statements that were run by any user. You can customize the scope of the SQL tracing information by adjusting the field values of the SQLTRACE configuration parameter.

```
Syntax for the SQLTRACE configuration parameter

>>-SQLTRACE--+------------------------+------------------------>
             |           .-low----.    |
```

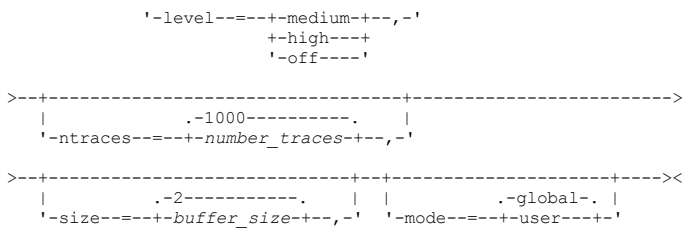


Table 1. Options for the SQLTRACE configuration parameter value

Field	Values
level	Amount of information traced: • `Low` = Default. Captures statement statistics, statement text, and statement iterators. • `Medium` = Captures all of the information included in low-level tracing, plus table names, the database name, and stored procedure stacks. • `High` = Captures all of the information included in medium-level tracing, plus host variables. • `Off` = Specifies no SQL tracing.
ntraces	The *number_traces* value is the number of SQL statements to trace before reusing the resources. Default is 1000. The range is 500 - 2147483647.
size	The *buffer_size* value is the maximum size of variable length data to be stored, in KB. Default is 2. The range is 1 -100. If this buffer size is exceeded, the database server discards saved data.
mode	Scope of tracing performed: • `Global` = Default. All users. • `User` = Users who have tracing enabled by an SQL administration API **task()** or **admin()** function. Specify this mode if you want to get a sample of the SQL that a small set of users is running.

The **onstat -g his** command displays SQL trace information.

Related reference:
onstat -g his command: Print SQL trace information
set sql tracing argument: Set global SQL tracing (SQL administration API)
Related information:
Specifying startup SQL tracing information by using the SQLTRACE configuration parameter

SSL_KEYSTORE_LABEL configuration parameter

Use the SSL_KEYSTORE_LABEL configuration parameter to specify the label of the server digital certificate used in the keystore database, a protected database that stores SSL keys and digital certificates.

onconfig.std value
: Not set.

values
: Up to 512 characters for the label of the database server certificate used in Secure Sockets Layer (SSL) protocol communications

takes effect
: After you edit your onconfig file and restart the database server.

Usage

When setting up the database server keystore that contains the digital key and certificates for SSL communication, the digital certificate of the database server is given a label (i.e. a name) in the keystore. Use this label as the value for the SSL_KEYSTORE_LABEL parameter.

For more information on the keystore for SSL communication and configuration parameters that you need to set, see the *IBM® Informix Security Guide*.

Related information:
Secure sockets layer protocol

STACKSIZE configuration parameter

Use the STACKSIZE configuration parameter to specify the stack size for the database server user threads.

onconfig.std value
: STACKSIZE 32 for 32-bit database servers
STACKSIZE 64 for 64-bit database servers

values

32 through limit determined by the database server configuration and the amount of memory available

units
Kilobytes

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The value of STACKSIZE does not have an upper limit, but setting a value that is too large wastes virtual memory space and can cause swap-space problems.

For 32-bit platforms, the default STACKSIZE value of 32 kilobytes is sufficient for nonrecursive database activity. For 64-bit platforms, the recommended STACKSIZE value is 64 kilobytes. When the database server performs recursive database tasks, as in some SPL routines, for example, it checks for the possibility of stack-size overflow and automatically expands the stack.

User threads execute user-defined routines. To increase the stack size for a particular routine, use the **stack** modifier on the CREATE FUNCTION statement.
Warning: Setting the value of STACKSIZE too low can cause stack overflow, the result of which is undefined but usually undesirable.
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
SHMVIRTSIZE configuration parameter
Related information:
Stacks
INFORMIXSTACKSIZE environment variable
CREATE FUNCTION statement

STATCHANGE configuration parameter

Use the STATCHANGE configuration parameter to specify a positive integer for a global percentage of a change threshold for the server to use to determine if distribution statistics qualify for an update when the automatic mode for UPDATE STATISTICS operations is enabled.

onconfig.std value
STATCHANGE 10

values
0 - 100

units
percentage of a change threshold

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The database server uses the value of the STATCHANGE configuration parameter when the AUTO_STAT_MODE configuration parameter, the AUTO_STAT_MODE session environment variable, or the AUTO keyword of the UPDATE STATISTICS statement has enabled the automatic mode for UPDATE STATISTICS operations.

The STATCHANGE setting specifies a change threshold for the database server to use to determine if distribution statistics qualify for an update when the automatic mode for UPDATE STATISTICS operations is enabled. When this mode is enabled, the UPDATE STATISTICS statement compares the STATCHANGE setting with the percentage of rows that have changed in each table or fragment since the current data distributions were calculated, and selectively updates only the missing or stale distribution statistics for each table or fragment within the scope of the UPDATE STATISTICS statement.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
AUTO_STAT_MODE configuration parameter
Related information:
Statistics options of the CREATE TABLE statement

STMT_CACHE configuration parameter

Use the STMT_CACHE configuration parameter to determine whether the database server uses the SQL statement cache.

onconfig.std value
`STMT_CACHE 0`

values
`0` = SQL statement cache not used (equivalent to **onmode -e** `OFF`).
`1` = SQL statement cache enabled, but user sessions do not use the cache. Users use the cache only if they set the environment variable STMT_CACHE to `1` or execute the SQL statement SET STATEMENT CACHE ON.

`2` = SQL statement cache turned on. All statements are cached. To turn off statement caching, set the environment variable STMT_CACHE to `0` or execute the SQL statement SET STATEMENT CACHE OFF.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

You can enable the SQL statement cache in one of two modes:

- Always use the SQL statement cache unless a user explicitly specifies not to use it. Set the STMT_CACHE configuration parameter to `2` or **onmode -e** `ON`.
- Use the SQL statement cache only when a user explicitly specifies to use it. Set the STMT_CACHE configuration parameter to `1` or **onmode -e** `ENABLE`.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -e: Change usage of the SQL statement cache
Related information:
STMT_CACHE environment variable
Using the SQL statement cache

STMT_CACHE_HITS configuration parameter

Use the STMT_CACHE_HITS configuration parameter to specify the number of hits (references) to a statement before it is fully inserted in the SQL statement cache.

onconfig.std value
STMT_CACHE_HITS 0
values
`0` = Fully insert all qualified statements in the SQL statement cache.
`>0` = The first time a user issues a unique statement, the database server inserts a *key-only* entry in the cache that identifies the statement. Subsequent identical statements increment the hit count of the *key-only* cache entry. When the hit count of the *key-only* cache entry reaches the specified number of hits, the database server fully inserts the statement in the cache. Set *hits* to `1` or more to exclude ad hoc queries from entering the cache.

units
Integer
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -W: Change settings for the SQL statement cache
onstat -g ssc command: Print SQL statement occurrences
Related information:
Number of SQL statement executions

STMT_CACHE_NOLIMIT configuration parameter

Use the STMT_CACHE_NOLIMIT configuration parameter to control whether to insert qualified statements into the SQL statement cache.

onconfig.std value
STMT_CACHE_NOLIMIT 0
if not present
1
values
`0` = Prevents statements from being inserted in the cache. The cache can grow beyond the size limit if most of the statements in the cache are currently in use, because the cache cleaning cannot catch up with the insert rate. If you are concerned about memory usage, turn off STMT_CACHE_NOLIMIT to prevent the database server from allocating a large amount of memory for the cache.
`1` = Always insert statements in the SQL statement cache regardless of the cache size.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -W: Change settings for the SQL statement cache
onstat -g ssc command: Print SQL statement occurrences
Related information:
Number of SQL statement executions

STMT_CACHE_NUMPOOL configuration parameter

Use the STMT_CACHE_NUMPOOL configuration parameter to specify the number of memory pools for the SQL statement cache. To obtain information about these memory pools, use **onstat -g ssc pool**.

Because the database server does not insert all statements that allocate memory from the memory pools in the cache, the cache size might be smaller than the total size of the memory pools.

onconfig.std value
: STMT_CACHE_NUMPOOL 1

values
: 1 - 256

units
: Positive integer

takes effect
: After you edit your onconfig file and restart the database server.

Related reference:
onstat -g ssc command: Print SQL statement occurrences
Related information:
Number of SQL statement executions

STMT_CACHE_QUERY_PLAN configuration parameter

Use the STMT_CACHE_QUERY_PLAN configuration parameter to produce a query plan from any query that exists in the Statement Cache.

onconfig.std value
: `STMT_CACHE_QUERY_PLAN`

values
: `0` = Disabled.
`1` = Enabled

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

You can use STMT_CACHE_QUERY_PLAN to enable SQL statement cache to hold query plan for each cached statement.

STMT_CACHE_SIZE configuration parameter

Use the STMT_CACHE_SIZE configuration parameter to specify the size of the SQL statement caches in kilobytes. The new cache size takes effect the next time a statement is added to a cache.

onconfig.std value
: STMT_CACHE_SIZE 512

values
: Positive integer

units
: Kilobytes

takes effect
: After you edit your onconfig file and restart the database server.

Related information:
Monitoring and tuning the size of the SQL statement cache

STOP_APPLY configuration parameter

Use the STOP_APPLY configuration parameter to stop an RS secondary server from applying log files received from the primary server.

onconfig.std value
: STOP_APPLY 0

default value

0

values

0 = Apply logs
1 = Stop applying logs immediately

YYYY:*MM*:*DD*-*hh*:*mm*:*ss* = Stop the log apply at a specified time, where:

- *YYYY* = Year
- *MM* = Month
- *DD* = Day
- *hh* = Hour (24-hour notation)
- *mm* = Minute
- *ss* = Second

takes effect

After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Stopping the application of log files allows you to recover quickly from erroneous database modifications by restoring the data from the RS secondary server. You can configure the server to either stop the application of logs immediately, or at a specified point in time. When setting the value of STOP_APPLY you must also set LOG_STAGING_DIR. If STOP_APPLY is configured and LOG_STAGING_DIR is not set to a valid and secure directory, the server cannot be initialized.

Log files are stored in binary format in a directory specified by the LOG_STAGING_DIR configuration parameter. You must specify a valid and secure location for the log files.

To see information about the data being sent to the log-staging directory set for a RS secondary server, run the **onstat -g rss verbose** command on the RS secondary server.

If the write to the staging file fails, the RS secondary server raises event alarm 40007.

The time value specified for the STOP_APPLY configuration parameter is assumed to be in the same timezone as the RS secondary server.

The **dbexport** utility cannot support write operations on an updatable secondary server unless the STOP_APPLY parameter is set. (Besides STOP_APPLY, the UPDATABLE_SECONDARY and USELASTCOMMITTED configuration parameters must also be set to enable write operations by **dbexport** on a secondary data replication server.)

If a remote stand-alone secondary (RSS) server has its STOP_APPLY configuration parameter set to a value other than 0, that server cannot use cluster transaction coordination.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DELAY_APPLY Configuration Parameter
UPDATABLE_SECONDARY configuration parameter
CLUSTER_TXN_SCOPE configuration parameter
LOG_STAGING_DIR configuration parameter
onstat -g cluster command: Print high-availability cluster information
Related information:
CLUSTER_TXN_SCOPE session environment option

STORAGE_FULL_ALARM configuration parameter

Use the STORAGE_FULL_ALARM configuration parameter to configure the frequency and severity of messages and alarms when storage spaces become full.

onconfig.std value

STORAGE_FULL_ALARM 600,3

values

seconds = 0 (off) or a positive integer indicating the number of seconds between notifications.
severity_level = 0 (no alarms) or 1 - 5

units

seconds,*severity_level*

takes effect

After you edit your onconfig file and restart the database server.

Usage

When a storage space, such as a dbspace, sbspace, blobspace, or tblspace, or a partition becomes full, an alarm is raised and a message is sent to the online message log. You can specify the number of seconds between notifications with the first value of this parameter. You can specify the lowest severity for event alarms to be returned. Setting a specific severity prevents events that have a lower severity from being raised. But events that have the same or greater severity as the severity specified are raised. You can prevent alarms when storage spaces become full by setting this parameter to 0.

Regardless of the value of STORAGE_FULL_ALARM, messages are sent to the online message log when storage spaces or partitions become full.

Related reference:

Event Alarm Parameters
Related information:
Monitor storage spaces

SYSALARMPROGRAM configuration parameter

Use the SYSALARMPROGRAM configuration parameter to specify the full path name of the evidence.sh script. The database server executes evidence.sh when a database server failure occurs. You can use the output from the evidence.sh script to diagnose the cause of a database server failure.

onconfig.std value

- On UNIX: $INFORMIXDIR/etc/evidence.sh
- On Windows: Not set. (Commented out.) Listed as $INFORMIXDIR\etc\evidence.bat

values
pathname = Full path name of the evidence.sh script.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

On Windows, you must enable command extensions for evidence.bat to successfully complete. You can enable and disable the extensions for the Command Prompt you are working in by issuing the following commands:

- Enable: **cmd /x**
- Disable: **cmd /y**

You can also enable and disable command extensions from the Windows XP registry:

Table 1. Enabling command extensions from the Windows registry

Attribute	Value
Hive	HKEY_CURRENT_USER
Key	Software\Microsoft\Command Processor
Name	EnableExtensions
Type	REG_DWORD
Values	0 (disable), 1 (enable)

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

SYSSBSPACENAME configuration parameter

Use the SYSSBSPACENAME configuration parameter to specify the name of the sbspace in which the database server stores fragment-level data-distribution statistics, which the **syfragsdist** system catalog table stores as BLOB objects in its **encsdist** column. Also use SYSSBSPACENAME to specify the name of the sbspace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types.

onconfig.std value
Not set.
if not present
0
values
Up to 128 bytes. SYSSBSPACENAME must be unique, begin with a letter or underscore, and contain only digits, letters, underscores, or **$** characters.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.
refer to

- Updating statistics, in the chapter on individual query performance in your *IBM® Informix Performance Guide*
- Sbspace characteristics, in the chapter on configuration effects on I/O in your *IBM Informix Performance Guide*
- Writing user-defined statistics, in the performance chapter in *IBM Informix User-Defined Routines and Data Types Developer's Guide*
- Providing statistics data for a column, in the *IBM Informix DataBlade API Programmer's Guide*

Usage

To support fragment level statistics, you must specify the name of an sbspace as the SYSSBSPACENAME setting, and you must allocate that sbspace (by using the **onspaces** utility, as described below. For any table whose STATLEVEL attribute is set to FRAGMENT, the database server returns an error if SYSSBSPACENAME is not set,

or if the sbspace to which SYSSBSPACENAME is set was not properly allocated).

For the distribution statistics of a column in a fragmented table, you can estimate how many bytes of storage capacity the sbspace requires by this formula:

```
nfrags * 1.25 * ((10000 / resolution) * ((2 * column_width) + 6))
```

Here 1.25 approximates the number of overflow bins. The formula also includes these variables:

- *column_width* is the width in bytes of the column that the UPDATE STATISTICS statement specifies.
- *nfrags* is the number of fragments of the table.
- *resolution* is the *percent* value in the resolution clause of the UPDATE STATISTICS statement that calculates the distribution.

The *resolution* is also what the **dbschema -hd** *table* command displays for the column distribution statistics.
SYSSBSPACENAME also specifies the name of the sbspace in which the database server stores statistics that the UPDATE STATISTICS statement collects for certain user-defined data types. Normally, the database server stores statistics in the **sysdistrib** system catalog table.

Do not confuse the SYSSBSPACENAME configuration parameter with the SBSPACENAME configuration parameter .

Because the data distributions for user-defined data types can be large, you have the option to store them in an sbspace instead of in the **sysdistrib** system catalog table. If you store the data distributions in an sbspace, use DataBlade API or Informix® ESQL/C functions to examine the statistics.

Even though you specify an sbspace with the SYSSBSPACENAME parameter, you must create the sbspace with the -c -S option of the **onspaces** utility before you can use it. The database server validates the name of this sbspace when one of the following occurs:

- The database server attempts to write data distributions of the multirepresentational type to SYSSBSPACENAME when it executes the UPDATE STATISTICS statement with the MEDIUM or HIGH keywords.
- The database server attempts to delete data distributions of the multirepresentational type to SYSSBSPACENAME when it executes the UPDATE STATISTICS statement with the DROP DISTRIBUTIONS keywords.

If SBSSPACENAME is not set, or if storage is not allocated to that sbspace, the database server might not be able to store the distribution statistics, so that the UPDATE STATISTICS operation fails with error -9814.
Although you can store smart large objects in the sbspace specified in SYSSBSPACENAME, keeping the distribution statistics and smart large objects in separate sbspaces is recommended, because:

- You avoid disk contention when queries are accessing smart large objects, and the query optimizer is using the distributions to determine a query plan.
- Disk space takes longer to fill up when each sbspace is used for a different purpose.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
SBSPACENAME configuration parameter
Sbspace Structure
onspaces -c -S: Create an sbspace

TBLSPACE_STATS configuration parameter

Use the TBLSPACE_STATS configuration parameter to turn on and off the collection of tblspace statistics. Use the **onstat -g ppf** command to list tblspace statistics.

onconfig.std value
TBLSPACE_STATS 1
values
0 = Turn off the collection of tblspace statistics. The **onstat -g ppf** command displays `partition profiles disabled`.
1 = Turn on the collection of tblspace statistics.

units
Integer
takes effect
After you edit your onconfig file and restart the database server.

Related reference:
onstat -g ppf command: Print partition profiles

TBLTBLFIRST configuration parameter

Use the TBLTBLFIRST configuration parameter if you want to specify the first extent size of tblspace **tblspace** in the root dbspace. Set this parameter if you do not want the database server to automatically manage the extent size.

onconfig.std value
TBLTBLFIRST 0
values
From the equivalent of 250 pages specified in kilobytes to the size of the first chunk minus the space needed for any system objects.
units
Kilobytes in multiples of page size
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

When you reset the value in memory by running the **onmode -wm** command.

Usage

You might want to specify first and next extent sizes to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you need to place the tblspace **tblspace** extents in non-primary chunks. (A primary chunk is the initial chunk in a dbspace.)

You can use **oncheck -pt** and **oncheck -pT** to show the first and next extent sizes of a tblspace **tblspace**.

If you want to configure the first extent for a non-root dbspace, use the **onspaces** utility.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
TBLTBLNEXT configuration parameter
oncheck -pt and -pT: Display tblspaces for a Table or Fragment
The onspaces utility
onspaces -c -d: Create a dbspace
Related information:
Specifying the first and next extent sizes for the tblspace tblspace

TBLTBLNEXT configuration parameter

The TBLTBLNEXT configuration parameter specifies the next extent size of tblspace **tblspace** in the root dbspace. Set this parameter if you do not want the database server to automatically manage the extent size.

onconfig.std value
TBLTBLNEXT 0
values
From equivalent of 4 pages specified in kilobytes to the maximum chunk size minus three pages
units
Kilobytes
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If there is not enough space for a next extent in the primary chunk, the extent is allocated from another chunk. If the specified space is not available, the closest available space is allocated.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
TBLTBLFIRST configuration parameter
onspaces -c -d: Create a dbspace
Related information:
Specifying the first and next extent sizes for the tblspace tblspace

TEMPTAB_NOLOG configuration parameter

Use the TEMPTAB_NOLOG configuration parameter to disable logging on temporary tables.

onconfig.std value
TEMPTAB_NOLOG 0
values
0 = Enable logical logging on temporary table operations
1 = Disable logical logging on temporary table operations

2 = Enable logical logging on temporary table operations for primary server and disable logical logging on temporary table operations for secondary servers(HDR, RSS and SDS).

On primary/standard server: same behavior as 0

On secondary servers: same behavior as 1

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

This parameter can improve performance in application programs because it prevents IBM® Informix® from transferring temporary tables over the network. The setting can be updated dynamically with the **onmode -wf** utility.

If you enable this setting, be aware that because no data is logged when using temporary tables, rolling back a transaction on a temporary table will no longer undo the work in the temporary table.

For HDR, RSS and SDS secondary servers in a high-availability cluster, logical logging on temporary tables should always be disabled by setting the TEMPTAB_NOLOG configuration parameter to 1 or 2.

When server type changes, logging for temporary tables will be enabled/disabled depending on the current server role. It will be effective for temporary tables created afterwards.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

TENANT_LIMIT_CONNECTIONS configuration parameter

The TENANT_LIMIT_CONNECTIONS configuration parameter specifies the maximum number of connections to a tenant database.

onconfig.std value
: 0 (off)

if not present
: 0 (off)

values
: 1 - 65536

takes effect
: After you edit your onconfig file and restart the database server.

Usage

When the limit is reached, subsequent connection requests to the tenant database are rejected.

The `tenant_limit_connections` tenant database property set through the **tenant create** or **tenant update** SQL API command takes precedent over the TENANT_LIMIT_CONNECTIONS configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)

TENANT_LIMIT_MEMORY configuration parameter

The TENANT_LIMIT_MEMORY configuration parameter specifies the maximum amount of shared memory for all sessions that are connected to the tenant database.

onconfig.std value
: 0 (off)

if not present
: 0 (off)

values
: 102400 - 2147483648

units
: KB

takes effect
: After you edit your onconfig file and restart the database server.

Usage

When the limit is exceeded, the session that is using the most shared memory is terminated. The value ranges from 100 MB to 2 TB, but must be specified as an integer that represents the number of KB.

The `tenant_limit_memory` tenant database property set through the **tenant create** or **tenant update** SQL administration API command takes precedent over the TENANT_LIMIT_MEMORY configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)

TENANT_LIMIT_SPACE configuration parameter

The TENANT_LIMIT_SPACE configuration parameter specifies the maximum amount of storage space available to a tenant database. Storage space includes all permanent dbspaces, BLOB spaces, and sbspaces.

onconfig.std value
: 0 (off)

if not present
: 0 (off)

values
: 1048576 - 1717986918400

units
: KB

takes effect
: After you edit your onconfig file and restart the database server.

Usage

The TENANT_LIMIT_SPACE configuration parameter limits the amount of permanent storage space available to a tenant database, and can conserve system resources within a tenant-database environment. When the limit is reached, subsequent operations that require additional disk space are rejected. The value ranges from 1 GB to 200 TB, but must be specified as an integer that represents the number of KB.

The `tenant_limit_space` tenant database property set through the **tenant create** or **tenant update** SQL administration API command takes precedent over the TENANT_LIMIT_SPACE configuration parameter setting.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
Multitenancy
Managing tenant databases
Limit session resources

TLS_VERSION configuration parameter

Use the TLS_VERSION configuration parameter to specify the Transport Layer Security (TLS) version that the database server uses for network connections. TLS version 1.2 is enabled by default if no specification is done in onconfig file.

onconfig.std value
: Not set. – which defaults to 1.2.

default value
: 1.2

values
: One or more TLS versions. Multiple versions are separated by commas.

- `1.0` = TLS version 1.0
- `1.1` = TLS version 1.1
- `1.2` = TLS version 1.2
- `1.3` = TLS version 1.3

takes effect
: After you edit the onconfig file and restart the database server.

Usage

TLS is the successor to Secure Sockets Layer (SSL) and provides cryptographic protocols for client/server connections.

GSKit TLS 1.3 does not support FIPS mode yet. If the server offers both TLS 1.2 and TLS 1.3, then the preference for FIPS will cause connection via TLS 1.2 .

Note:
Limiting server to TLS 1.3 will require disabling FIPS mode by the client. JDBC support for TLS 1.3 requires Java 11.

Related information:
Secure sockets layer protocol
Environment variables for client

TXTIMEOUT configuration parameter

Use the TXTIMEOUT configuration parameter to specify the amount of time that a participant in a two-phase commit waits before it initiates participant recovery. This parameter is used only for distributed queries that involve a remote database server. Nondistributed queries do not use this parameter.

onconfig.std value
: TXTIMEOUT 300

values

Positive integers

units
Seconds

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
How the two-phase commit protocol handles failures

UNSECURE_ONSTAT configuration parameter

Use the UNSECURE_ONSTAT configuration parameter to remove the database system administrator (DBSA) user access restriction for onstat commands.

onconfig.std value
Not set.

values
`1` = All users can run **onstat** commands to view running SQL statements

takes effect
After you edit your onconfig file and restart the database server.

Usage

By default, the onstat commands that show the SQL statement text from an active session are restricted to DBSA users. To remove this restriction, set the UNSECURE_ONSTAT configuration parameter to `1`. The **onstat** commands that show SQL statements include **onstat -g his**, **onstat -g ses**, **onstat -g stm**, **onstat -g ssc**, and **onstat -g sql**.

UPDATABLE_SECONDARY configuration parameter

Use the UPDATABLE_SECONDARY configuration parameter to set the number of connections to establish between the primary and secondary servers. Setting this configuration parameter enables client applications to perform update, insert, and delete operations on a high-availability secondary server.

onconfig.std value
UPDATABLE_SECONDARY 0

values
Any number from zero (the default value) up to twice the number of CPU VPs. Setting the value to `0` configures the secondary server as read-only. Setting the value from `1` through twice the number of CPU VPs makes the secondary server updatable and configures connection threads.

units
Number of network connections between a given secondary server and its primary server

takes effect
After you edit your onconfig file and restart the database server.

Isolation Levels for Secondary Data Replication Servers

If the UPDATABLE_SECONDARY configuration parameter is not set or is set to zero, a secondary data replication server is read-only. In this case, only the DIRTY READ or READ UNCOMMITTED transaction isolation levels are available on secondary servers.

If the UPDATABLE_SECONDARY parameter is set to a valid number of connections greater than zero, a secondary data replication server can support the COMMITTED READ , COMMITTED READ LAST COMMITTED, or COMMITTED READ transaction isolation level, or the USELASTCOMMITTED session environment variable. Only SQL DML statements, such as INSERT, UPDATE, MERGE, and DELETE, and the **dbexport** utility, can support write operations on an updatable secondary server. (Besides UPDATABLE_SECONDARY, the STOP_APPLY and USELASTCOMMITTED configuration parameters must also be set to enable write operations by **dbexport** on a secondary data replication server.)

Related reference:
STOP_APPLY configuration parameter
onstat -g cluster command: Print high-availability cluster information
Related information:
Database updates on secondary servers
Set the wait time for SMX activity between servers

USELASTCOMMITTED configuration parameter

Use the USELASTCOMMITTED configuration parameter to specify the isolation level for which the LAST COMMITTED feature of the COMMITTED READ isolation level is implicitly in effect.

onconfig.std value
    USELASTCOMMITTED "NONE"
default value
    "NONE"
values
    "NONE" = No isolation level identified. If your session encounters an exclusive lock when attempting to read a row in the Committed Read, Dirty Read, Read Committed, or Read Uncommitted isolation level, your transaction cannot read that row until the concurrent transaction that holds the exclusive lock is committed or rolled back.
    "COMMITTED READ" = All transactions from a Committed Read isolation level are treated as last committed transactions. The database server reads the most recently committed version of the data when it encounters an exclusive lock while attempting to read a row in the Committed Read or Read Committed isolation level.
    "DIRTY READ" = All transactions from a Dirty Read isolation level are treated as last committed transactions. The database server reads the most recently committed version of the data if it encounters an exclusive lock while attempting to read a row in the Dirty Read or Read Uncommitted isolation level.
    "ALL" = All transactions from both Committed Read and Dirty Read isolation levels are treated as last committed transactions. database server reads the most recently committed version of the data if it encounters an exclusive lock while attempting to read a row in the Committed Read, Dirty Read, Read Committed, or Read Uncommitted isolation level.
takes effect
    After you edit your onconfig file and restart the database server.
    When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
    When you reset the value in memory by running the **onmode -wm** command.

Usage

The LAST COMMITTED feature can reduce the risk of locking conflicts between concurrent transactions on tables that have exclusive row locks. The USELASTCOMMITTED configuration parameter can also enable LAST COMMITTED semantics for READ COMMITTED and READ UNCOMMITTED isolation levels of the SET TRANSACTION statement.

The USELASTCOMMITTED configuration parameter only works with tables that have been created or altered to have ROW as their locking granularity. Tables created without any explicit lock mode setting will use the default setting in DEF_TABLE_LOCKMODE. If DEF_TABLE_LOCKMODE is set to PAGE, the USELASTCOMMITTED configuration parameter cannot enable access to the most recently committed data in tables on which uncommitted transactions hold exclusive locks, unless the tables were explicitly altered to have ROW level of locking granularity.

Use with Shared Disk secondary database servers

The USELASTCOMMITTED configuration parameter is also valid on Shared Disk (SD) secondary database servers. The following table shows valid values for the USELASTCOMMITTED configuration parameter on SD secondary servers and their descriptions.

Table 1. Valid secondary server USELASTCOMMITTED values

USELASTCOMMITTED value	Description
NONE	COMMITTED READ LAST COMMITTED is not the default isolation level for sessions
COMMITTED READ	COMMITTED READ LAST COMMITTED is the default isolation level for all sessions with Committed Read isolation
DIRTY READ	COMMITTED READ LAST COMMITTED is the default isolation level for all sessions with Dirty Read isolation
ALL	COMMITTED READ LAST COMMITTED is the default isolation level for all sessions with Committed Read or Dirty Read isolation

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
DEF_TABLE_LOCKMODE configuration parameter
Related information:
USELASTCOMMITTED session environment option
SET ISOLATION statement

USEOSTIME configuration parameter

Use the USEOTIME configuration parameter to control whether the database server uses subsecond precision when obtaining the current time from the operating system.

onconfig.std value
    USEOSTIME 0
values
    0 = Off
    1 = On

takes effect
    During initialization
refer to

- Your *IBM® Informix Performance Guide*
- Using the CURRENT function to return a datetime value, in the *IBM Informix Guide to SQL: Syntax*

Usage

Setting USEOSTIME to `1` specifies that the database server is to use subsecond precision when it obtains the current time from the operating system for SQL statements. The following example shows subseconds in a datetime value:

```
2001-09-29 12:50:04.612
```

If subsecond precision is not needed, the database server retrieves the current time from the operating system once per second, making the precision of time for client applications one second. If you set USEOSTIME to `0`, the current function returns a zero (`.000`) for the year to fraction field.

When the host computer for the database server has a clock with subsecond precision, applications that depend on subsecond accuracy for their SQL statements should set USEOSTIME to `1`.

Systems that run with USEOSTIME set to nonzero notice a performance degradation of up to 4 to 5 percent compared to running with USEOSTIME turned off.

This setting does not affect any calls regarding the time from application programs to Informix® embedded-language library functions.

USERMAPPING configuration parameter (UNIX, Linux)

Use the USERMAPPING configuration parameter to set whether or not the database server accepts connections from mapped users.

default value
: `OFF`

values
: `OFF` = Only users that are registered in the IBM® Informix® host computer OS with a login service can connect to the database server. Externally authenticated users without OS accounts on the host computer cannot connect to database server resources.
`BASIC` = Users can connect to without an OS account. A user without an OS account cannot perform privileged user operations on the database server, even if the user maps to a server administrator user or group ID.

`ADMIN` = Users can connect to without an OS account. If a user has authenticated with the identity of a privileged user and is mapped to the proper server administrator group ID, the user can perform DBSA, DBSSO, or AAO work on the database server.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

Externally authenticated users without operating system (OS) accounts on the host computer can access database server resources when USERMAPPING is turned on by setting the parameter with the BASIC or ADMIN value. The setting of BASIC or ADMIN also determines whether or not mapped users can be granted administrative privileges.

Important: Changing the USERMAPPING configuration parameter from OFF to ADMIN or BASIC is not the only step in setting up for mapped users. To map users with the appropriate user properties, you must also use DDL statements such as CREATE USER and ALTER USER to register values in appropriate system tables of the SYSUSER database. Depending on the DDL statement used and the defined table mapping, the following tables will be updated or populated:

- SYSINTAUTHUSERS
- SYSUSERMAP
- SYSSURORGATES
- SYSSURROGATEGROUPS

USRC_HASHSIZE configuration parameter

The USRC_HASHSIZE configuration parameter specifies the number of hash buckets in the LBAC credential memory cache. This memory cache holds information about the LBAC credentials of users.

onconfig.std value
: USRC_HASHSIZE 31

values
: Any positive integer

units
: KB

takes effect
: After you edit your onconfig file and restart the database server.

Related information:
Maintaining a label-based access-control implementation

USRC_POOLSIZE configuration parameter

The USRC_POOLSIZE configuration parameter specifies the maximum number of entries in each hash bucket of the LBAC credential memory cache. This memory cache holds information about the LBAC credentials of users.

onconfig.std value
: USRC_POOLSIZE 127

values
: A positive value 127 or greater that represents half of the initial maximum number of entries in the cache. The maximum value is dependent upon the shared memory configuration and available shared memory for the server instance.

takes effect
: After you edit your onconfig file and restart the database server.
: When you increase the value in memory by running the **onmode -wm** command.
: When you reset the value in memory by running the **onmode -wm** command.

The initial number of entries in the cache is twice the value of the USRC_POOLSIZE configuration parameter. For example, if the USRC_POOLSIZE configuration parameter is set to 127, 254 entries are allowed in the cache. If all entries in a cache are full, the cache size automatically grows by 10%. To reduce the size of the cache, decrease the value of the USRC_POOLSIZE configuration parameter in the onconfig file and restart the server.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
Maintaining a label-based access-control implementation

USTLOW_SAMPLE configuration parameter

Use the USTLOW_SAMPLE configuration parameter to enable the generation of index statistics based on sampling when you run UPDATE STATISTICS statements in LOW mode.

For an index with more than 100 K leaf pages, the gathering of statistics using sampling can increase the speed of the UPDATE STATISTICS operation.

onconfig.std value
: USTLOW_SAMPLE 1

values
: 0 = disable sampling
: 1 = enable sampling

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
Related information:
USTLOW_SAMPLE environment option
Data sampling during update statistics operations

VP_MEMORY_CACHE_KB configuration parameter

Use the VP_MEMORY_CACHE_KB parameter to create a private memory cache for each CPU virtual processor and tenant virtual processor.

onconfig.std value
: VP_MEMORY_CACHE_KB 0

values
: 0 = Off
: The total size of all private memory caches, optionally followed by a comma and the mode of the caches.
: Size, in KB:

  - 800 to 40% of the SHMTOTAL configuration parameter value.

: Mode:

  - STATIC = Default. The specified size is the maximum combined size of all private memory caches.
  - DYNAMIC = The specified size is the initial size of all private memory caches. The cache size changes dynamically but cannot exceed the value of the SHMTOTAL configuration parameter.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

Private memory caches can improve performance for memory that is allocated by threads in the server. Private memory caches have no impact on the memory that is allocated to and used by buffer pools or shared memory communication.

When you set the value of the VP_MEMORY_CACHE_KB configuration parameter to a number other than 0, a private memory cache is created for each CPU virtual processor and tenant virtual processor. By default, size of all private memory caches combined is limited to the specified number of KB.

If you want the size of each private memory cache to increase and decrease automatically, as needed, include a comma and the word DYNAMIC after the size, for example, VP_MEMORY_CACHE_KB 1000,DYNAMIC. Although the maximum initial size of all private memory caches combined cannot exceed 40 percent of the value of the SHMTOTAL configuration parameter, with DYNAMIC mode set, the size of the caches can expand beyond the initial limit. The total size of the caches cannot exceed the value of the SHMTOTAL configuration parameter. DYNAMIC mode can improve performance when many threads are disconnecting at the same time or there is contention for the shared memory lock. You can use the **onstat -g wmx** command to display information about mutexes, such as the shared memory lock mutex `shmcb sh_lock`, and any threads waiting on mutexes.

Attention: Dynamic memory caches on busy systems can grow quickly and use a large amount of available memory. Therefore, if you set the mode to DYNAMIC, set the SHMTOTAL configuration parameter to a specific limit instead of the default value of 0, which does not limit the amount of memory.
If you reset the VP_MEMORY_CACHE_KB configuration parameter to 0, the memory caches are emptied and disabled.

The **onstat -g vpcache** command returns statistics about private memory caches.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g vpcache command: Print CPU virtual processor and tenant virtual processor private memory cache statistics
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
Related information:
Private memory caches

VPCLASS configuration parameter

Use the VPCLASS configuration parameter to create and configure virtual processors.

onconfig.std values
UNIX: `VPCLASS cpu,num=1,noage`

Windows:

- `VPCLASS cpu,num=1,noage`
- `#VPCLASS aio,num=1`
- `#VPCLASS jvp,num=1`

values
Up to 128 bytes of characters. Each VPCLASS configuration parameter value must be unique, begin with a letter or underscore, and contain only digits, letters, underscores, or $ characters. Do not include blank spaces. See the Usage section.

separators
Separate each field with a comma.

takes effect
After you edit your onconfig file and restart the database server.

Usage

You can add multiple VPCLASS configuration parameter entries in your onconfig file. Each VPCLASS configuration parameter must describe a different class of virtual processors. Put each definition on a separate line.

Syntax for the VPCLASS configuration parameter

```
>>-VPCLASS--+-class-----------------------+-------------------->
            +-| cpu class |----------------+
            +-aio--+----------+--=--+-0-+--+
            |      '-autotune-'     '-1-'  |
            '-user_defined--+-------------+-'
                            '-,--noyield-'

>--+-----------------------+--+--------------------+-----------><
   '-,--num--=--number_vps-'  '-,--max--=--maximum-'
```

cpu class

```
|--cpu-------------------------------------------------------->

>--+-------------------------------------------------------+---->
   |              .-,--------------------------------.    |
   |              V                                  |    |
   '-,--aff--=--(----+-processor--------------------+-+--)-'
                     '-start -end--+--------------+-'
                                   '-/--increment-'

>--+----------+--+-------------------------+-----------------|
   '-,--noage-'  '-,--+----------+--=--+-0-+-'
                      '-autotune-'     '-1-'
```

Table 1. Options for the VPCLASS configuration parameter value

Field	Values
class	The *class* value is the name of the virtual processor class. The database server starts most virtual processors as needed. Typically, you might set the VPCLASS configuration parameter for the CPU, AIO, JVP, and user-defined virtual processor classes. The virtual processor class name is not case-sensitive. For a list of class names, see Virtual processor classes.
user_defined	The *user_defined* value is the name of a virtual processor class that you create for user-defined routines. Make sure the SINGLE_CPU_VP configuration parameter is set to 0.
autotune	Specifies whether the database server adds virtual processors for the specified class as needed to improve performance, up to the value of the **max** option, if it is included. • **autotune=0** prevents the automatic addition of virtual processors • **autotune=1** enables the automatic addition of virtual processors If the class is **cpu**, any CPU virtual processors that are automatically added do not have affinity. The **aff** option is ignored.
cpu	Specifies the CPU virtual processor class.
num	The *number_vps* value sets the number of virtual processors of the specified class that the database server starts when the database server starts. The default value is 1. The range of values for the **cpu** and **aio** virtual processor classes is 1 - 10000. The range of values for all other virtual processor classes is 0 - 10000. You can use the **onmode -p** command to add virtual processors for the class for the current session.
max	The *maximum* value specifies the maximum number of virtual processors that the database server can start for the class. The value can be any integer greater than 0. By default, the number is unlimited.
aff	On multiprocessor computers that support processor affinity, the **aff** option specifies the CPUs to which the database server binds CPU virtual processors. The operating system numbers the CPUs from 0 to one less than the number of CPUs. By default, CPU virtual processors are assigned to available processors in round-robin fashion. The **aff** option takes one or more integers: • *processor* = The CPU number to which to bind the CPU virtual processors. The CPU numbers can be listed in any order. • *start* = The beginning of a range of CPU numbers. • *end* = The end of a range of CPU numbers. • *increment* = A factor that specifies which of the CPU numbers in a range are used. For example, `aff=(1-5/2)` specifies to use CPU numbers 1, 3, and 5.
noage	Disables priority aging for CPU virtual processors, if the operating system implements priority aging. By default, priority aging is in effect.
noyield	Specifies that a user-defined virtual processor class does not yield, which allows the C UDR to yield to other threads that need access to the user-defined virtual processor class. By default, threads for user-defined virtual processors yield. A nonyielding user-defined virtual processors class runs a user-defined routine in a way that gives the routine exclusive use of the virtual processor class. User-defined routines that use a noyield virtual-processor class run serially and never yield the virtual processors to another thread. Specify only one virtual processor in a nonyielding user-defined virtual processor class, because the UDR runs on a single virtual processor until it completes and any additional virtual processors would be idle.

The options can appear in any order, separated by commas.

Use the **onmode -p** command to dynamically add or remove virtual processors for the current database session. The **onmode -p** command does not update the onconfig file.

CPU virtual processors

On a single-processor computer, allocate only one CPU virtual processor. On a multiprocessor computer, allocate a total number of CPU virtual processes plus user-defined virtual processors up to the number of CPUs on the computer.

When the database server starts, the number of CPU virtual processors is automatically increased to half the number of CPU processors on the database server computer, unless the SINGLE_CPU_VP configuration parameter is enabled.

If you include the **autotune** option, the database server adds CPU virtual processors as needed to improve performance, up to the number of CPUs on the computer.

The value of the **num** option of the VPCLASS configuration parameter for the CPU class is not updated when the database server automatically adds CPU virtual processors.

You can configure processor affinity and whether to allow aging. For example, the following entry creates four CPU virtual processors that are bound to CPU numbers 7, 8, 9, and 10, and are not affected by priority aging:

```
VPCLASS CPU,num=4,aff=(7-10),noage
```

AIO virtual processors

Use a VPCLASS configuration parameter entry for the AIO virtual processor class to specify an exact number of AIO virtual processors or to enable the database server to add AIO virtual processors as needed.

When no VPCLASS configuration parameter entry for the AIO virtual processor class is set, the number of AIO virtual processors is determined by the setting of the AUTO_AIOVPS configuration parameter and is limited to 128:

- If AUTO_AIOVPS is set to 1 (on), the number of AIO virtual processors that are initially started is equal to the number of AIO chunks.
- If AUTO_AIOVPS is set to 0 (off), the number of AIO virtual processors that are started is equal to the greater of 6 or twice the number of AIO chunks.

Java virtual processors

If you use Java™ user-defined routines or Java applications, create at least one Java virtual processor by adding a VPCLASS configuration parameter entry for the JVP virtual processor class. If you set the number of JVPs to zero, or if there is no VPCLASS parameter for the JVP class, you cannot run Java UDRs.

Related reference:
AUTO_AIOVPS configuration parameter
DS_MAX_QUERIES configuration parameter
DS_TOTAL_MEMORY configuration parameter
NETTYPE configuration parameter
The number of configured inline poll threads exceeds the number of CPU virtual processors.
Virtual processor limit exceeded.
onmode -p: Add or drop virtual processors
Related information:
Tenant virtual processor class
Virtual processor classes
CPU virtual processors
User-defined classes of virtual processors
Java virtual processors
AIO virtual processors

VP_KAIO_PERCENT configuration parameter

VP_KAIO_PERCENT is the percentage of total KAIO event resources on the system that each CPU VP will allocate.

onconfig.std value
    VP_KAIO_PERCENT 0
if not present
    0 (off)
values
    1-100
units
    Percent
takes effect
    After you edit your onconfig file and restart the database server.
    When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
    When you reset the value in memory by running the **onmode -wm** command.

Usage

VP_KAIO_PERCENT is the percentage of total KAIO event resources on the system that each CPU VP will allocate.

If you set VP_KAIO_PERCENT to 10 and the total number of KAIO events configured in the OS kernel is 50000 then each CPU VP that starts will attempt to allocate 5000 events.

A value of 0 follows legacy algorithm for allocating KAIO resources to VPs.

WSTATS configuration parameter

Use the WSTATS configuration parameter to specify whether the **onstat -g wst** command displays wait statistics for threads within the system.

Attention: You should expect a small performance impact due to the cost of gathering statistical information. Enabling the WSTATS configuration parameter for production systems is not recommended.

onconfig.std value
    WSTATS 0
range of values
    0 = Disable wait statistics
    1 = Enable wait statistics

takes effect
    After you edit your onconfig file and restart the database server.
    When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
    When you reset the value in memory by running the **onmode -wm** command.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -g wst command: Print wait statistics for threads

The sysmaster database

These topics describe the **sysmaster** database and provide reference information for the *system-monitoring interface* (SMI).

These topics include:

- A description of the **sysmaster** database
- Information about how to use SMI tables
- Descriptions of the SMI tables
- A map of the documented SMI tables

For information about the ON-Bar tables, see the *IBM® Informix® Backup and Restore Guide*.

- The sysmaster Database
- The System-Monitoring Interface
- The System-Monitoring Interface Tables
  The **sysmaster** database contains many tables that you can use to monitor your system.

The sysmaster Database

The database server creates and maintains the **sysmaster** database. It is analogous to the system catalog for databases, which is described in the *IBM® Informix® Guide to SQL: Reference*. Just as a system catalog for every database managed by the database server keeps track of objects and privileges in the database, a **sysmaster** database for every database server keeps track of information about the database server.

The **sysmaster** database contains the *system-monitoring interface* (SMI) tables. The SMI tables provide information about the state of the database server. You can query these tables to identify processing bottlenecks, determine resource usage, track session or database server activity, and so on. This chapter describes these tables, which are slightly different from ordinary tables.
Warning: The database server relies on information in the **sysmaster** database. Do not change any of the tables in **sysmaster** or any of the data within the tables. Such changes could cause unpredictable and debilitating results.
The database server creates the **sysmaster** database when it initializes disk space. The database server creates the database with unbuffered logging. You cannot drop the database or any of the tables in it, and you cannot turn logging off.

As user **informix** on UNIX or a member of the **Informix-Admin** group on Windows, you can create SPL routines in the **sysmaster** database. (You can also create triggers on tables within **sysmaster**, but the database server never executes those triggers.)

Joins of multiple tables in **sysmaster** might return inconsistent results because the database server does not lock the tables during a join. You can join **sysmaster** tables with tables in other databases. However, to join **sysmaster** tables with tables in a nonlogging database, first make the nonlogging database the current database.

- The buildsmi Script
- The bldutil.sh Script

The buildsmi Script

When you bring the database server up for the first time, it runs a script called **buildsmi**, which is in the **etc** directory. This script builds the database and tables that support SMI. The database server requires approximately 1750 free pages of logical-log space to build the **sysmaster** database.

If you receive an error message that directs you to run the **buildsmi** script, a problem probably occurred while the database server was building the SMI database, tables, and views. When you use **buildsmi**, the existing **sysmaster** database is dropped and then re-created.

This script must be run as user **informix** on UNIX, or as a member of the **Informix-Admin** group on Windows, after ensuring that no connections to the sysmaster database are made during the build of the database. For example, if a Scheduler task is running when the **buildsmi** script commences, the script fails when the Scheduler attempts to access any of the sysmaster tables.

Errors issued while the **buildsmi** script runs are written (on UNIX) to the file **/tmp/buildsmi.out**, or on Windows to the file **%INFORMIXDIR%\etc\buildsmi_out.*%INFORMIXSERVER%***, where ***%INFORMIXSERVER%*** is the name of the Informix® instance.

The bldutil.sh Script

When you initialize the database server for the first time, it runs a script called **bldutil.sh** on UNIX or **bldutil.bat** on Windows. This script builds the **sysutils** database. If it fails, the database server creates an output file in the **tmp** directory. The output file is **bldutil.process_id** on UNIX and **bldutil.out** on Windows. The messages in this output file reflect errors that occurred during the script execution.

The System-Monitoring Interface

This section describes the SMI tables and how you access them to monitor the database server operation.

- Understanding the SMI Tables
- Accessing SMI tables

Understanding the SMI Tables

The SMI (system-monitoring interface) consists of tables and pseudo-tables that the database server maintains automatically. While the SMI tables appear to the user as tables, they are not recorded on disk as normal tables are. Instead, the database server constructs the tables in memory, on demand, based on information in shared memory at that instant. When you query an SMI table, the database server reads information from these shared-memory structures. Because the database server continually updates the data in shared memory, the information that SMI provides lets you examine the current state of your database server.

The SMI tables provide information about the following topics:

- Auditing
- Checkpoints
- Chunk I/O
- Chunks
- Database-logging status
- Dbspaces
- Disk usage
- Environment variables
- Extents
- Locks
- Networks
- SQL statement cache statistics
- SQL statements
- System profiling
- Tables
- User profiling
- Virtual-processor CPU usage

The data in the SMI tables changes dynamically as users access and modify databases that the database server manages.

Accessing SMI tables

Any user can use SQL SELECT statements to query an SMI table, but standard users cannot run statements other than the SELECT statement. Users who attempt to run other statements result in permission errors. The administrator can run SQL statements other than SELECT, but the results of such statements are unpredictable.

Tip: For more predictable results, query the views that are associated with each table instead of querying the tables directly.
If you query the **systabpaghdrs** table directly, you must specify an appropriate value for the `pg_partnum` parameter. The value is `pg_partnum > 1048576`. However, if you query the view that is associated with the **systabpaghdrs** table, you do not have to specify this value for the `pg_partnum` parameter.

Informix® includes the **sysadtinfo** and **sysaudit** tables. Only the user **informix** on UNIX or members of the **Informix-Admin** group on Windows can query the **sysadtinfo** and **sysaudit** tables.

You cannot use the **dbschema** or **dbexport** utilities on any of the tables in the **sysmaster** database. If you do, the database server generates the following error message:

```
Database has pseudo tables - can't build schema
```

- SELECT statements
  You can use SELECT statements on SMI tables wherever you can use SELECT against ordinary tables.
- Triggers and Event Alarms
- SPL and SMI Tables
- Locking and SMI Tables

SELECT statements

You can use SELECT statements on SMI tables wherever you can use SELECT against ordinary tables.

For example, you can use SELECT statements ordinary tables from DB-Access, in an SPL routine, with Informix® ESQL/C, and so on.
Restriction: You cannot meaningfully reference **rowid** when you query SMI tables. SELECT statements that use **rowid** do not return an error, but the results are unpredictable.

All standard SQL syntax, including joins between tables, sorting of output, and so on, works with SMI tables. For example, if you want to join an SMI table with a non-SMI table, name the SMI table with the following standard syntax:

```
sysmaster[@dbservername]:[owner.]tablename
```


Triggers and Event Alarms

Triggers based on changes to SMI tables never run. Although you can define triggers on SMI tables, triggers are activated only when an INSERT, UPDATE, or DELETE statement occurs on a table. The updates to the SMI data occur within the database server, without the use of SQL, so a trigger on an SMI table is never activated, even though the data returned by a SELECT statement indicates that it should be.

To create an event alarm, query for a particular condition at predefined intervals, and execute an SPL routine if the necessary conditions for the alarm are met.

SPL and SMI Tables

You can access SMI tables from within a SPL routine. When you reference SMI tables, use the same syntax that you use to reference a standard table.

Locking and SMI Tables

The information in the SMI tables changes based on the database server activity. However, the database server does not update the information using SQL statements. When you use SMI tables with an isolation level that locks objects, it prevents other users from accessing the object, but it does not prevent the data from changing. In this sense, all the SMI tables have a permanent Dirty Read isolation level.

The System-Monitoring Interface Tables

The **sysmaster** database contains many tables that you can use to monitor your system.

Tip: For each system-monitoring interface (SMI) table, there is a corresponding view with the same name. For the best results, query the views that are associated with tables instead of querying the underlying tables directly.
Many other tables in the **sysmaster** database are part of the system-monitoring interface but are not documented. Their schemas and column content can change from version to version. The **flags_text** table now contains more rows. To view the new rows, first drop and then re-create the **sysmaster** database.

The following table lists the SMI tables.

Table 1. SMI tables

Table	Description	Reference
sysadtinfo	Auditing configuration information	sysadtinfo
sysaudit	Auditing event masks	sysadtinfo
syscheckpoint	Checkpoint information	syscheckpoint
syschkio	Chunk I/O statistics	syschkio
syschunks	Chunk information	syschunks
syscluster	High-availability cluster information	syscluster
syscmsmsla	Connection Manager information	syscmsmsla
syscmsmtab	Connection Manager information	syscmsmtab
syscmsmunit	Information for each Connection Manager unit in a Connection Manager configuration file	syscmsmunit
syscompdicts_full	Compression dictionary information	syscompdicts_full
sysconfig	Configuration information	sysconfig
sysdatabases	Database information	sysdatabases
sysdbslocale	Locale information	sysdbslocale
sysdbspaces	Dbspace information	sysextents
sysdri	Data-replication information	sysdri
sysdual	Is a single-row table	sysdual

Table	Description	Reference
sysenv	Server startup environment	sysenv
sysenvses	Session-level environment variable	sysenvses
sysextents	Extent-allocation information	sysextents
sysextspaces	External spaces information	sysextspaces
sysha_lagtime	Secondary server lag-time statistics	sysha_lagtime Table
sysha_type	Information about connected servers	sysha_type
sysha_workload	Secondary server workload statistics	sysha_workload
sysipl	Index page logging information	sysipl
syslocks	Active locks information	syslocks
syslogs	Logical-log file information	syslogs
syslogfil	System log file information	syslogfil table
sysmgminfo	Memory Grant Manager and Parallel Data Query information	sysmgminfo
sysnetclienttype	Client type network activity	sysnetclienttype
sysnetglobal	Global network information	sysnetglobal
sysnetworkio	Network I/O	sysnetworkio table
sysonlinelog	Online log information	sysonlinelog
sysprofile	System-profile information	sysprofile
sysproxyagents	Information about all the proxy agent threads	sysproxyagents
sysproxydistributors	Proxy distributor information	sysproxydistributors
sysproxysessions	Information about sessions that use updatable secondary servers	sysproxysessions table
sysproxytxnops	Information about transactions that are run through each proxy distributor	sysproxytxnops table
sysproxytxns	Information about all of the current transactions that run through each proxy distributor	sysproxytxns table
sysptprof	Table information	sysptprof table
sysrsslog	RS secondary server information	sysrsslog
sysscblst	Memory by user	sysscblst
syssesprof	Counts of various user actions	syssesprof
syssesappinfo	Distributed Relational Database Architecture™ (DRDA) client-session information.	syssesappinfo
syssessions	Description of each user connected	syssessions
syssessiontempspaceusage	Contains information about each session's temp space usage.	syssessiontempspaceusage
syssmx	SMX (server multiplexer group) connection information	syssmx
syssmxses	SMX (server multiplexer group) session information	syssmxses
syssqexplain	SQL statement information that is enabled by the SET EXPLAIN statement	syssqexplain table
syssqltrace	SQL statement information	syssqltrace
syssqltrace_hvar	SQL statement tracing host variable information	syssqltrace_hvar
syssqltrace_info	SQL profile trace system information	syssqltrace_info
syssqltrace_iter	SQL statement iterators	syssqltrace_iter
syssrcrss	RS secondary server statistics	syssrcrss
syssrcsds	SD secondary server statistics	syssrcsds
systabnames	Database, owner, and table name for the tblspace **tblspace**	systabnames
systabpaghdrs	Page headers	None
systhreads	Wait statistics	systhreads
systrgrss	RS secondary server statistics	systrgrss
systrgsds	SD secondary server statistics	systrgsds
sysvpprof	User and system CPU used by each virtual processor	sysvpprof

- The sysutils Tables
- sysadtinfo
- sysaudit
- syschkio
  The **syschkio** system-monitoring interface table provides I/O statistics for individual chunks that the database server manages.
- syscheckpoint
- syschunks
  The **syschunks** table contains a description of each of the chunks that the database server manages.
- sysckptinfo
  The **sysckptinfo** system-monitoring interface table provides historical information about the previous twenty checkpoints.
- syscluster

- syscmsm
- syscmsmsla
- syscmsmtab
- syscmsmunit
- syscompdicts_full
  The **syscompdicts_full** table and the **syscompdicts** view provide information on all compression dictionaries. The only difference between the table and the view is that, for security purposes, the view does not contain the **dict_dictionary** column.
- sysconfig
- sysdatabases
- sysdbslocale
- sysdbspaces
  The **sysdbspaces** table contains a description of each of the storage spaces that the database server manages.
- sysdri
- sysdual
- sysenv
- sysenvses
- sysextents
- sysextspaces
- sysfeatures
- sysha_lagtime Table
  The **sysha_lagtime** table provides a history of the amount of time that it took to apply a log record on any of the secondary nodes.
- sysha_type
- sysha_workload
- sysipl
- syslocks
- syslogs
- syslogfil table
  The syslogfil table provides information about the logical log files.
- sysmgminfo
- sysnetclienttype
- sysnetglobal
- sysnetworkio table
  The **sysnetworkio** table contains information about the system network.
- sysonlinelog
- sysprofile
- sysproxyagents
- sysproxydistributors
- sysproxysessions table
  The **sysproxysessions** table contains information about each of the sessions that are using redirected-write functionality. This table is only valid on the secondary server.
- sysproxytxnops table
  The **sysproxytxnops** table contains information about each of the transactions that are running through each proxy distributor.
- sysproxytxns table
  The **sysproxytxns** table contains information about all of the current transactions that are running through each proxy distributor.
- sysptnhdr
  The **sysptrhdr** table contains information about partition headers.
- sysptprof table
  The **sysptprof** table lists information about a tblspace. Tblspaces correspond to tables.
- sysrepevtreg table
  Use the **sysrepevtreg** pseudo table to register for a pre-defined set of events from the Connection Manager.
- sysrepstats table
  Use the **sysrepstats** table to post events to Connection Manager.
- sysrsslog
- sysscblst
- syssscelem
- syssesappinfo
- syssesprof
- syssessions
- syssessiontempspaceusage
  The **syssessiontempspaceusage** table contains information about each session's temp space usage, includes both implicit and explicit temp tables.
- syssmx
- syssmxses
- syssqexplain table
  The **syssqexplain** pseudo table stores information about SQL queries.
- syssqltrace
- syssqltrace_hvar
  The **syssqltrace_hvar** table describes information about the SQL tracing host variable.
- syssqltrace_info
  The **syssqltrace_info** table describes information about the SQL profile trace system.
- syssqltrace_iter
- syssrcrss
- syssrcsds
  The **syssrcsds** table provides SD secondary server related statistics at the primary server.
- systabnames
- systhreads
- systrgrss
- systrgsds
  The **systrgsds** table provides SD secondary server related statistics at the SD secondary server.
- sysvpprof
- The SMI Tables Map

- Information from onstat in the SMI Tables

The sysutils Tables

ON-Bar uses the following tables in the **sysutils** database. For more information, see the *IBM® Informix® Backup and Restore Guide*.

Table
Description

bar_action
: Lists all backup and restore actions that are attempted against an object, except during a cold restore. Use the information in this table to track backup and restore history.

bar_instance
: Writes a record to this table for each successful backup. ON-Bar might later use the information for a restore operation.

bar_object
: Describes each backup object. This table provides a list of all storage spaces and logical logs from each database server for which at least one backup attempt was made.

bar_server
: Lists the database servers in an installation. This table is used to ensure that backup objects are returned to their proper places during a restore.

sysadtinfo

The **sysadtinfo** table contains information about the auditing configuration for the database server. For more information, see your *IBM® Informix® Security Guide*. You must be user **informix** or user **root** on UNIX or a member of the **Informix-Admin** group on Windows to retrieve information from the sysadtinfo table.

Column	Type	Description
adtmode	integer	Controls the level of auditing.
adterr	integer	Specifies how the database server behaves when it encounters an error while it writes an audit record.
adtsize	integer	Maximum size of an audit file
adtpath	char(256)	Directory where audit files are written
adtfile	integer	Number of the audit file

sysaudit

For each defined audit mask (that is, for each *username*), the **sysaudit** table contains flags that represent the database events that generate audit records. The **success** and **failure** columns represent the bitmasks that compose the audit masks. If a bit is set in both the **success** the and **failure** columns, the corresponding event generates an audit record whether or not the event succeeded.

You must be user **informix** or **user** root on UNIX or a member of the **Informix-Admin** group on Windows to retrieve information from the **sysaudit** table.

Use the **onaudit** utility to list or modify an audit mask. For information about **onaudit** and auditing, see your *IBM® Informix® Security Guide*.

Column	Type	Description
username	char(32)	Name of the mask
succ1	integer	Bitmask of the audit mask for success
succ2	integer	Bitmask of the audit mask for success
succ3	integer	Bitmask of the audit mask for success
succ4	integer	Bitmask of the audit mask for success
succ5	integer	Bitmask of the audit mask for success
fail1	integer	Bitmask of the audit mask for failure
fail2	integer	Bitmask of the audit mask for failure
fail3	integer	Bitmask of the audit mask for failure
fail4	integer	Bitmask of the audit mask for failure
fail5	integer	Bitmask of the audit mask for failure

syschkio

The **syschkio** system-monitoring interface table provides I/O statistics for individual chunks that the database server manages.

Column	Type	Description
chunknum	smallint	Chunk number
reads	integer	Number of physical reads
pagesread	integer	Number of pages read
writes	integer	Number of physical writes
pageswritten	integer	Number of pages written
mreads	integer	Number of physical reads (mirror)
mpagesread	integer	Number of pages read (mirror)
mwrites	integer	Number of physical writes (mirror)
mpageswritten	integer	Number of pages written (mirror)

syscheckpoint

The **syscheckpoint** table provides information and statistics about checkpoints.

Column	Type	Description
interval	integer	Number of checkpoints since the server was started
type	char(12)	Hard or Interval
caller	char(10)	Caller of the checkpoint
clock_time	integer	Time of day the checkpoint occurred
crit_time	float	Time spent waiting for the critical section to be released
flush_time	float	Time spent flushing pages to disk
cp_time	float	Duration from checkpoint pending until checkpoint done
n_dirty_buffs	integer	Number of dirty buffers
plogs_per_sec	integer	Number of physical log pages processed in a second
llogs_per_sec	integer	Number of logical log pages processed in a second
dskflush_per_sec	integer	Number of buffer pool pages flushed in a second
ckpt_logid	integer	Unique id of the logical log at the checkpoint
ckpt_logpos	integer	Position of the logical log at the checkpoint
physused	integer	Number of pages used in the physical log
logused	integer	Number of pages used in the logical log
n_crit_waits	integer	Number of users who had to wait to enter a critical section
tot_crit_wait	float	Duration spent waiting for all users waiting at the checkpoint critical section block
longest_crit_wait	float	Longest critical section wait
block_time	float	Duration of the checkpoint that blocked the system

syschunks

The **syschunks** table contains a description of each of the chunks that the database server manages.

In the **flags** and **mflags** columns, each bit position represents a separate flag. Thus, it might be easier to read values in the **flags** and **mflags** columns if the values are returned by the HEX function.

Table 1. The syschunks table

Column	Type	Description
chknum	smallint	Chunk number
dbsnum	smallint	Dbspace number
nxchknum	smallint	Number of the next chunk in this dbspace

Column	Type	Description
chksize	integer	Number of pages in this chunk (in units of system default page size)
offset	integer	Page offset of the chunk in its device or path
pagesize	integer	Page size (in bytes)
nfree	integer	Number of free pages in the chunk The amount of free space depends on the type of space: • dbspace = multiply the number of free pages by the system default page size of either 2 KB or 4 KB. • blobspace = multiply the number of free pages by the blobpage size. • sbspace = multiply the number of free pages by the sbpage size (which is the same as the system default page size).
is_offline	integer	1 = chunk is offline 0 = chunk is online
is_recovering	integer	1 = the chunk is being recovered 0 = the chunk is not being recovered
is_blobchunk	integer	1 = the chunk is in a blobspace 0 = the chunk is not in a blobspace
is_sbchunk	integer	1 = the chunk is an sbspace 0 = the chunk is not in an sbspace
is_inconsistent	integer	1 = the chunk is undergoing logical restore 0 = the chunk is not being restored
is_extendable	integer	1 = the chunk is extendable 0 = the chunk is not extendable
flags	smallint	The flags have the following numeric and hexadecimal values and meanings: • 16 (0x0010) = Chunk is a mirrored chunk • 32 (0x0020) = Chunk is in offline mode • 64 (0x0040) = Chunk is in online mode • 128 (0x0080) = Chunk is in recovery mode • 256 (0x0100) = Chunk is mirrored • 512 (0x0200) = Chunk is part of a blobspace • 1024 (0x0400) = Chunk is being dropped • 4096 (0x1000) = Chunk is inconsistent • 8192 (0x2000) = Chunk is extendable • 16384 (0x4000) = Chunk was added during roll forward • 32768 (0x8000) = Chunk was renamed • 65536 (0x10000) = Chunk uses big chunk page header • 131072 (0x20000) = Chunk has a tblspace tblspace extent • 262144 (0x40000) = No checkpoint was completed since this chunk was initialized (primarily for internal use)
fname	char(256)	Path name for the file or device of this chunk
mdsize	integer	Size in pages of the metadata area of a chunk that belongs to an sbspace. -1 = the chunk does not belong to an sbspace.
mfname	char(256)	Path name for the file or device of the mirrored chunk, if any
moffset	integer	Page offset of the mirrored chunk
mis_offline	integer	1 = mirror is offline 0 = mirror is online
mis_recovering	integer	1 = mirror is being recovered 0 = mirror is not being recovered
mflags	smallint	Mirrored chunk flags; values and meanings are the same as the **flags** column.
udfree	integer	Free space in pages within the user data area of a chunk that belongs to an sbspace. -1 = the chunk does not belong to an sbspace.
udsize	integer	Size in pages of the user data area of a chunk that belongs to an sbspace. -1 = the chunk does not belong to an sbspace.

sysckptinfo

The **sysckptinfo** system-monitoring interface table provides historical information about the previous twenty checkpoints.

Column	Type	Description

Column	Type	Description
ckpt_status	int	0x0011 = A checkpoint was blocked because the physical log ran out of resources. 0x0021 = A checkpoint was blocked because the logical log ran out of resources. 0x0041 = A checkpoint was blocked because transactions were too long. 0x1000 = The physical log is too small. 0x2000 = The logical log space is too small. 0x4000 = The physical log is too small for RTO.
plogs_per_S	int	Average rate of physical logging activity.
llogs_per_S	int	Average rate of logical logging activity.
dskF_per_S	int	Average rate of pages flushed to disk.
longest_dskF	int	Longest duration of time to flush the buffer pool to the disk during checkpoint processing.
dirty_pgs_S	int	Average rate of pages being modified.
sug_plog_sz	int	Suggested physical log size.
sug_llog_sz	int	Suggested logical log space size.
ras_plog_sp	int	Rate at which fast recovery can restore the physical log.
ras_llog_sp	int	Rate at which fast recovery can replay the logical log.
boottime	int	Time it takes for the server to boot shared memory and open chunks.
auto-ckpts	int	1 = on, 0 = off.
auto_lru	int	1 = on, 0 = off.
cur_intvl	int	Current® checkpoint interval id.
auto_aiovp	int	1 = on, 0 = off.

Related reference:
onstat -g ckp command: Print checkpoint history and configuration recommendations

syscluster

The **syscluster** system catalog table stores information about servers in a high-availability cluster. The **syscluster** table has the following columns.

Table 1. **syscluster** table information

Column	Type	Explanation
name	CHAR(128)	The name of the primary server.
role	CHAR(1)	Code to indicate whether the server is a primary server or secondary server.
syncmode	CHAR(8)	The synchronization mode between the primary server and the secondary server: sync or async.
nodetype	CHAR(8)	The type of server: HDR, RSS, or SDS.
supports_updates	CHAR(1)	Indicates whether client applications can perform update, insert, and delete operations on the secondary server (as specified by the UPDATABLE_SECONDARY configuration parameter).
server_status	CHAR(32)	Indicates the status of the secondary server.
connection_status	CHAR(32)	Indicates the connection status of the secondary server.
delayed_apply	INTEGER	Indicates whether the secondary server waits for a specified amount of time before applying logs (as specified by the DELAY_APPLY configuration parameter).
stop_apply	CHAR(24)	Indicates whether the secondary server has stopped applying log files received from the primary server (as specified by the STOP_APPLY configuration parameter).
logid_sent	INTEGER	Indicates the log ID of the most recent log page sent by the primary server to the secondary server.
logpage_sent	INTEGER	Indicates the page number of the most recent log page sent by the primary server to the secondary server.
logid_acked	INTEGER	Indicates the log ID of the most recent log page the secondary server acknowledged.
logpage_acked	INTEGER	Indicates the page number of the most recent log page the secondary server acknowledged.
ack_time	DATETIME YEAR TO SECOND	Indicates the date and time of the last acknowledged log.
sdscycle	INTEGER	Indicates the cycle number to which the primary server has advanced. Used internally by support to monitor coordination of the primary server with the secondary server.
sdscycle_acked	INTEGER	Indicates the cycle number that the shared disk secondary server has acknowledged. Used internally by support to monitor coordination of the primary server with the secondary server.

syscmsm

The **syscmsm** table is a view of the **syscmsmtab** and **syscmsmsla** tables. It contains Connection Manager service level agreement (SLA) information. The table is updated one time every five seconds.

Table 1. **syscmsm** table information

Column	Type	Description
sid	integer	Connection Manager session ID
name	char(128)	Connection Manager name
host	char(256)	Host name
unit	char(128)	Unit name
type	char(128)	Unit type
servers	char(1024)	Unit servers
foc	char(128)	Failover configuration (FOC)
flag	integer	Arbitrator flag. A value of 1 indicates that the Connection Manager Arbitrator is active. A value of 0 indicates that the Arbitrator is inactive.
sla_name	char(128)	SLA name
sla_define	char(128)	SLA definition
connections	integer	Number of connections that are made through Connection Manager

syscmsmsla

The **syscmsmsla** table contains Connection Manager service level agreement (SLA) information. The table is updated one time every five seconds.

Table 1. **syscmsmsla** table information

Column	Type	Description
address	int8	CMSLA internal address
sid	integer	Connection Manager session ID
sla_name	char(128)	SLA name
sla_define	char(128)	SLA define
connections	integer	Number of connections made through Connection Manager

syscmsmtab

The **syscmsmtab** table contains Connection Manager information.

Table 1. **syscmsmtab** table information

Column	Type	Description
address	int8	Connection Manager internal address
sid	integer	Connection Manager session ID
name	char(128)	Connection Manager name
host	char(256)	Host name
flag	integer	Arbitrator flag. A value of 1 indicates that the Connection Manager Arbitrator is active. A value of 0 indicates that the Arbitrator is inactive.

syscmsmunit

The **syscmsmunit** table contains information for each Connection Manager unit in a Connection Manager configuration file.

Table 1. **syscmsmunit** table information

Column	Type	Description

Column	Type	Description
address	int8	Connection Manager internal address
sid	integer	Connection Manager session ID
unit	char(128)	Unit name
type	char(128)	Unit type
servers	char(1024)	Unit servers
foc	char(128)	Failover configuration (FOC)
flag	integer	Arbitrator flag. A value of 1 indicates that the Connection Manager Arbitrator is active. A value of 0 indicates that the Arbitrator is inactive.

syscompdicts_full

The **syscompdicts_full** table and the **syscompdicts** view provide information on all compression dictionaries. The only difference between the table and the view is that, for security purposes, the view does not contain the **dict_dictionary** column.

Only user **informix** can retrieve information from the **syscompdicts_full** table. The **syscompdicts** view is not restricted to user **informix**.

The following table shows the information that the **syscompdicts_full** table and the **syscompdicts** view provide for each compression dictionary.

Table 1. Compression Dictionary Information

Column	Type	Description
dict_partnum	integer	Partition number to which the compression dictionary applies
dict_code_version	integer	Version of the code that is creating the compression dictionary 1 is the first version.
dict_dbsnum	integer	Number of the dbspace that the dictionary resides in
dict_create_timestamp	integer	Timestamp that shows when the dictionary was created
dict_create_loguniqid	integer	Unique ID for the logical log that was active when the dictionary was created
dict_create_logpos	integer	Position within the logical log when the dictionary was created
dict_drop_timestamp	integer	Timestamp that shows when the dictionary was dropped.
dict_drop_loguniqid	integer	Unique ID for the logical log that was created when the dictionary was dropped.
dict_drop_logpos	integer	Position within the logical log when the dictionary was dropped.
dict_dictionary	byte	Compression dictionary binary object This column is not included in the **syscompdicts** view.

Sample syscompdicts information

A row of information in the **syscompdicts** view could displays columns containing this information:

```
dict_partnum        1048939
dict_code_version   1
dict_dbsnum         1
dict_create_times+  1231357656
dict_create_logun+  11
dict_create_logpos  1695768
dict_drop_timesta+  0
dict_drop_loguniq+  0
dict_drop_logpos    0
```

You can use an UNLOAD statement to unload the compression dictionary to a compression dictionary file, as follows:

```
UNLOAD TO 'compression_dictionary_file'
   SELECT * FROM sysmaster:syscompdicts_full;
```


sysconfig

The **sysconfig** table describes the effective, original, and default values of the configuration parameters. For more information about the ONCONFIG file and the configuration parameters, see Database configuration parameters.

Column	Type	Description
cf_id	integer	Unique numeric identifier

Column	Type	Description
cf_name	char(128)	Configuration parameter name
cf_flags	integer	Reserved for future use
cf_original	char(256)	Value in the ONCONFIG file at boot time
cf_effective	char(256)	Value currently in use
cf_default	char(256)	Value provided by the database server if no value is specified in the ONCONFIG file

sysdatabases

The **sysdatabases** view describes each database that the database server manages.

Table 1. **sysdatabases** view information

Column	Type	Description	
name	char(128)	Database name	
partnum	integer	The partition number (tblspace identifier) for the systables table for the database	
owner	char(32)	User ID of the creator of the database	
created	date	Date created	
is_logging	integer	1 If logging is active, 0 if not	
is_buff_log	integer	1 If buffered logging, 0 if not	
is_ansi	integer	1 If ANSI/ISO-compliant, 0 if not	
is_nls	integer	1 If GLS-enabled, 0 if not	
is_case_insens	integer	1 If case-insensitive for NCHAR and NVARCHAR columns, 0 if not	
flags	smallint	Logging flags (hex values)	
		0	No logging
		1	Unbuffered logging
		2	Buffered logging
		4	ANSI/ISO-compliant database
		8	Read-only database
		10	GLS database
		20	Checking of the logging mode of **syscdr** database bypassed
		100	Changed status to buffered logging
		200	Changed status to unbuffered logging
		400	Changed status to ANSI/ISO compliant
		800	Database logging turned off
		1000	Long ID support enabled

sysdbslocale

The **sysdbslocale** table lists the locale of each database that the database server manages.

Table 1. **sysdbslocale** table information

Column	Type	Description
dbs_dbsname	char(128)	Database name
dbs_collate	char(32)	The locale of the database

sysdbspaces

The **sysdbspaces** table contains a description of each of the storage spaces that the database server manages.

Table 1. **sysdbspaces** table information

Column	Type	Description
dbsnum	smallint	Space number
name	char(128)	Space name
owner	char(32)	User ID of owner of the space
fchunk	smallint	Number of the first chunk in the space
nchunks	smallint	Number of chunks in the space
create_size	decimal	The minimum size of a chunk that can be created for this space using the storage pool.
extend_size	decimal	The minimum size by which a chunk in this storage space can be extended, either manually or automatically.
pagesize	integer	Page size
is_mirrored	integer	1 = The space is mirrored 0 = The space is not mirrored
is_blobspace	integer	1 = The space is a blobspace 0 = The space is not a blobspace
is_sbspace	integer	1 = The space is an sbspace 0 = The space is not an sbspace
is_temp	integer	1 = The space is a temporary dbspace 0 = The space is not a temporary dbspace
is_encrypted	integer	1 = The space is encrypted 0 = The space is not encrypted
flags	smallint	Each bit position represents a separate flag. Thus, it might be easier to read values in this column if you return the values with the HEX function. • 1 (0x0001) = The space has no mirror • 2 (0x0002) = The space uses mirroring • 4 (0x0004) = Mirroring is disabled • 8 (0x0008) = The space is newly mirrored • 16 (0x0010) = The space is a blobspace • 32 (0x0020) = The blobspace is on removable media • 128 (0x0080) = The blobspace has been dropped • 512 (0x0200) = The space is being recovered • 1024 (0x0400) = The space has been physically recovered • 2048 (0x0800) = The space is in logical recovery • 32768 (0x8000) = The space is an sbspace

Related reference:
DISK_ENCRYPTION configuration parameter

sysdri

The **sysdri** table provides information about the High-Availability Data-Replication status of the database server.

Column	Type	Description
type	char(50)	High-Availability Data Replication type Possible values: • primary • secondary • standard • not initialized
state	char(50)	State of High-Availability Data Replication Possible values: • off • on • connecting • failure • read-only
name	char(128)	The name of the other database server in the High-Availability Data-Replication pair
intvl	integer	The High-Availability Data-Replication interval
timeout	integer	The High-Availability Data-Replication timeout value for this database server
lostfound	char(256)	The pathname to the lost-and-found file

sysdual

The **sysdual** table returns exactly one column and one row.

Column	Type	Description
dummy	char(1)	Dummy columns returning "X"

sysenv

The **sysenv** table displays the startup environment settings of the database server.

Column	Type	Description
env_id	integer	Identifier variable number
env_name	char(128)	Environment variable name
env_value	char(512)	Environment variable value

sysenvses

The **sysenvses** table displays the environment variable at the session level.

Column	Type	Description
envses_sid	integer	Session id
envses_id	integer	Identifier variable number
envses_name	char(128)	Session environment variable name
envses_value	char(512)	Session environment variable value

sysextents

The **sysextents** table provides information about extent allocation.

Column	Type	Description
dbsname	char(128)	Database name
tabname	char(128)	Table name
chunk	integer	Chunk number
offset	integer	Number of pages into the chunk where the extent begins
size	integer	Size of the extent, in pages

sysextspaces

The sysextspaces table provides information about external spaces. Indexes for the **id** column and the **name** column allow only unique values.

Column	Type	Description
id	integer	External space ID
name	char(128)	External space name
owner	char(32)	External space owner
flags	integer	External space flags (reserved for future use)
refcnt	integer	External space reference count.
locsize	integer	Size of external space location, in bytes
location	char (256)	Location of external space

sysfeatures

The **sysfeatures** view provides general information about various features of the Informix® database server instance. The **sysfeatures** view is created from an internal table named **syslicenseinfo**, which is stored permanently on the disk. When the database server instances are initialized, the table is pre-allocated with a fixed size which allows tracking of 260 weeks of data. The data wraps every five years.

Metrics are sampled every 15 minutes and only the highest values during the particular week are stored. Each row in the table contains data only for the specific week it represents.

Column	Type	Description
week	smallint	The week that the information was recorded.
year	smallint	The year that the information was recorded.
version	char(12)	The Informix server version.
max_cpu_vps	smallint	The maximum number of CPU virtual processors.
max_vps	smallint	The maximum number of virtual processors.
max_conns	integer	The maximum number of concurrent physical connections on a standalone or high-availability cluster primary server instance.
max_sec_conns	integer	The maximum number of concurrent physical connections on an HDR secondary or RS secondary server instance.
max_sds_clones	smallint	The maximum number of SD secondary server instances connected to the primary server.
max_rss_clones	smallint	The maximum number of RS secondary server instances connected to the primary server.
total_size	integer	The maximum disk space allocated in all chunks (in megabytes).
total_size_used	integer	The maximum disk space used in all chunks (in megabytes).
max_memory	integer	The maximum memory allocated in all segments (in megabytes).
max_memory _used	integer	The maximum memory used in all segments (in megabytes).
is_primary	integer	Indicates whether the server was a primary server in a particular week; 1 = yes, 0 = no.
is_secondary	integer	Indicates whether the server was an HDR secondary server in a particular week; 1 = yes, 0 = no.
is_sds	integer	Indicates whether the server was an SD secondary server in a particular week; 1 = yes, 0 = no (not implemented; always 0).
is_rss	integer	Indicates whether the server was an RS secondary server in a particular week; 1 = yes, 0 = no.
is_er	integer	Indicates whether the server was an enterprise replication server in a particular week; 1 = yes, 0 = no.
is_pdq	integer	Indicates whether the PDQ feature was used on the server instance in the particular week; 1 = yes, 0 = no.

sysha_lagtime Table

The **sysha_lagtime** table provides a history of the amount of time that it took to apply a log record on any of the secondary nodes.

The **sysha_lagtime** table contains a history of the last 20 samplings performed for a particular secondary server.

Table 1. **sysha_lagtime** table information

Column	Type	Description
lt_secondary	CHAR(128)	Name of secondary server
lt_time_last_update	INTEGER	Time at which log record was last updated
lt_lagtime_1	FLOAT	Amount of time required to apply log record for the most recent five-second interval
lt_lagtime_2	FLOAT	Amount of time required to apply log record for the second most recent five-second interval
lt_lagtime_3	FLOAT	Amount of time required to apply log record for the third most recent five-second interval
lt_lagtime_4	FLOAT	Amount of time required to apply log record for the fourth most recent five-second interval
lt_lagtime_5	FLOAT	Amount of time required to apply log record for the fifth most recent five-second interval
lt_lagtime_6	FLOAT	Amount of time required to apply log record for the sixth most recent five-second interval
lt_lagtime_7	FLOAT	Amount of time required to apply log record for the seventh most recent five-second interval
lt_lagtime_8	FLOAT	Amount of time required to apply log record for the eighth most recent five-second interval
lt_lagtime_9	FLOAT	Amount of time required to apply log record for the ninth most recent five-second interval
lt_lagtime_10	FLOAT	Amount of time required to apply log record for the tenth most recent five-second interval
lt_lagtime_11	FLOAT	Amount of time required to apply log record for the eleventh most recent five-second interval
lt_lagtime_12	FLOAT	Amount of time required to apply log record for the twelfth most recent five-second interval

Column	Type	Description
lt_lagtime_13	FLOAT	Amount of time required to apply log record for the thirteenth most recent five-second interval
lt_lagtime_14	FLOAT	Amount of time required to apply log record for the fourteenth most recent five-second interval
lt_lagtime_15	FLOAT	Amount of time required to apply log record for the fifteenth most recent five-second interval
lt_lagtime_16	FLOAT	Amount of time required to apply log record for the sixteenth most recent five-second interval
lt_lagtime_17	FLOAT	Amount of time required to apply log record for the seventeenth most recent five-second interval
lt_lagtime_18	FLOAT	Amount of time required to apply log record for the eighteenth most recent five-second interval
lt_lagtime_19	FLOAT	Amount of time required to apply log record for the nineteenth most recent five-second interval
lt_lagtime_20	FLOAT	Amount of time required to apply log record for the twentieth most recent five-second interval

sysha_type

The **sysha_type** table is a single row table that is used to describe the type of server that is connected.

Table 1. **sysha_type** table information

Column	Type	Description
ha_type	integer	Server type (see table below)
ha_primary	char(128)	Server name (see table below)

Table 2. Descriptions for the values in the **sysha_type** table. This table describes the values in the **sysha_type** table.

Value of *ha_type*	Value of *ha_primary*	Description
0	NULL	Not part of a high-availability environment
1	*<primary server name>*	Primary server
2	*<primary server name>*	HDR secondary server
3	*<primary server name>*	SD secondary server
4	*<primary server name>*	RS secondary server

sysha_workload

The **sysha_workload** table contains workload statistics on each of the secondary servers.

Table 1. **sysha_workload** table information

Column	Type	Description
wl_secondary	char(128)	Name of secondary server
wl_time_last_update	integer	Time at which workload last updated
wl_type	char(12)	This row contains the ready queue size, user CPU time, and system CPU time
wl_workload_1	float	Most recent workload activity
wl_workload_2	float	Second most recent workload activity
wl_workload_3	float	Third most recent workload activity
wl_workload_4	float	Fourth most recent workload activity
wl_workload_5	float	Fifth most recent workload activity
wl_workload_6	float	Sixth most recent workload activity
wl_workload_7	float	Seventh most recent workload activity
wl_workload_8	float	Eighth most recent workload activity
wl_workload_9	float	Ninth most recent workload activity
wl_workload_10	float	Tenth most recent workload activity
wl_workload_11	float	Eleventh most recent workload activity
wl_workload_12	float	Twelfth most recent workload activity
wl_workload_13	float	Thirteenth most recent workload activity
wl_workload_14	float	Fourteenth most recent workload activity
wl_workload_15	float	Fifteenth most recent workload activity

Column	Type	Description
wl_workload_16	float	Sixteenth most recent workload activity
wl_workload_17	float	Seventeenth most recent workload activity
wl_workload_18	float	Eighteenth most recent workload activity
wl_workload_19	float	Nineteenth most recent workload activity
wl_workload_20	float	Twentieth most recent workload activity

sysipl

The **sysipl** table provides information about the status of index page logging at the primary server.

Table 1. **sysipl** table information

Column	Type	Description
ipl_status	integer	Index page logging status
ipl_time	integer	Time at which index page logging was enabled

syslocks

The **syslocks** table provides information about all the currently active locks in the database server.

Table 1. **syslocks** table information

Column	Type	Description	
dbsname	char(128)	Database name	
tabname	char(128)	Table name	
rowidlk	integer	Real rowid, if it is an index key lock	
keynum	smallint	Key number of index key lock	
type	char(4)	Type of lock	
		B	Byte lock
		IS	Intent shared lock
		S	Shared lock
		XS	Shared key value held by a repeatable reader
		U	Update lock
		IX	Intent exclusive lock
		SIX	Shared intent exclusive lock
		X	Exclusive lock
		XR	Exclusive key value held by a repeatable reader
owner	integer	Session ID of the lock owner	
waiter	integer	Session ID of the user waiting for the lock. If more than one user is waiting, only the first session ID appears.	

syslogs

The **syslogs** table provides information about space use in logical-log files. In the **flags** column, each bit position represents a separate flag. For example, for a log file, the **flags** column can have flags set for both current log file and temporary log file. Thus, it might be easier to read values in the **flags** column if the values are returned using the HEX function.

Table 1. **syslogs** table information

Column	Type	Description
number	smallint	Logical-log file number
uniqid	integer	Log-file ID
size	integer	Number of pages in the log file
used	integer	Number of pages used in the log file
is_used	integer	1 If file is used, 0 if not

Column	Type	Description		
is_current	integer	1 If file is the current file, 0 if not		
is_backed_up	integer	1 If file has been backed up, 0 if not		
is_new	integer	1 If the log has been added since the last level-0 dbspace backup, 0 if not		
is_archived	integer	1 If file has been placed on the backup tape, 0 if not		
is_temp	integer	1 If the file is flagged as a temporary log file, 0 if not		
flags	smallint	Flags	Hexadecimal	Meaning
		1	0x01	Log file is in use
		2	0x02	File is current log file
		4	0x04	Log file has been backed up
		8	0x08	File is newly added log file
		16	0x10	Log file has been written to dbspace backup media
		32	0x20	Log is a temporary log file

syslogfil table

The syslogfil table provides information about the logical log files.

Table 1. Information about the columns in the syslogfil table.

Column	Type	Description
address	int8	Memory address of the logfile structure
number	small integer	Log file number
flags	integer	For a description of the values and their meanings, see the **Flag values** section below.
fillstamp	integer	Internal timestamp when the log file was filled
filltime	integer	UNIX time when the log file was filled
uniqid	integer	Unique ID for the log file
chunk	integer	Number of the chunk that contains the log file
offset	integer	Page offset in the chunk where log file begins
size	integer	Total number of pages in the log file
used	integer	Number of pages used in the log file

Flag values

The flag values correspond to many of the flag values for the **onstat -l** command.

Hexadecimal	Onstat -l flag value	Meaning
0x1	U	Log file is in use
0x2	C	File is current log file
0x4	B	Log file has been backed up
0x8	A	File is a newly added log file
0x20	None	A temporary log file
0x40	D	Log file will be dropped after the file is archived
0x4000	L	Log file contains the last checkpoint written

sysmgminfo

The **sysmgminfo** table provides an overview of the Memory Grant Manager (MGM) and Parallel Data Query (PDQ) information.

Table 1. **sysmgminfo** table information

Column	Type	Description
max_query	integer	Maximum number of active queries allowed
total_mem	integer	Total MGM memory
avail_mem	integer	Free MGM memory

Column	Type	Description
total_seq	integer	Total number of sequential scans
avail_seq	integer	Unused sequential scans
active	integer	Number of active MGM queries
ready	integer	Number of ready MGM queries
min_free_mem	integer	Minimum free MGM memory
avg_free_mem	float	Average free MGM memory
std_free_mem	float	Standard free MGM memory
min_free_seq	integer	Minimum free MGM sequential scans
avg_free_seq	float	Average free MGM sequential scans
std_free seq	float	Standard free MGM sequential scans
max_active	integer	Maximum active MGM SQL operations
cnt_active	integer	Number of active MGM SQL operations
avg_active	float	Average active MGM SQL operations
std_active	float	Standard active MGM SQL operations
max_ready	integer	Maximum ready MGM SQL operations
cnt_ready	integer	Number of ready MGM SQL operations
avg_ready	float	Average ready MGM SQL operations
std_ready	float	Standard ready MGM SQL operations

sysnetclienttype

The **sysnetclienttype** table provides an overview of the network activity for each client type.

Column	Type	Description
nc_cons_allowed	integer	Whether or not connections are allowed
nc_accepted	integer	Number of connections that were accepted
nc_rejected	integer	Number of network connections that were rejected
nc_reads	int8	Number of network reads for this client type
nc_writes	int8	Number of network writes for this client type
nc_name	char(18)	Name of the client type

sysnetglobal

The **sysnetglobal** table provides an overview of the system network.

Column	Type	Description
ng_reads	int8	Number of network reads
ng_writes	int8	Number of network writes
ng_connects	int8	Number of network connections
ng_his_read_count	int8	Number of network reads by users who have disconnected **ng_his_read_bytes**
ng_his_read_bytes	int8	Data transferred to the server by users who have disconnected
ng_his_write_count	int8	Number of network writes by users who have disconnected
ng_his_write_bytes	int8	Data transferred to the client by users who have disconnected
ng_num_netscbs	integer	Number of network subscribers
ng_max_netscbs	integer	Maximum number of network subscribers
ng_free_thres	integer	Threshold for the maximum number of freed buffers in the buffer list
ng_free_cnt	integer	Number of times the ng_free_thres limit has been reached
ng_wait_thres	integer	Threshold for the maximum number of buffers that can be held in the buffer list for one connection
ng_wait_cnt	integer	Number of times the ng_wait_thres limit has been reached
ng_pvt_thres	integer	Threshold for the maximum number of freed buffers in the private buffer queue

Column	Type	Description
ng_netbuf_size	integer	Size of the transport network buffers
ng_buf_alloc	integer	Number of network buffers allocated
ng_buf_alloc_max	integer	Maximum value of allocated network buffers
ng_netscb_id	integer	Next netscb id

sysnetworkio table

The **sysnetworkio** table contains information about the system network.

Column	Type	Description
net_id	integer	Netscb id
sid	integer	Session id
net_netscb	int8	Netscb prt
net_client_type	integer	Client type Int
net_client_name	char(12)	Client protocol name
net_read_cnt	int8	Number of network reads
net_write_cnt	int8	Number of network writes
net_open_time	integer	Time this session connected
net_last_read	integer	Time of the last read from the network
net_last_write	integer	Time of the last write from the network
net_stage	integer	Connect / Disconnect / Receive
net_options	integer	Options from sqlhosts
net_protocol	integer	Protocol
net_type	char(10)	Type of network protocol
net_server_fd	integer	Server fd
net_poll_thread	integer	Poll thread

sysonlinelog

The **sysonlinelog** table provides a view of the information stored in the online.log file.

Column	Type	Description
offset	int8	File offset
next_offset	int8	Offset to the next message
line	char(4096)	Single line of text from the file

sysprofile

The **sysprofile** table contains profile information about the database server.

Column	Type	Description
name	char(13)	Name of profiled event. (See table that follows for a list of possible events.)
value	integer	Value of profiled event. (See table that follows for a list of possible events.)

The following table lists the events that, together with a corresponding value, make up the rows of the **sysprofile** table.

Events Profiled in sysprofile	Description
dskreads	Number of actual reads from disk
bufreads	Number of reads from shared memory
dskwrites	Actual number of writes to disk
bufwrites	Number of writes to shared memory

Events Profiled in sysprofile	Description
isamtot	Total number of calls
isopens	isopen calls
isstarts	isstart calls
isreads	isread calls
iswrites	iswrite calls
isrewrites	isrewrite calls
isdeletes	isdelete calls
iscommits	iscommit calls
isrollbacks	isrollback calls
ovlock	Overflow lock table
ovuser	Overflow user table
ovtrans	Overflow transaction table
latchwts	Latch request waits
bufwts	Buffer waits
lockreqs	Lock requests
lockwts	Lock waits
ckptwts	Checkpoint waits
deadlks	Deadlocks
lktouts	Deadlock time-outs
numckpts	Number checkpoints
plgpagewrites	Physical-log pages written
plgwrites	Physical-log writes
llgrecs	Logical-log records
llgpagewrites	Logical-log writes
llgwrites	Logical-log pages written
pagreads	Page reads
pagwrites	Page writes
flushes	Buffer-pool flushes
compress	Page compresses
fgwrites	Foreground writes
lruwrites	Least-recently used (LRU) writes
chunkwrites	Writes during a checkpoint
btradata	Read-ahead data pages read through index leaf node
btraidx	Read-ahead data pages read through index branch or root node
dpra	Data pages read into memory with read-ahead feature
rapgs_used	Read-ahead data pages that user used
seqscans	Sequential scans
totalsorts	Total sorts
memsorts	Sorts that fit in memory
disksorts	Sorts that did not fit in memory
maxsortspace	Maximum disk space used by a sort

sysproxyagents

The **sysproxyagents** table contains information about all proxy agent threads. Proxy agent threads run on the primary server and accept requests from secondary servers to process DML operations. The primary server also contains a proxy distributor that handles secondary server updates. Secondary servers determine how many instances of the proxy distributor to create based on the UPDATABLE_SECONDARY setting in the secondary server's ONCONFIG file.

Column	Type	Description
tid	integer	Transaction ID of the proxy agent thread running on the primary server. This ID is created by the proxy distributor to handle work from the secondary server session.
flags	integer	Flags of the proxy agent thread.

Column	Type	Description
proxy_id	integer	ID of the proxy distributor on behalf of the currently executing proxy agent thread (TID).
source_session_id	integer	ID of the user's session on the secondary server.
proxy_txn_id	integer	Number of the current transaction. These numbers are unique to the proxy distributor.
current_seq	integer	The sequence number of the current operation in the current transaction.
sqlerrno	integer	Error number of any SQL error (or 0 on success)
iserrno	integer	Error number of any ISAM/RSAM error (or 0 on success)

sysproxydistributors

The **sysproxydistributors** table contains information about the proxy distributors.

On the primary server, this table contains information about all of the proxy distributors in a high-availability cluster. On a secondary server, this table contains information about only those proxy distributors that are assigned to process updates to the secondary server.

Column	Type	Description
node_name	char	Name of the secondary server as it is known by the primary server (for example, INFORMIXSERVER, HA_ALIAS, and so on).
proxy_id	integer	ID of the proxy distributor. These IDs are unique within a high-availability cluster.
transaction_count	integer	Number of transactions currently being processed by the proxy distributor.
hot_row_total	integer	Total number of hot rows ever handled by the proxy distributor. A hot row is a row on a secondary server that is updated multiple times by more than one client. When a row is updated multiple times, the secondary server reads the before image from the primary server by placing an update lock on the row if the most recent update operation from a different session is not replayed on the secondary server.

sysproxysessions table

The **sysproxysessions** table contains information about each of the sessions that are using redirected-write functionality. This table is only valid on the secondary server.

The following table provides information about the columns in the **sysproxysessions** table:

Column	Type	Description
session_id	integer	ID of a user's session on the secondary server.
proxy_id	integer	ID of the proxy distributor on behalf of which the proxy agent thread (TID) is running
proxy_tid	integer	Transaction ID of the proxy agent thread running on the primary server. This ID is created by the proxy distributor to handle work from the secondary server session.
proxy_txn_id	integer	Number of the current transaction. These numbers are unique to the proxy distributor.
current_seq	integer	The sequence number of the current operation in the current transaction.
pending_ops	integer	The number of operations buffered on the secondary server that have not yet been sent to the primary server.
reference_count	integer	Indicates the number of threads (for example, sqlexec, sync reply, recovery, and so on) that are using the information for this transaction. When reference_count equals 0, the transaction processing has completed (either successfully or unsuccessfully).

sysproxytxnops table

The **sysproxytxnops** table contains information about each of the transactions that are running through each proxy distributor.

On the primary server, this table contains information about all of the proxy distributors in the high-availability cluster. On a secondary server, this table only contains information about the proxy distributors used to process updates to the secondary server.

The following table provides information about the columns in the **sysproxytxnops** table:

Column	Type	Description
proxy_id	integer	ID of the proxy distributor. These IDs are unique within a high-availability cluster.
proxy_txn_id	integer	Number of the transaction. These numbers are unique to the proxy distributor.
sequence_number	integer	The number of the operation.

Column	Type	Description
operation_type	char(10)	The type of operation to be performed; Insert, Update, Delete, or Other.
rowidn	integer	The ID of the row on which to apply the operation.
table	char	The full table name, trimmed to fit a reasonable length. Format: *database*:*owner*.*tablename*
sqlerrno	integer	Error number of any SQL error (or 0 on success)

sysproxytxns table

The **sysproxytxns** table contains information about all of the current transactions that are running through each proxy distributor.

On the primary server, this table contains information about each of the proxy distributors in the high-availability cluster. On a secondary server, this table only contains information about the proxy distributors used to process updates to the secondary server.

The following table provides information about the columns in the **sysproxytxns** table:

Column	Type	Description
proxy_id	integer	ID of the proxy distributor. These IDs are unique within a high-availability cluster.
proxy_txn_id	integer	Number of the transaction. These numbers are unique to the proxy distributor.
reference_count	integer	Indicates the number of threads (for example, sqlexec, sync reply, recovery, and so on) that are using the information for this transaction. When the count becomes 0 this indicates the transaction processing is complete. (either successfully or unsuccessfully).
pending_ops	integer	On the primary server: the number of operations received from the secondary server that have not yet been processed. On the secondary server, the number of operations buffered on the secondary server that have not yet been sent to the primary server.
proxy_sid	integer	Proxy Session ID

sysptnhdr

The **sysptrhdr** table contains information about partition headers.

Column	Type	Description
partnum	integer	Partnum of the table
flags	integer	Partition flags
rowsize	integer	Row size (maximum for variable)
ncols	smallint	Number of VARCHAR or BLOB columns
nkeys	smallint	Number of indexes
nextns	smallint	Number of extents
pagesize	smallint	Page size
created	integer	Date created
serialv	integer	Current SERIAL value
fextsiz	integer	First extent size, in pages
nextsiz	integer	Next extent size, in pages
nptotal	integer	Number of pages allocated
npused	integer	Number of pages used
npdata	integer	Number of data pages
octptnm	integer	Optical BLOB partnum
lockid	integer	Table lock ID
nrows	bigint	Number of data rows
ninserts	bigint	Number of insert operations
nupdates	bigint	Number of update operations
ndeletes	bigint	Number of delete operations
cur_serial8	int8	Current SERIAL8 value
cur_bigserial	bigint	Current BIGSERIAL value

Column	Type	Description
dbsnum	integer	Number of partitions in the dbspace
pta_oldvers	smallint	In-place alter
pta_newvers	smallint	In-place alter
pta_bmpagenum	integer	In-place alter
pta_totpgs	integer	In-place alter
pta_opems_allocd	integer	In-place alter
pta_opems_filled	integer	In-place alter
glscollname	char(32)	In-place alter
flags2	integer	Partition flags2
sid	integer	Temporary table session ID

You can run the following query to see the number of allocated pages for temporary tables:

```
SELECT i.sid, hex(i.flags) flags, hex(i.partnum) partition,
   trim(n.dbsname) || ":" || trim(n.owner) || ":" || trim(n.tabname) table,
   i.nptotal allocated_pages
  FROM sysmaster:systabnames n, sysmaster:sysptnhdr i
 WHERE (sysmaster:bitval(i.flags, "0x0020") = 1)
   AND i.partnum = n.partnum
```

For example, the query can return information similar to the following output:

```
session with query "select * from customer into temp good "
sid               60
flags             0x00000861
partition         0x00100249
table             demo:informix:good
allocated_pages   8

session with temp table generated from query "select from <view>"
sid               64
flags             0x00008821
partition         0x00100249
table             demo:informix:_temptable
allocated_pages   8

temp table from sorting
sid               33
flags             0x000048A0
partition         0x00200004
table             SORTTEMP:informix:th_tmprun_0x4a1b2370
allocated_pages   128

temp table from hashing
sid               31
flags             0x000048A0
partition         0x00200003
table             HASHTEMP:informix:th_overflow_0xffffffffffffffff
allocated_pages   16
```


sysptprof table

The **sysptprof** table lists information about a tblspace. Tblspaces correspond to tables.

Profile information for a table is available only when a table is open. When the last user who has a table open closes it, the tblspace in shared memory is freed, and any profile statistics are lost.

The following table provides information about the columns in the **sysptprof** table:

Column	Type	Description
dbsname	char(128)	Database name
tabname	char(128)	Table name
partnum	integer	Partition (tblspace) number
lockreqs	integer	Number of lock requests
lockwts	integer	Number of lock waits
deadlks	integer	Number of deadlocks
lktouts	integer	Number of lock timeouts
isreads	integer	Number of isreads
iswrites	integer	Number of iswrites
isrewrites	integer	Number of isrewrites
isdeletes	integer	Number of isdeletes

Column	Type	Description
bufreads	integer	Number of buffer reads
bufwrites	integer	Number of buffer writes
seqscans	integer	Number of sequential scans
pagreads	integer	Number of page reads
pagwrites	integer	Number of page writes

sysrepevtreg table

Use the **sysrepevtreg** pseudo table to register for a pre-defined set of events from the Connection Manager.

The following table provides information about the columns in the **sysrepevtreg** table:

Table 1. **sysrepevtreg** table information

Column	Type	Description
evt_bitmap	integer	Event ID bitmap
evt_timeout	integer	Maximum time in seconds that the client can wait for event data. Valid timeout values are: • 0; no wait (default) • -1; wait forever • *n* (where *n* > 0) wait *n* seconds
evt_hwm	integer	Pending event list high-water mark
evt_info	char(256)	Event information (Not yet implemented)

sysrepstats table

Use the **sysrepstats** table to post events to Connection Manager.

The following table provides information about the columns in the **sysrepstats** table:

Table 1. **sysrepstats** table information

Column	Type	Description
repstats_type	integer	Event ID
repstats_subtype	integer	Sub event ID
repstats_time	integer	Time at which event was initiated
repstats_ver	integer	Version number of event data
repstats_desc	lvarchar	Event data

- User Interface for sysrepstats and sysrepevtreg Tables
  Client applications can post events to Connection Manager or to other clients by inserting event information into the **sysrepstats** pseudo table. Client applications can register events using the sysmaster pseudo table **sysrepevtreg**, and receive event data by issuing select or fetch statements against the **sysrepstats** pseudo table.

User Interface for sysrepstats and sysrepevtreg Tables

Client applications can post events to Connection Manager or to other clients by inserting event information into the **sysrepstats** pseudo table. Client applications can register events using the sysmaster pseudo table **sysrepevtreg**, and receive event data by issuing select or fetch statements against the **sysrepstats** pseudo table.

By posting events to the **sysrepstats**, you can issue control messages to Connection Manager without having to directly connect to Connection Manager itself.

When Connection Manager registers that it wishes to receive events, it passes a bitmap of the event types that it wants to receive. As events are received, they are posted to the thread that placed the request.

Event Classes

The following table lists each event class, its bit value, and a description of the event class.

Table 1. Event Classes

Event class name	Bit value	Description
REPEVT_CLUST_CHG	0x1	Event class for High-Availability cluster changes
REPEVT_CLUST_PERFSTAT	0x2	Event class for workload statistics for the server nodes in a High-Availability cluster
REPEVT_CLUST_LATSTAT	0x4	Event class for replication latency information for server nodes in a High-Availability cluster
REPEVT_CM_ADM	0x8	Connection Manager administration commands
REPEVT_SRV_ADM	0x10	Event class for server mode changes
REPEVT_ER_ADM	0x20	Event class for events related to Enterprise Replication (ER)
REPEVT_CLIENT	0x40	User-defined client event

Sub-events for the Event Class REPEVT_CLUST_CHG

The following table lists sub-events for the event class REPEVT_CLUST_CHG:

Table 2. Sub-events for the Event Class REPEVT_CLUST_CHG

Sub-event name	Value	Description	Event available at:
REPEVT_SUB_CLUST_ADD	1	Adding new node to a High-Availability cluster	Only at primary server in a High-Availability cluster
REPEVT_SUB_CLUST_DROP	2	Dropping a node from a High-Availability cluster	Only at primary server in a High-Availability cluster
REPEVT_SUB_CLUST_CON	3	High-Availability secondary node connected to primary server	Only at primary server in a High-Availability cluster
REPEVT_SUB_CLUST_DIS	4	High-Availability secondary node disconnected from primary server	Only at primary server in a High-Availability cluster
REPEVT_SUB_CLUST_NEWPRIM	5	High-Availability primary node changed	Only at secondary servers in a High-Availability cluster
REPEVT_SUB_CLUST_DROFF	6	HDR secondary node disconnected from primary server	HDR primary and secondary servers
REPEVT_SUB_CLUST_DRON	7	HDR secondary node connected to primary server	HDR primary and secondary servers

Sub-events for the Event Class REPEVT_CLUST_PERFSTAT

The following table lists sub-events for the event class REPEVT_CLUST_PERFSTAT:

Table 3. Sub-events for the Event Class REPEVT_CLUST_PERFSTAT

Sub-event name	Value	Description	Event available at:
REPEVT_ SUB_LOCAL_PERFSTAT	1	Work load statistics for local server	All servers in a High-Availability cluster
REPEVT_ SUB_REMOTE_PERFSTAT	2	Work load statistics for High-Availability secondary servers	Only at the primary server in a High-Availability cluster

Sub-events for the Event Class REPEVT_CLUST_LATSTAT

The following table lists sub-events for the event class REPEVT_CLUST_LATSTAT:

Table 4. Sub-events for the Event Class REPEVT_CLUST_LATSTAT

Sub-event name	Value	Description	Event available at:
REPEVT_SUB_LOCAL_LATSTAT	1	Replication latency statistics for secondary servers in a High-Availability cluster	Only at the primary server in a High-Availability cluster

Sub-events for the Event Class REPEVT_CM_ADM

The following table lists sub-events for the event class REPEVT_CM_ADM:

Table 5. Sub-events for the Event Class REPEVT_CM_ADM

Sub-event name	Value	Description	Event available at:
REPEVT_SUB_CM_ADM_REQ	1	Command request	All server instances
REPEVT_SUB_CM_ADM_ACK	2	Command response	All server instances
REPEVT_SUB_CM_REG	3	Connection Manager registered with server	All server instances
REPEVT_SUB_CM_DEREG	4	Connection Manager de-registered with server	All server instances
REPEVT_SUB_CM_FATAL	5	Connection Manager terminated without de-registering with server	All server instances

Sub-events for the Event Class REPEVT_SRV_ADM

The following table lists sub-events for the event class REPEVT_SRV_ADM:

Table 6. Sub-events for the Event Class REPEVT_SRV_ADM

Sub-event name	Value	Description	Event available at:
REPEVT_ SUB_SRV_BLK	1	Server blocked due to DDRBLOCK	All server instances
REPEVT_ SUB_SRV_UBLK	2	Server unblocked; DDRBLOCK removed	All server instances

Sub-events for the Event Class REPEVT_ER_ADM

The following table lists sub-events for the event class REPEVT_ER_ADM:

Table 7. Sub-events for the Event Class REPEVT_ER_ADM

Sub-event name	Value	Description	Event available at:
REPEVT_SUB_ER_SPOOL_FULL	1	ER blocked while waiting for space to be added in either the queue data sbspace or dbspace, or in the grouper paging sbspace.	Enterprise Replication server nodes

sysrsslog

The **sysrsslog** table captures information about RS secondary servers at the primary server.

Table 1. **sysrsslog** table information

Column	Type	Description
server_name	char(128)	Server name
from_cache	integer	Total pages read from log buffer cache
from_disk	integer	Total pages read from disk
logpages_tossed	integer	Total number of log pages not written to log buffer cache

sysscblst

These columns of the **sysscblst** table provide information about session memory amounts.

Column	Type	Description
memtotal	integer	Total memory available
memused	integer	Total memory used

syssscelem

The **syssscelem** table provides information about Statement Cache Entries.

Column	Type	Description
uniqid	integer	Unique id of element
lru	integer	Index of lru queue
hash	integer	Hash value of cached entry
ref_cnt	integer	Num threads referencing the statement
hits	integer	Num times element used
flag	integer	• Invalid 0x1 • Fully Cached 0x2 • Inserting 0x4
valid	integer	Valid 1, Invalid 0
locked	integer	Locked 1, Unlocked 0
heap_ptr	bigint	Address of memory heap for cache entry
database	char(128)	Database name

Column	Type	Description
user	char(32)	User
stmtstring	char(16000)	Statement string
queryplan	char(16000)	Query plan

syssesappinfo

The **syssesappinfo** table in the **sysmaster** displays information on Distributed Relational Database Architecture™ (DRDA) client sessions. The **syssesappinfo** table has the following columns.

Table 1. **syssesappinfo** table column information

Column	Type	Explanation
sesapp_sid	INTEGER	Client session ID
sesapp_name	CHAR(128)	Session application name
sesapp_value	CHAR(512)	Session value

syssesprof

The **syssesprof** table lists cumulative counts of the number of occurrences of user actions such as writes, deletes, or commits.

Column	Type	Description
sid	integer	Session ID
lockreqs	integer	Number of locks requested
locksheld	integer	Number of locks currently held
lockwts	integer	Number of times waited for a lock
deadlks	integer	Number of deadlocks detected
lktouts	smallint	Number of deadlock time-outs
logrecs	integer	Number of logical-log records written
isreads	integer	Number of reads
iswrites	integer	Number of writes
isrewrites	integer	Number of rewrites
isdeletes	integer	Number of deletes
iscommits	integer	Number of commits
isrollbacks	integer	Number of rollbacks
longtxs	integer	Number of long transactions
bufreads	integer	Number of buffer reads
bufwrites	integer	Number of buffer writes
seqscans	integer	Number of sequential scans
pagreads	integer	Number of page reads
pagwrites	integer	Number of page writes
total_sorts	integer	Number of total sorts
dsksorts	integer	Number of sorts that did not fit in memory
max_sortdiskspace	integer	Maximum space used by a sort
logspused	integer	Number of bytes of logical-log space used by current transaction of session
maxlogsp	integer	Maximum number of bytes of logical-log usage for any single transaction since the session started

syssessions

The **syssessions** table provides general information on each user connected to the database server. In the **state** column, each bit position represents a separate flag. Thus, it might be easier to read values in the **state** column if the values are returned using the HEX function.

Table 1. **syssessions** table information

Column	Type	Description		
sid	integer	Session ID		
username	char(32)	User ID		
uid	smallint	User ID number		
pid	integer	Process ID of the client		
hostname	char(256)	Hostname of client		
tty	char(16)	Name of the user's **stderr** file		
connected	integer	Time that user connected to the database server		
feprogram	char(255)			
pooladdr	integer	Session pool address		
is_wlatch	integer	1 if the primary thread for the session is waiting for a latch		
is_wlock	integer	1 if the primary thread for the session is waiting for a lock		
is_wbuff	integer	1 if the primary thread for the session is waiting for a buffer		
is_wckpt	integer	1 if the primary thread for the session is waiting for a checkpoint		
is_wlogbuf	integer	1 if the primary thread for the session is waiting for a log buffer		
is_wtrans	integer	1 if the primary thread for the session is waiting for a transaction		
is_monitor	integer	1 if the session is a special monitoring process		
is_incrit	integer	1 if the primary thread for the session is in a critical section		
state	integer	Flags	Hexadecimal	Meaning
		1	0x00000001	User structure in use
		2	0x00000002	Waiting for a latch
		4	0x00000004	Waiting for a lock
		8	0x00000008	Waiting for a buffer
		16	0x00000010	Waiting for a checkpoint
		32	0x00000020	In a read call
		64	0x00000040	Writing logical-log file to backup tape
		256	0x00000100	In a critical section
		512	0x00000200	Special daemon
		1024	0x00000400	Archiving
		2048	0x00000800	Clean up dead processes
		4096	0x00001000	Waiting for write of log buffer
		8192	0x00002000	Special buffer-flushing thread
		16384	0x00004000	Remote database server
		32768	0x00008000	Deadlock timeout used to set RS_timeout
		65536	0x00010000	Regular lock timeout
		262144	0x00040000	Waiting for a transaction
		524288	0x00080000	Primary thread for a session
		1048576	0x00100000	Thread for building indexes
		2097152	0x00200000	B-tree cleaner thread

syssessiontempspaceusage

The **syssessiontempspaceusage** table contains information about each session's temp space usage, includes both implicit and explicit temp tables.

Column	Type	Description
sid	integer	ID of the session which allocated temp space
flags	char(10)	Partition flags of the temp space partition
partition	char(10)	Partition number of the temp space partition
table	lvarchar(290)	Table name of the temp space partition

Column	Type	Description
allocated_pages	integer	Number of pages allocated by the temp space partition

Example:

```
select * from syssessiontempspaceusage

sid               53
flags             0x00000861
partition         0x001001A4
table             stores_demo:jmcmahon:foo
allocated_pages   8
```


syssmx

The **syssmx** table provides SMX (server multiplexer group) connection information.

Table 1. **syssmx** table column information

Column	Type	Description
address	int8	SMX pipe address
name	char(128)	Target server name
encryption_status	char(20)	Enabled or disabled
buffers_sent	integer	Number of buffers sent
buffers_recv	integer	Number of buffers received
bytes_sent	int8	Number of bytes sent
bytes_recv	int8	Number of bytes received
reads	integer	Number of read calls
writes	integer	Number of write calls
retries	integer	Number of write call retries

syssmxses

The **syssmxses** table provides SMX (server multiplexer group) session information.

Table 1. **syssmxses** table column information

Column	Type	Description
name	char(128)	Target server name
address	int8	SMX session address
client_type	char(20)	SMX client type
reads	integer	Number of read calls
writes	integer	Number of write calls

syssqexplain table

The **syssqexplain** pseudo table stores information about SQL queries.

The information stored includes the plan of the query optimizer, an estimate of the number of rows returned, and the relative cost of the query.

Table 1. The syssqexplain pseudo table

Column	Type	Description
sqx_sessionid	INTEGER	The session ID associated with the SQL statement.
sqx_sdbno	INTEGER	The position of the query in the array of session IDs.
sqx_iscurrent	CHAR	Whether the query is the current SQL statement.
sqx_executions	INTEGER	The total number of executions of the query.
sqx_cumtime	FLOAT	The cumulative time to run the query. Important: If SQL tracing is disabled a zero is shown.

Column	Type	Description
sqx_bufreads	INTEGER	The number of buffer reads performed while running the query. Important: If SQL tracing is disabled a zero is shown.
sqx_pagereads	INTEGER	The number of page reads performed while running the query. Important: If SQL tracing is disabled a zero is shown.
sqx_bufwrites	INTEGER	The number of buffer writes performed while running the query. Important: If SQL tracing is disabled a zero is shown.
sqx_pagewrites	INTEGER	The number of page writes performed while running the query. Important: If SQL tracing is disabled a zero is shown.
sqx_totsorts	INTEGER	The total number of sorts performed while running the query. Important: If SQL tracing is disabled a zero is shown.
sqx_dsksorts	INTEGER	The number of disk sorts performed while running the query. Important: If SQL tracing is disabled a zero is shown.
sqx_sortspmax	INTEGER	The maximum disk space required by a sort.
sqx_conbno	SMALLINT	The position in the conblock list.
sqx_ismain	CHAR	Whether the query is in the main block for the statement.
sqx_selflag	VARCHAR(200,0)	The type of SQL statement, for example: SELECT, UPDATE, DELETE.
sqx_estcost	INTEGER	The estimated cost of the query.
sqx_estrows	INTEGER	The estimated number of rows returned by the query.
sqx_seqscan	SMALLINT	The number of sequential scans used by the query.
sqx_srtscan	SMALLINT	The number of sort scans used by the query.
sqx_autoindex	SMALLINT	The number of autoindex scans used by the query.
sqx_index	SMALLINT	The number of index paths used by the query.
sqx_remsql	SMALLINT	The number of remote paths used by the query.
sqx_mrgjoin	SMALLINT	The number of sort-merge joins used by the query.
sqx_dynhashjoin	SMALLINT	The number of dynamic hash joins used by the query.
sqx_keyonly	SMALLINT	The number of key-only scans used by the query.
sqx_tempfile	SMALLINT	The number of temporary files used by the query.
sqx_tempview	SMALLINT	The number of temporary tables for views created by the query.
sqx_secthreads	SMALLINT	The number of secondary threads used by the query.
sqx_sqlstatement	CHAR	The SQL query that was run.

syssqltrace

The **syssqltrace** table provides detailed information about a single SQL statement.

Column	Type	Description
sql_id	int8	Unique SQL execution ID
sql_address	int8	Address of the statement in the code block
sql_sid	int	Database session ID of the user running the SQL statement
sql_uid	int	User ID of the statement running the SQL
sql_stmttype	int	Statement type
sql_stmtname	char(40)	Statement type displayed as a word
sql_finishtime	int	Time this statement completed (UNIX)
sql_begintxtime	int	Time this transaction started
sql_runtime	float	Statement execution time
sql_pgreads	int	Number of disk reads for this SQL statement
sql_bfreads	int	Number of buffer reads for this SQL statement
sql_rdcache	float	Percentage of time the page was read from the buffer pool
sql_bfidxreads	int	Number of index page buffer reads
sql_pgwrites	int	Number of pages written to disk
sql_bfwrites	int	Number of pages modified and returned to the buffer pool
sql_wrcache	float	Percentage of time a page was written to the buffer pool but not to disk

Column	Type	Description
sql_lockreq	int	Total number of locks required by this SQL statement
sql_lockwaits	int	Number of times the SQL statement waited on locks
sql_lockwttime	float	Time the system waited for locks during SQL statement
sql_logspace	int	Amount of space the SQL statement used in the logical log
sql_sorttotal	int	Number of sorts that ran for the statement
sql_sortdisk	int	Number of sorts that ran on disk
sql_sortmem	int	Number of sorts that ran in memory
sql_executions	int	Number of times the SQL statement ran
sql_totaltime	float	Total amount of time spent running the statement
sql_avgtime	float	Average amount of time spent running the statement
sql_maxtime	float	Maximum amount of time spent executing the SQL statement
sql_numiowaits	int	Number of times an I/O operation had to wait
sql_avgiowaits	float	Average amount of time that the SQL statement had to wait
sql_totaliowaits	float	Total amount of time that the SQL statement had to wait for I/O. This excludes any asynchronous I/O.
sql_rowspersec	float	Average number of rows (per second) produced
sql_estcost	int	Cost associated with the SQL statement
sql_estrows	int	Estimated number of rows returned for the SQL statement as predicted by the optimizer
sql_actualrows	int	Number of rows returned for the SQL statement
sql_sqlerror	int	SQL error number
sql_isamerror	int	RSAM/ISAM error number
sql_isollevel	int	Isolation level of the SQL statement.
sql_sqlmemory	int	Number of bytes needed to execute the SQL statement
sql_numiterators	int	Number of iterators used by the statement
sql_database	char(128)	Database name
sql_numtables	int	Number of tables used in executing the SQL statement
sql_tablelist	char(4096)	List of table names directly referenced in the SQL statement. If the SQL statement fires triggers that execute statements against other tables, the other tables are not listed.
sql_statement	char(1600)	SQL statement that ran

syssqltrace_hvar

The **syssqltrace_hvar** table describes information about the SQL tracing host variable.

Column	Type	Description
sql_id	int8	SQL execution ID
sql_address	int8	Address of the SQL statement block
sql_hvar_id	int	ID of the SQL host variable
sql_hvar_flags	int	Flags for the host variable
sql_hvar_typeid	int	Type ID of the host variable
sql_hvar_xtypeid	int	xtype ID of the host variable
sql_hvar_ind	int	Index of the host variable
sql_hvar_type	char(128)	Type of host variable
sql_hvar_data	char(8192)	Value of host variable

syssqltrace_info

The **syssqltrace_info** table describes information about the SQL profile trace system.

Column	Type	Description
flags	integer	SQL trace flags

Column	Type	Description
ntraces	integer	Number of items to trace
tracesize	integer	Size of the text to store for each SQL trace item
duration	integer	Trace buffer (in seconds)
sqlseen	int8	Number of SQL items traced since start or resizing
starttime	integer	Time tracing was enabled
memoryused	int8	Number of bytes of memory used by SQL tracing

syssqltrace_iter

The **syssqltrace_iter** table lists the SQL statement iterators.

Column	Type	Description
sql_id	int8	SQL execution ID
sql_address	int8	Address of the SQL statement block
sql_itr_address	int8	Address of the iterator
sql_itr_id	int	Iterator ID
sql_itr_left	int	Iterator ID to the left
sql_itr_right	int	Iterator ID to the right
sql_itr_cost	int	Iterator cost
sql_itr_estrows	int	Iterator estimated rows
sql_itr_numrows	int	Iterator actual rows processed
sql_itr_type	int	Iterator type
sql_itr_misc	int	Iterator miscellaneous flags
sql_it_info	char(256)	Iterator miscellaneous flags displayed as text

syssrcrss

The **syssrcrss** table provides RS secondary server related statistics at the primary server.

Table 1. **syssrcrss** table column information

Column	Type	Description
address	int8	RS secondary server control block address
server_name	char(128)	Database server name
server_status	char(20)	Quiescent, active, or inactive
connection_status	char(20)	Connected or disconnected
log_transmission_status	char(20)	Active or blocked
next_page_tosend_log_uniq	integer	Unique log ID of next page to send
next_page_tosend_log_page	integer	Page number of next page to send
seq_tosend	integer	Sequence ID of last buffer sent
last_seq_acked	integer	Sequence ID of last buffer acknowledged

syssrcsds

The **syssrcsds** table provides SD secondary server related statistics at the primary server.

The **syssrcsds** table contains the columns that are shown in the following table.

Column	Type	Description
address	int8	SD secondary server control block address
source_server	char(128)	Primary database server name

Column	Type	Description
connection_status	char(20)	Connected or disconnected
last_received_log_uniq	integer	Unique log ID of last log page received
last_received_log_page	integer	Page number of last log page received
next_lpgtoread_log_uniq	integer	Unique log ID of next log page to read
next_lpgtoread_log_page	integer	Page number of next log page to read
last_acked_lsn_uniq	integer	Unique log ID of last LSN acknowledged
last_acked_lsn_pos	integer	Log position of last LSN acknowledged
last_seq_received	integer	Sequence ID of last buffer received
last_seq_acked	integer	Sequence ID of last buffer acknowledged
cur_pagingfile	char(640)	Current® paging file name
cur_pagingfile_size	int8	Current paging file size
old_pagingfile	char(640)	Old paging file name
old_pagingfile_size	int8	Old paging file size

systabnames

The **systabnames** table describes each table that the database server manages.

Column	Type	Description
partnum	integer	tblspace identifier
dbsname	char(128)	Database name
owner	char(32)	User ID of owner
tabname	char(128)	Table name
collate	char(32)	Collation associated with a database that supports GLS

systhreads

The **systhreads** table provides information about each thread.

Column	Type	Description
th_id	INTEGER	The numeric identifier of the thread.
th_addr	INTEGER	The memory address of the thread control block.
th_joinlist	INTEGER	If a list of the threads are waiting for this thread to exit, the th_joinlist column shows the address of the first thread in the list.
th_joinnext	INTEGER	If a list of the threads are waiting for this thread to exit, the th_joinnext column shows the address of the next thread in the join list.
th_joinee	INTEGER	The address of the thread whose exit this thread is waiting for.
th_name	CHAR(12)	The name of the thread.
th_state	INTEGER	The status code of the thread.
th_priority	INTEGER	The priority of the thread.
th_class	INTEGER	The code for the class of virtual processor that thread will run on.
th_vpid	INTEGER	The ID of the virtual processor that the thread was last scheduled to run on.
th_mtxwait	INTEGER	The address of the mutex that this thread is waiting for.
th_conwait	INTEGER	The address of the condition that this thread is waiting for.
th_waketime	INTEGER	The time of the expiration of the last sleep. The time is calculated by an internal clock. A value of -1 means that the time value is indeterminate.
th_startwait	INTEGER	The time when the last wait began. The time is calculated by an internal clock.
th_startrun	INTEGER	The time when the last execution began. The time is calculated by an internal clock.

systrgrss

The **systrgrss** table provides RS secondary server related statistics at the RS secondary server.

Column	Type	Description
address	int8	RS secondary server control block address
source_server	char(128)	Source server serving the RS secondary server
connection_status	char(20)	Connected or disconnected
last_received_log_uniq	integer	Unique log ID of last log page received
last_received_log_page	integer	Page number of last log page received
last_seq_received	integer	Sequence ID of last buffer received
last_seq_acked	integer	Sequence ID of last buffer acknowledged

systrgsds

The **systrgsds** table provides SD secondary server related statistics at the SD secondary server.

The **systrgsds** table contains these columns:

Column	Type	Description
address	int8	SD secondary server control block address
source_server	char(128)	Source server serving the SD secondary server
connection_status	char(20)	Connected or disconnected
last_received_log_uniq	integer	Unique log ID of last log page received
last_received_log_page	integer	Page number of last log page received
next_lptoread_log_uniq	integer	Unique log ID of next log page to read
next_lptoread_log_page	integer	Page number of next log page to read
last_acked_lsn_uniq	integer	Unique log ID of last LSN acknowledged
last_acked_lsn_pos	integer	Log position of last LSN acknowledged
last_seq_received	integer	Sequence ID of last buffer received
last_seq_acked	integer	Sequence ID of last buffer acknowledged
cur_pagingfile	char(640)	Current® paging file name
cur_pagingfile_size	int8	Current paging file size
old_pagingfile	char(640)	Old paging file name
old_pagingfile_size	int8	Old paging file size

sysvpprof

The **sysvpprof** table lists user and system CPU time for each virtual processor.

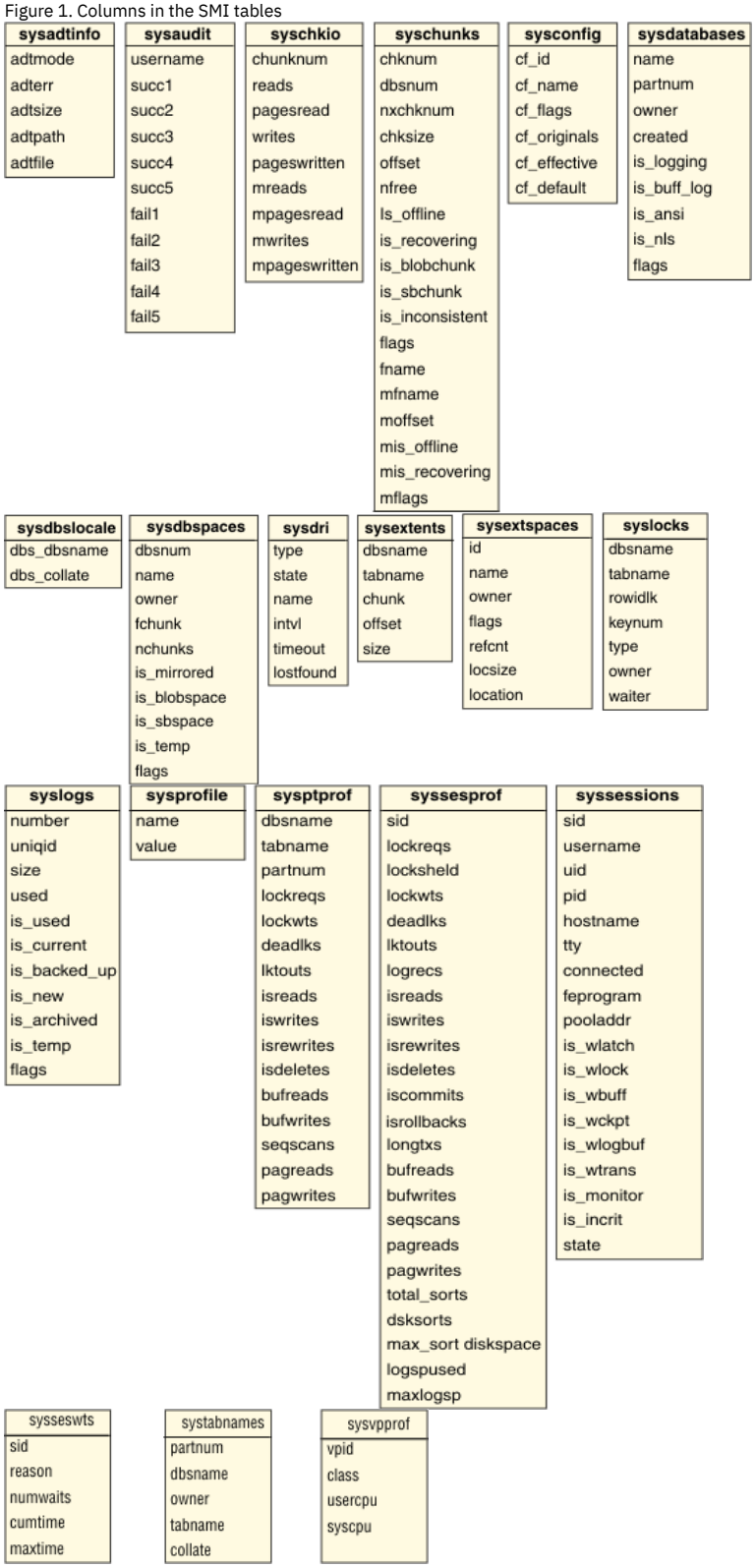
Column	Type	Description
vpid	integer	Virtual processor ID
	char(50)	Type of virtual processor: • cpu • adm • lio • pio • aio • tli • soc • str • shm • opt • msc • adt
usercpu	float	Number of microseconds of user time

Column	Type	Description
syscpu	float	Number of microseconds of system time

The SMI Tables Map

Figure 1 displays the columns in some of the SMI tables.

Figure 1. Columns in the SMI tables



Information from onstat in the SMI Tables

To obtain information provided by the **onstat** utility, you can use SQL to query appropriate SMI tables. The following table indicates which SMI tables to query to obtain the information provided by a given **onstat** option. For descriptions of the **onstat** options, see Monitor the database server status.

onstat Option	SMI Tables to Query	onstat Fields *Not* in SMI Tables
-d	sysdbspaces syschunks	address bpages
-D	sysdbspaces syschkio	
-F	sysprofile	address flusher snoozer state data
-g ath	systhreads	
-g dri	sysdri	Last DR CKPT (id/pg)
-g glo	sysvpprof	Listing of virtual processors by
-g ipl	sysipl	
-g rss	sysrsslog systrgrss syssrcrss	
-g his	syssqltracing	
-g sds	syssrcsds systrgsds	
-g smx	syssmx	
-g smx ses	syssmxses	
-k	syslocks	address lklist tblsnum
-l	syslogs sysprofile	All physical-log fields (except numpages and numwrits) All logical-log buffer fields (except numrecs, numpages, and numwrits) address begin % used
-p	sysprofile	
-u	syssessions syssesprof	address wait nreads nwrites

The sysadmin Database

The **sysadmin** database contains the tables that contain and organize the Scheduler tasks and sensors, store data collected by sensors, and record the results of Scheduler jobs and SQL administration API functions.

By default, only user **informix** is granted access to the **sysadmin** database; other users can be granted access to the **sysadmin** database.

Do not drop or alter the **sysadmin** database because it is used by several important database server components. You can, however, move the **sysadmin** database from its default root dbspace location if the root dbspace does not have enough space for storing task properties and command history information. To move the **sysadmin** database, use the SQL administration API **admin()** or **task()** function with the reset sysadmin argument.

Important: Moving the **sysadmin** database resets the database back to the original state when it was first created; all data, command history, and results tables are lost. Only built-in tasks, sensor, and thresholds remain in the **sysadmin** tables.

- The Scheduler tables
  The Scheduler stores information about tasks and sensors in five tables in the **sysadmin** database: **ph_task**, **ph_run**, **ph_group**, **ph_alert**, and **ph_threshold**.
- The results tables
  The results tables contain historical data about sensors that are run by the Scheduler.
- The command_history table
  The **command_history** table contains the list and results of all the SQL administration API functions that were run in the previous 30 days.
- The storagepool table
  The **storagepool** table in **sysadmin** database contains information about all of the entries in the storage pool in an Informix® instance. Each entry represents free space that the server can use when automatically expanding a storage space.
- The tenant table
  The **tenant** table in the **sysadmin** database contains information about the tenant databases.

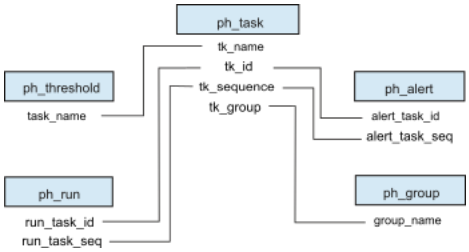
Related reference:
reset sysadmin argument: Move the sysadmin database (SQL administration API)

The Scheduler tables

The Scheduler stores information about tasks and sensors in five tables in the **sysadmin** database: **ph_task**, **ph_run**, **ph_group**, **ph_alert**, and **ph_threshold**.

The Scheduler is an administrative tool that enables the database server to execute database functions and procedures at predefined times or as determined internally by the server. The five tables used by the Scheduler contain built-in tasks and sensors that run automatically. You can also add your own tasks and sensors by inserting rows into these tables. These tables have relationships between their columns that are described in the following illustration.

Figure 1. Relationships between the Scheduler tables



For detailed information about using the Scheduler, see the *IBM® Informix® Administrator's Guide*.

- The ph_task Table
  The **ph_task** table contains information about Scheduler tasks and sensors. The **ph_task** table contains built-in tasks and sensors that are scheduled to run automatically.
- The ph_run Table
  The **ph_run** table contains information about how and when each Scheduler task or sensor ran.
- The ph_group Table
  The **ph_group** table contains information about the Scheduler group names. The **ph_group** table contains several groups that are used to categorize built-in tasks and sensors, as well as the default group MISC.
- The ph_alert Table
  The **ph_alert** table contains information about event alarms generated by the database server or alerts generated by the Scheduler. Alerts that are associated with built-in tasks and sensors are automatically added to the **ph_alert** table.
- The ph_threshold Table
  The **ph_threshold** table contains information about thresholds for Scheduler tasks.

Related information:
Scheduler tables

The ph_task Table

The **ph_task** table contains information about Scheduler tasks and sensors. The **ph_task** table contains built-in tasks and sensors that are scheduled to run automatically.

Table 1. The ph_task table

Column	Type	Description
tk_id	serial	Sequential job ID. System updated; do not modify. Referenced in the **alert_task_id** column in the **ph_alert** table and in the **run_task_id** column in the **ph_run** table.
tk_name	char(36)	Job name. A unique index on this column ensures that no two names are the same. Referenced in the **task_name** column of the **ph_threshold** table.
tk_description	lvarchar	Description about what the task or sensor does.
tk_type	char(18)	Type of job: • TASK: Invokes an action at a specific time and frequency • SENSOR: (Default) A task that collects, stores, and purges data to or from a result table • STARTUP TASK: A task that runs only when the server starts • STARTUP SENSOR: A sensor that runs only when the server starts
tk_sequence	integer	Current® data collection number. System updated; do not modify. Referenced in the **alert_task_id** column of the **ph_alert** table and the **run_task_seq** column of the **ph_run** table.
tk_result_table	varchar	Results table name for storing data collected by a sensor. The table is created by the CREATE TABLE statement in the **tk_create** column.
tk_create	lvarchar	The CREATE TABLE statement used to create the results table to store data collected by a sensor. One of the columns in the table must be named ID and hold the **tk_sequence** value. This value indicates the age of the row and can be used for purging the row.
tk_dbs	varchar(250)	The database in which the task is run. Default is `sysadmin`.
tk_execute	lvarchar	The SQL statement to execute. The length of the command is limited to 2048 bytes.
tk_delete	interval day(2) to second	Data older than this interval is deleted from the result table. Default is `1:00:00` (one day).
tk_start_time	datetime hour to second	Time when the task or sensor starts. Default is `08:00:00`.

Column	Type	Description
tk_stop_time	datetime hour to second	Time of day after which the task or sensor cannot be scheduled to be run. The database server schedules the next execution on the next valid day. Default is `19:00:00`. Can be `NULL`, indicating no stop time.
tk_frequency	interval day(2) to second	How often this task or sensor runs. Default is `1` (once a day).
tk_next_execution	datetime year to second	Next time this task or sensor will be run. After a startup task or sensor has run, this value is NULL. When a task or a sensor is enabled, the database server calculates this time from the values of **tk_start_time**, **tk_stop_time**, and **tk_frequency** columns, and the days of the week the task or sensor is enabled, according to the values of **tk_monday**, **tk_tuesday**, **tk_wednesday**, **tk_thursday**, **tk_friday**, **tk_saturday**, **tk_sunday** columns. For example, *`new_next_execution_time = current_next_execution_time + tk_frequency`*, where the *new_next_execution_time* is greater than the *current_next_execution_time*. If **tk_frequency** is not present, the task is run once.
tk_total_executions	integer	The number of times that the task or sensor was run. System updated; do not modify. Default is `0`.
tk_total_time	float	Total time spent executing this task or sensor. System updated; do not modify Default is `0.0` seconds.
tk_monday	boolean	Whether the task or sensor is run on Monday. Default is `T` (true).
tk_tuesday	boolean	Whether the task or sensor is run on Tuesday. Default is `T` (true).
tk_wednesday	boolean	Whether the task or sensor is run on Wednesday. Default is `T` (true).
tk_thursday	boolean	Whether the task or sensor is run on Thursday. Default is `T` (true).
tk_friday	boolean	Whether the task or sensor is run on Friday. Default is `T` (true).
tk_saturday	boolean	Whether the task or sensor is run on Saturday. Default is `T` (true).
tk_sunday	boolean	Whether the task or sensor is run on Sunday. Default is `T` (true).
tk_attributes	integer	Flags System updated; do not modify.
tk_group	varchar(128)	Group name. Must be the same as a value in the **group_name** column in the **ph_group** table. Default is `MISC`.
tk_enable	boolean	Whether the task or sensor is enabled. Default is `T` (the task is enabled).
tk_priority	integer	Job priority, on a scale of 0- 5, with higher numbers indicating higher priority. If there are several jobs to execute simultaneously, the job with the highest priority executes first. Default is `0` (low priority).

The ph_run Table

The **ph_run** table contains information about how and when each Scheduler task or sensor ran.

Table 1. The ph_run table

Column	Type	Description
run_id	serial	Sequential ID generated during execution. System updated; do not modify.
run_task_id	integer	The job ID. Referenced from the **tk_id** column of the **ph_task** table.
run_task_seq	integer	Unique sequence number of the task or sensor. Referenced from the **tk_sequence** column of the **ph_task** table.
run_retcode	integer	The return code from a stored procedure function or a user-defined routine, or the SQLCode from SQL statements.
run_time	datetime year to second	When this task or sensor was run.

Column	Type	Description
run_duration	float	The time that it took to run this task or sensor (in seconds).
run_ztime	integer	The time when the server statistics (the **onstat -z** command) were last run.
run_btime	integer	The time when the server was started.
run_mttime	integer	Internal counter of the server.

The ph_group Table

The **ph_group** table contains information about the Scheduler group names. The **ph_group** table contains several groups that are used to categorize built-in tasks and sensors, as well as the default group MISC.

Table 1. The ph_group table

Column	Type	Description
group_id	serial	Group ID. System updated; do not modify.
group_name	varchar(128)	Unique name of the group. Referenced in the **tk_group** column of the **ph_task** table.
group_description	lvarchar	Description of the group.

The ph_alert Table

The **ph_alert** table contains information about event alarms generated by the database server or alerts generated by the Scheduler. Alerts that are associated with built-in tasks and sensors are automatically added to the **ph_alert** table.

Table 1. The ph_alert table

Column	Type	Description
id	serial	The alert ID. System generated; do not modify.
alert_task_id	serial	The task or sensor ID. Must be the same as a value in the **tk_id** column in the **ph_task** table. The task ID for event alarms is 15.
alert_task_seq	integer	Identifies which invocation of a task created the alert. System generated; do not modify. Referenced from the **tk_sequence** column in the **ph_task** table.
alert_type	char(8)	The type of alert: • INFO (Default) • WARNING • ERROR The severity of an alert or event alarm is indicated by the combination of the alert type and the alert color. See Table 2.
alert_color	char(15)	The color of the alert: • GREEN (Default) • YELLOW • RED The severity of an alert or event alarm is indicated by the combination of the alert type and the alert color. See Table 2.
alert_time	datetime year to second	The time when the alert was generated. System updated; do not modify.

Column	Type	Description
alert_state	char(15)	Indicates which state the object is in: NEW (Default) The alert was added and no other action has occurred on this alert. IGNORED The alert was acknowledged by the DBA and no action was taken. ACKNOWLEDGED The alert was acknowledged by the DBA. ADDRESSED The alert was addressed by the DBA.
alert_state_changed	datetime year to second	The last time that the state was changed. System updated; do not modify.
alert_object_type	char(15)	The type of object that the alert is for: • ALARM • CHUNK • DATABASE • DBSPACE • INDEX • MISC (Default) • SERVER • SQL_STATEMENT • TABLE • USER
alert_object_name	varchar(255)	The name of the object that the alert is for or the event alarm class ID.
alert_message	lvarchar	The detailed message describing the alert or event alarm.
alert_action_dbs	lvarchar(256)	The name of the database to use for the corrective action. Default is **sysadmin**.
alert_action	lvarchar	The corrective action. An SQL script to invoke to correct the problem. This script must comply with all multi-statement prepare rules. Can be `NULL` if no action is available.
alert_object_info	bigint	For alerts of type ALARM, the event ID of the event alarm.

The following table defines the alert colors for the three types of messages and event alarms.

Table 2. Alert types and colors

Message Type	Green	Yellow	Red
Informative	A status message indicating a component's operation status. An event alarm of severity 1 (not noteworthy).	An important status message. An event alarm of severity 2 (information).	A status message that requires action.
Warning	A warning from the database that was automatically addressed.	A future event that needs to be addressed. An event alarm of severity 3 (attention).	A predicted failure is imminent. Immediate action is required.
Error	A failure in a component corrected itself.	A failure in a component corrected itself but might need DBA action.	A failure in a component requires DBA action. An event alarm of severity 4 (emergency) or 5 (fatal).

The **ph_alerts** view shows alert information and associated task or sensor information.

Related concepts:
Events in the ph_alert Table

The ph_threshold Table

The **ph_threshold** table contains information about thresholds for Scheduler tasks.

The **ph_threshold** table contains built-in thresholds that are associated with built-in tasks and sensors. For example, a threshold named COMMAND HISTORY RETENTION determines the length of time rows should remain in the **command_history** table.

Table 1. The ph_threshold table

Column	Type	Description
id	integer	Threshold ID.
name	char	The name of the threshold.

Column	Type	Description
task_name	varchar	Scheduler task name associated with the threshold. Must be the same as a value in the **tk_name** column in the **ph_task** table.
value	lvarchar	The value of the threshold.
value_type	char	The data type of the value column: • STRING • NUMERIC • NUMERIC(p,s)
description	lvarchar	A long description of the threshold.

The results tables

The results tables contain historical data about sensors that are run by the Scheduler.

Most sensors create a new table to store their results. The name of the table is listed in the **tk_result_table** column in the **ph_task** table. The structure of the table is defined by the CREATE TABLE statement in the **tk_create** column of the **ph_task** table.

The built-in sensors automatically create results tables when they run that start with the prefix **mon_**.

Table 1. Results tables

Column	Type	Description
ID	integer	The iteration sequence number of the sensor. Must be set to $DATA_SEQ_ID. Referenced from the **run_task_seq** column of the **ph_run** table.
user columns	any	You can specify any types of columns to hold the information returned by a sensor.

The command_history table

The **command_history** table contains the list and results of all the SQL administration API functions that were run in the previous 30 days.

The **command_history** table shows each SQL administration API function that was run and displays information about the user who ran the function, the time the function was run, the primary arguments of the function, and the message returned when the database server completed running the function.

Table 1. The command_history table

Column	Data Type	Description
cmd_number	serial	The unique ID for each row.
cmd_exec_time	datetime year-to-second	The time that the function started.
cmd_user	varchar	The user who ran the function.
cmd_hostname	varchar	The name of the host computer from which the function was run.
cmd_executed	varchar	The primary argument of the function that was run.
cmd_ret_status	integer	Return code.
cmd_ret_msg	lvarchar	Return message.

The following table shows sample arguments and the associated results messages in a **command_history** table.

Table 2. Example information in a command_history table

Argument (cmd_executed)	Message Returned (cmd_ret_msg)
set sql tracing on	SQL tracing on with 1000 buffers of 2024 bytes.
create dbspace	Space 'space12' added.
checkpoint	Checkpoint completed.
add log	Added 3 logical logs to dbspace logdbs.

To display the command history, run the following SQL statement from the **sysadmin** database:

```
SELECT * FROM command_history;
```

The size of the command_history table

Depending on how many SQL administration API functions are run, the **command_history** table can grow quite large. You can change the amount of time that information is retained in the **command_history** table by updating the **value** field of the COMMAND HISTORY RETENTION row in the **ph_threshold** table.

You can also use SQL statements like DELETE or TRUNCATE TABLE to manually remove the data from this table.

Related information:
Viewing SQL administration API history

The storagepool table

The **storagepool** table in **sysadmin** database contains information about all of the entries in the storage pool in an Informix® instance. Each entry represents free space that the server can use when automatically expanding a storage space.

Table 1. The **storagepool** table

Column	Type	Description
entry_id	SERIAL	The ID of the storage pool entry.
path	VARCHAR (255)	The path for the file, directory, or device that the server can use when additional storage space is required.
beg_offset	BIGINT	The initial offset in kilobytes into the device at which the server can begin allocating space. If the storage pool information is for a directory, the end offset value is `0`.
end_offset	BIGINT	The initial offset in kilobytes into the device at which the server must stop allocating space. If the storage pool information is for a directory, the offset value is `0`.
chunk_size	BIGINT	The initial size of a chunk allocated from this entry.
status	VARCHAR (255)	The status of the storage pool entry. Status values are: • Active = A functional storage pool entry. The server can allocate chunks from this entry. • Full = There is no more free space in the storage pool entry. The server cannot allocate any more chunks from this entry. • Error = The storage pool entry generated an error when the server tried to allocate a chunk from the entry.
priority	INT	The priority of the directory, file, or device when the server searches through the storage pool for space. The server allocates space from a high-priority entry before it allocates space from a lower priority entry. • 1 = High priority • 2 = Medium priority • 3 = Low priority
last_alloc	DATETIME (year to second)	The date and time of the last allocation from this entry.
logid	INT	The ID of the log that was current at the time this entry was last used. The server uses this flag with the `logused` value when choosing between entries of identical priorities.
logused	INT	The position within the log that was current at the time this entry was last used. The server uses this flag with the `logid` value when choosing between entries of identical priorities.

The tenant table

The **tenant** table in the **sysadmin** database contains information about the tenant databases.

Table 1. The **tenant** table

Column	Type	Description
tenant_id	int	The unique ID of the tenant database.
tenant_dbsname	varchar(128)	The name of the tenant database.
tenant_resources	bson	The properties of the tenant database and the state of the tenant. Cast this column to JSON to view the information.
tenant_last_updated	datetime year to second	The time stamp of the last configuration change to the tenant database.
tenant_comment	lvarchar(2048)	Comments about the tenant database.

Disk Structures and Storage

In This Chapter

The database server achieves its high performance by managing its own I/O. The database server manages storage, search, and retrieval. As the database server stores data, it creates the structures it needs to search for and retrieve the data later. The database server disk structures also store and track control information needed to manage logging and backups. Database server structures contain all the information needed to ensure data consistency, both physical and logical.

Before you read this chapter, familiarize yourself with the disk-space terms and definitions in the chapter on where data is stored in the *IBM® Informix® Administrator's Guide*.

This chapter discusses the following topics related to disk data structures:

- Dbspace structure and storage
- Storage of simple large objects
- Sbspace structure
- Time stamps
- Database and table creation: what happens on disk

- Dbspace Structure and Storage
- Storage of Simple Large Objects
- Sbspace Structure
  An sbspace is similar to a blobspace except that it holds smart large objects.
- Time Stamps
- Database and Table Creation: What Happens on Disk

Dbspace Structure and Storage

This section explores the disk structures and storage techniques that the database server uses to store data in a dbspace.

- Structure of the Root Dbspace
  As part of disk-space initialization, the database server initializes specific structures in the initial chunk of the root dbspace.
- Reserved Pages
- Structure of a Regular Dbspace
- Structure of the Chunk Free-List Page
- Structure of the Tblspace Tblspace
- Structure of the Database Tblspace
- Structure and Allocation of an Extent
- Structure and Storage of a Dbspace Page
- Structure of Fragmented Tables
- Structure of B-Tree Index Pages
- Structure of R-Tree Index Pages

Related information:
Forest of trees indexes

Structure of the Root Dbspace

As part of disk-space initialization, the database server initializes specific structures in the initial chunk of the root dbspace.

The following structures are initialized:

- Twelve reserved pages
- The first chunk free-list page
- The tblspace tblspace
- The physical log
- The logical-log files
- The database tblspace

The ROOTNAME, ROOTOFFSET, ROOTPATH, and ROOTSIZE configuration parameters specify the size and location of the initial chunk of the root dbspace. If the root dbspace is mirrored, the MIRROROFFSET and MIRRORPATH configuration parameters specify the mirror-chunk location. For more information about these parameters, see Database configuration parameters.

To see the structure of the root chunk use the **oncheck -pe** command. For more information, see oncheck -ce, -pe: Check the chunk-free list.

Reserved Pages

The first 12 pages of the initial chunk of the root dbspace are reserved pages. Each reserved page contains specific control and tracking information used by the database server.

To obtain a listing of the contents of your reserved pages, execute the command **oncheck -pr**. To also list information about the physical-log and logical-log pages, including the active physical-log pages, run the **oncheck -pR** command.

The following example shows sample **oncheck -pr** output for interval checkpoints:

```
        Time of checkpoint      10/25/2005 17:05:20
        Checkpoint Interval               1234
```

The database server also stores current configuration information in a reserved page called PAGE_CONFIG. If you change the configuration parameters from the command line and run the **oncheck -pr** command without shutting down and restarting the database server, the configuration values in the command output do not match the current values in the reserved pages. The **oncheck** utility returns a warning message.

The following example shows sample output of the contents of a PAGE_CONFIG reserved page.

```
...

Validating Informix database server reserved pages - PAGE_CONFIG
    ROOTNAME                        rootdbs
    ROOTPATH                        /home/dyn_srv/root_chunk
    ROOTOFFSET                      4
    ROOTSIZE                        8000
    MIRROR                          0
    MIRRORPATH
    MIRROROFFSET                    0
    PHYSFILE                        1000
    LOGFILES                        5
    LOGSIZE                         500
    MSGPATH                         /home/dyn_srv/online.log
    CONSOLE                             /dev/ttyp5
...                                 ...
```

Related reference:
oncheck -pr and pR: Display reserved-page information

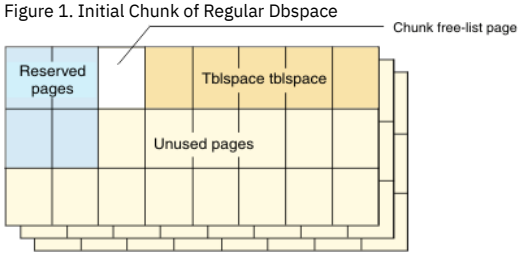
Structure of a Regular Dbspace

After disk-space initialization, you can add new dbspaces. When you create a dbspace, you assign at least one chunk (either raw or cooked disk space) to the dbspace. This chunk is referred to as the initial chunk of the dbspace. Figure 1 illustrates the structure of the initial chunk of a regular (nonroot) dbspace.

When the dbspace is first created, it contains the following structures:

- Two reserved pages
- The first chunk free-list page in the chunk
- The tblspace **tblspace** for this dbspace
- Unused pages

Figure 1. Initial Chunk of Regular Dbspace



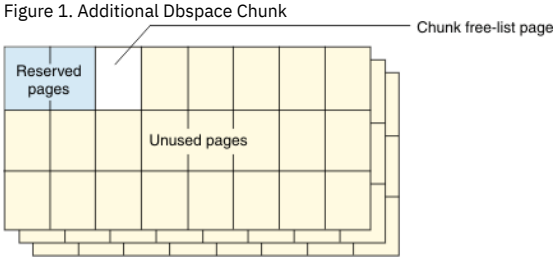
- Structure of an Additional Dbspace Chunk
- Structure of a Mirror Chunk

Structure of an Additional Dbspace Chunk

You can create a dbspace that contains more than one chunk. The initial chunk in a dbspace contains the tblspace **tblspace** for the dbspace. Additional chunks do not. When an additional chunk is first created, it contains the following structures:

- Two reserved pages
- The first chunk free-list page
- Unused pages

Figure 1 illustrates the structure of all additional chunks in a dbspace. (The structure also applies to additional chunks in the root dbspace.)
Figure 1. Additional Dbspace Chunk

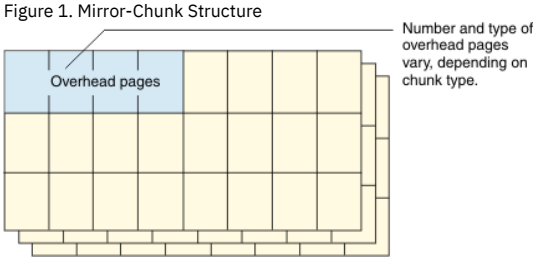


Structure of a Mirror Chunk

Each mirror chunk must be the same size as its primary chunk. When a mirror chunk is created, the database server writes the contents of the primary chunk to the mirror chunk immediately.

The mirror chunk contains the same control structures as the primary chunk. Mirrors of blobspace, sbspace, or dbspace chunks contain the same physical contents as their primary counterpart after the database server brings them online.

Figure 1 illustrates the mirror-chunk structure as it appears after the chunk is created.
Figure 1. Mirror-Chunk Structure



The mirror-chunk structure always shows no free space because all of its space is reserved for mirroring. For more information, see the chapter on what is mirroring in the *IBM® Informix® Administrator's Guide*.

Structure of the Chunk Free-List Page

In every chunk, the page that follows the last reserved page is the first of one or more chunk free-list pages that tracks available space in the chunk. For a non-root chunk, the initial length of the free space is equal to the size of the chunk minus three pages. If an additional chunk free-list page is needed to accommodate new entries, a new chunk free-list page is created in one of the free pages in the chunk. Figure 1 illustrates the location of the free-list page.

Use **oncheck -pe** to obtain the physical layout of pages in the chunk. For more information, see oncheck -ce, -pe: Check the chunk-free list.
Figure 1. Free-List Page

Structure of the Tblspace Tblspace

Each dbspace contains a tblspace called the *tblspace* ***tblspace*** that describes all tblspaces in the dbspace. When the database server creates a tblspace, it places an entry in the tblspace **tblspace** that describes the characteristics of the newly created tblspace. You cannot drop or move a chunk containing a tblspace **tblspace.**

A dbspace can have a maximum number of 2**20 tblspaces.

The default size of the first and next extents depends on whether the dbspace is the root dbspace or not, as shown in the following table.

Table 1. Default sizes for each extent and type of dbspace

Type of dbspace	Default Size of First Extent	Default Size of Next Extents
Root	• 500 KB for a 2 kilobyte page system • 1000 KB for a 4 kilobyte page system	• 100 KB for a 2 kilobyte page system • 200 KB for a 4 kilobyte page system
Non-root	• 100 KB for a 2 kilobyte page system • 200 KB for a 4 kilobyte page system	• 100 KB for a 2 kilobyte page system • 200 KB for a 4 kilobyte page system

You can specify a non-default size for the first and next extents for a tblspace **tblspace** in the following ways:

- For the root dbspace, set the TBLTBLFIRST and TBLTBLNEXT configuration parameters.
- For non-root dbspaces, use the **onspaces** utility **-ef** and **-en** options when you create a dbspace.

Tblspace tblspace entries

The tblspace **tblspace** describes the characteristics of tblspaces.

To display information on a tblspace, use the **oncheck -pt** command.

Table 1. tblspace tblspace entries

Component	Description
Page header	24 bytes, standard page-header information
Page-ending time stamp	4 bytes
Tblspace header	136 bytes, general tblspace information
Tblspace name	*database.owner.tablename* or *database.owner.indexname* Typically 30-40 bytes long but can be longer, depending on the length of the name.
Column information	8 bytes for each special column A special column is defined as a VARCHAR, BYTE, TEXT, or user-defined data type.
Index information	For attached indexes, each index in the partition has a 20-byte header that contains general information about the index, followed by a 4-byte entry for each column in the index. For detached indexes, a 4-byte entry for each column in the index.
Extent information	A 10-byte entry plus 10 bytes of information for each extent that is allocated to the tblspace. During the defragmentation of the tblspace, more bytes might be used.

Related reference:
oncheck -pt and -pT: Display tblspaces for a Table or Fragment

Tblspace Numbers

Each tblspace that is described in the tblspace tblspace receives a tblspace number. This tblspace number is the same value that is stored as the **partnum** field in the **systables** system catalog table and as the **partn** field in the **sysfragments** system catalog table.

The following SQL query retrieves the **partnum** for every table in the database (these can be located in several different dbspaces) and displays it with the table name and the hexadecimal representation of **partnum**:

```
SELECT tabname, tabid, partnum, HEX(partnum) hex_tblspace_name FROM systables
```

If the output includes a row with a table name but a **partnum** of 0, this table consists of two or more table fragments, each located in its own tblspace. For example, Figure 1 shows a table called **account** that has **partnum** 0.

Figure 1. Output from systables Query with partnum Values

```
tabname         tabid   partnum          hex_tblspace_name

sysfragments     25     1048611          0x00100023
branch          100     1048612          0x00100024
teller          101     1048613          0x00100025
account         102           0          0x00000000
history         103     1048615          0x00100027
results         104     1048616          0x00100028
```

To obtain the actual tblspace numbers for the fragments that make up the table, you must query the **sysfragments** table for the same database. Figure 2 shows that the **account** table from Figure 1 has three table fragments and three index fragments.

Figure 2. Output from sysfragments Table with partn Values

```
tabid  fragtype     partn          hex_tblspace_name

  102  T            1048614        0x00100026
  102  T            2097154        0x00200002
  102  T            3145730        0x00300002
  102  I            1048617        0x00100029
  102  I            2097155        0x00200003
  102  I            3145731        0x00300003
```

Tblspace Number Elements

The first page in a tblspace is logical page 0. (Physical page numbers refer to the address of the page in the chunk.) The root space tblspace **tblspace** is always contained in the first dbspace and on logical page 1 within the tblspace **tblspace**. (The bitmap page is page 0.)

Tblspace Tblspace Size

These tblspace **tblspace** pages are allocated as an extent when the dbspace is initialized. If the database server attempts to create a table, but the tblspace **tblspace** is full, the database server allocates a next extent to the tblspace.

When a table is removed from the dbspace, its corresponding entry in the tblspace **tblspace** is deleted.

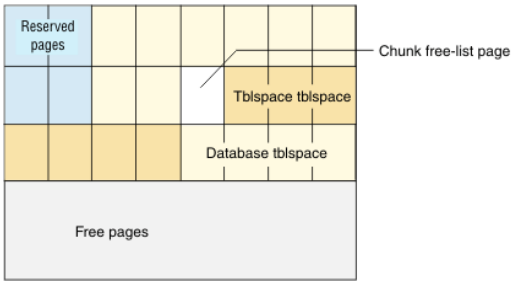
Tblspace Tblspace Bitmap Page

The first page of the tblspace **tblspace**, like the first page of any initial extent, is a bitmap that describes the page fullness of the following pages. Each page that follows has an entry on the bitmap page. If needed, additional bitmap pages are located throughout the contiguous space allocated for the tblspace, arranged so that each bitmap describes only the pages that follow it, until the next bitmap or the end of the dbspace. Bitmap pages fall at distinct intervals within tblspaces pages. Each bitmap page describes a fixed number of pages that follow it.

Structure of the Database Tblspace

The database tblspace appears only in the initial chunk of the root dbspace. The database tblspace contains one entry for each database managed by the database server. Figure 1 illustrates the location of the database tblspace.

Figure 1. Database Tblspace Location in Initial Chunk of Root Dbspace



Database Tblspace Number

The tblspace number of the database tblspace is always 0x100002. This tblspace number appears in an **onstat -t** listing if the database tblspace is active.

- Database Tblspace Entries

Database Tblspace Entries

Each database tblspace entry includes the following five components:

- Database name
- Database owner
- Date and time that the database was created
- The tblspace number of the **systables** system catalog table for this database
- Flags that indicate logging mode

The database tblspace includes a unique index on the database name to ensure that every database is uniquely named. For any database, the **systables** table describes each permanent table in the database. Therefore, the database tblspace only points to the detailed database information located elsewhere.

When the root dbspace is initialized, the database tblspace first extent is allocated. The initial-extent size and the next-extent size for the database tblspace are four pages. You cannot modify these values.

Structure and Allocation of an Extent

This section covers the following topics:

- Extent structure
- Next-extent allocation

- Extent Structure
- Next-Extent Allocation

Extent Structure

An extent is a collection of contiguous pages within a dbspace. Every permanent database table has two extent sizes associated with it. The initial-extent size is the number of kilobytes allocated to the table when it is first created. The next-extent size is the number of kilobytes allocated to the table when the initial extent, and every extent thereafter, becomes full.

Blobspaces do not use extents.

For specific instructions on how to specify and calculate the size of an extent, see your *IBM® Informix® Performance Guide*.

- Extent size
  The default size for first and next extents is 16 kilobytes. If this transforms to fewer than 4 pages in a particular dbspace, the database server uses the minimum extent size of 4 pages. If a dbspace has a size of 8 kilobytes, which transforms to 2 pages, the database server increases the extent size to 32 kilobytes.
- Page Types Within a Table Extent
- Page Types Within an Index Extent

Extent size

The default size for first and next extents is 16 kilobytes. If this transforms to fewer than 4 pages in a particular dbspace, the database server uses the minimum extent size of 4 pages. If a dbspace has a size of 8 kilobytes, which transforms to 2 pages, the database server increases the extent size to 32 kilobytes.

The maximum size of an extent is 2**31 pages, equivalent to the maximum chunk size.

If the chunk is smaller than the maximum size, the maximum extent size depends on the contiguous space available in the chunk.

Tblspaces that hold *index fragments* follow different rules for extent size. The database server bases the extent size for these tblspaces on the extent size for the corresponding table fragment. The database server uses the ratio of the row size to index key size to assign an appropriate extent size for the index tblspace (see the sections on estimating index page size and fragmenting table indexes in the *IBM® Informix® Performance Guide*).

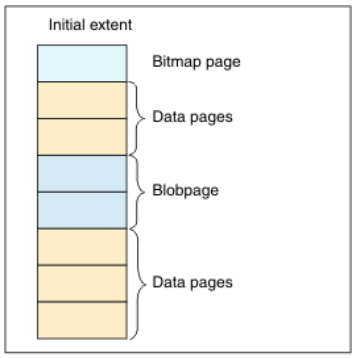
The maximum number of extents for a partition is 32767.

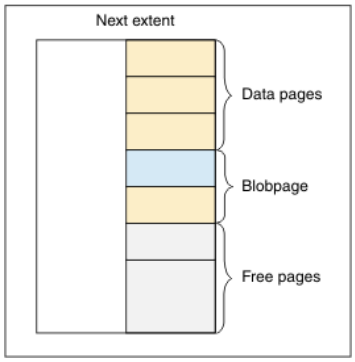
Page Types Within a Table Extent

Within the extent, individual pages contain different types of data. Extent pages for a table can be separated into the following categories:

- Data pages
  Data pages contain the data rows for the table.
- Bitmap pages
  Bitmap pages contain control information that monitors the fullness of every page in the extent.
- Blobpages
  Blobpages contain TEXT and BYTE data that is stored with the data rows in the dbspace. TEXT and BYTE data that resides in a blobspace is stored in blobpages, a structure that is completely different than the structure of a dbspace blobpage.
- Free pages
  Free pages are pages in the extent that are allocated for tblspace use, but whose function has not yet been defined. Free pages can be used to store any kind of information: data, including TEXT or BYTE data types; index; or bitmap.

Figure 1 illustrates the possible structure of a nonfragmented table with an initial-extent size of 8 pages and a next-extent size of 16 pages.
Figure 1. Extent Structure of a Table



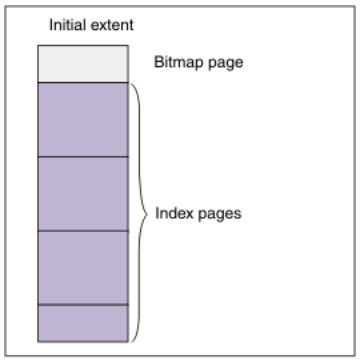


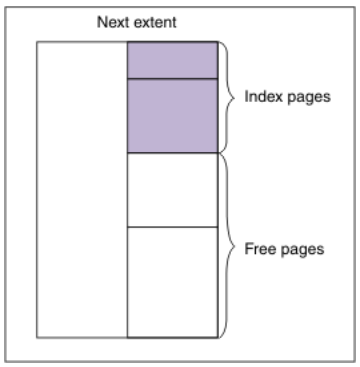
Page Types Within an Index Extent

The database server stores index pages into different tblspaces than the table with which it is associated. Within the extent, individual index pages contain different types of data. Index pages can be separated into the following categories:

- Index pages (root, branch, and leaf pages)
  Index pages contain the index information for the table.
- Bitmap pages
  Bitmap pages contain control information that monitors the fullness of every page in the extent.
- Free pages
  Free pages are pages in the extent that are allocated for tblspace use, but whose function has not yet been defined. Free pages can be used to store any kind of information: data, index, TEXT or BYTE data, or bitmap.

All indexes are detached unless you explicitly specify attached indexes.
Important: An extent that is allocated for a table fragment does not contain index pages. Index pages for a fragmented table always reside in a separate tblspace. For more information, see fragmenting table indexes in the chapter on table fragmentation and PDQ in the *IBM® Informix® Administrator's Guide*.
Figure 1 illustrates the extent structure of an index.
Figure 1. Extent Structure of an Index





Next-Extent Allocation

After the initial extent fills, the database server attempts to allocate another extent of contiguous disk space. The procedure that the database server follows is referred to as next-extent allocation.

Extents for a tblspace are tracked as one component of the tblspace **tblspace** information for the table. The maximum number of extents allocated for any tblspace is application and machine dependent because it varies with the amount of space available on the tblspace **tblspace** entry.

- Next-Extent Size
- Extent size doubling
- Lack of Contiguous Space
- Merge of Extents for the Same Table

Next-Extent Size

The number of kilobytes that the database server allocates for a next extent is, in general, equal to the size of a next extent, as specified in the SQL statement CREATE TABLE. However, the actual size of the next-extent allocation might deviate from the specified size because the allocation procedure takes into account the following three factors:

- Number of existing extents for this tblspace
- Availability of contiguous space in the chunk and dbspace
- Location of existing tblspace extents

The effect of each of these factors on next-extent allocation is explained in the paragraphs that follow and in Figure 1.

Extent size doubling

For permanent tables or user-defined temporary tables, the size of the next extent for every allocation is automatically doubled. The size doubles up to 128 kilobytes (KB). For example, if you create a table with the NEXT SIZE equal to 15 KB, the database server allocates the first extent at a size of 15 KB. The next extent is allocated at 30 KB, and the extent after that is allocated at 60 KB. When the extent size reaches 128 KB, the size is doubled only when the remaining space in the table is less than 10% of the total allocated space in the table.

For system-created temporary tables, the next-extent size begins to double after 4 extents have been added.

Lack of Contiguous Space

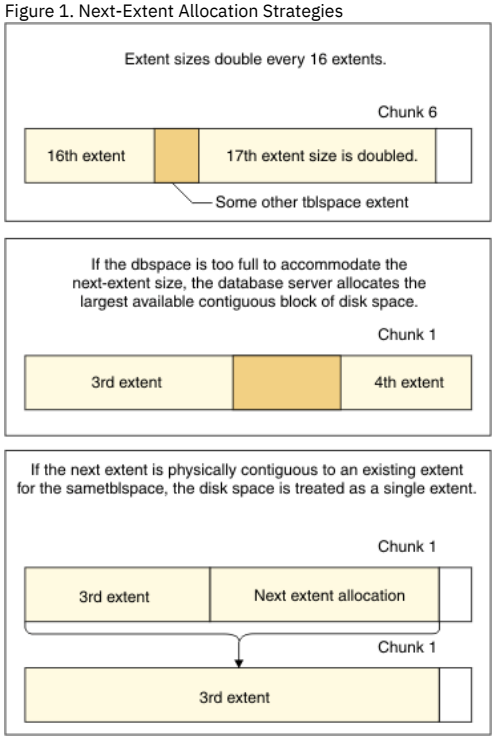
If the database server cannot find available contiguous space in the first chunk equal to the size specified for the next extent, it extends the search to the next chunk in the dbspace. Extents are not allowed to span chunks.

If the database server cannot find adequate contiguous space anywhere in the dbspace, it allocates to the table the largest available amount of contiguous space. (The minimum allocation is four pages. The default value is eight pages.) No error message is returned if an allocation is possible, even when the amount of space allocated is less than the requested amount.

Merge of Extents for the Same Table

If the disk space allocated for a next extent is physically contiguous with disk space already allocated to the same table, the database server allocates the disk space but does not consider the new allocation as a separate extent. Instead, the database server extends the size of the existing contiguous extent. Thereafter, all disk-space reports reflect the allocation as an extension of the existing extent. That is, the number of extents reported is always the number of physically distinct extents, not the number of times a next extent has been allocated plus one (the initial extent). Figure 1 illustrates extent-allocation strategies.

Figure 1. Next-Extent Allocation Strategies



After disk space is allocated to a tblspace as part of an extent, the space remains dedicated to that tblspace even if the data contained in it is deleted. For alternative methods of reclaiming this empty disk space, see your *IBM® Informix® Performance Guide*.

Structure and Storage of a Dbspace Page

The basic unit of database server I/O is a page. Page size might vary among computers.

In Informix®, the page size depends on the operating system.

- Rows in Nonfragmented Tables
- Rows in Fragmented Tables
- Recommendations on Use of Rowid
- Data-Row Format and Storage

Rows in Nonfragmented Tables

The database server can store rows that are longer than a page. The database server also supports the VARCHAR data type, which results in rows of varying length. As a result, rows do not conform to a single format.

Rows within a table are not necessarily the same length if the table contains one or more columns of type VARCHAR. In addition, the length of a row in such a table might change when an end user modifies data contained in the VARCHAR column.

The length of a row can be greater than a page.

TEXT and BYTE data is not stored within the data row. Instead, the data row contains a 56-byte descriptor that points to the location of the data. The descriptor can point to a dbspace page.

The descriptor can point to a blobspace blobpage.

For instructions about how to estimate the length of fixed-length and variable-length data rows, see your *IBM® Informix® Performance Guide*.

- Definition of Rowid
- Use of Rowids

Definition of Rowid

Informix® uses two different types of rowids to identify data in tables:

- *Serial rowid*
  These rowids are fields in a table and are assigned to tables created with the WITH ROWID option.
- *Internal rowid*
  The database server identifies each data row in a table with a unique internal rowid. This rowid identifies the location of the row within the dbspace.

  To obtain the internal rowids for a table, use the **oncheck -pD** option. For more information, see oncheck -cd and oncheck -cD commands: Check pages.

In a nonfragmented table, the term *rowid* refers to a unique 4-byte integer that defines the physical location of the row in the table. The page that contains the first byte of the data row is the page that is specified by the rowid. This page is called the data row *home page*.

Fragmented tables can also have rowids, but they are implemented in a different way. For more information on this topic, see Rows in Fragmented Tables.

Use of Rowids

Every data row in a nonfragmented table is uniquely identified by an unchanging rowid. When you create an index for a nonfragmented table, the rowid is stored in the index pages associated with the table to which the data row belongs. When the database server requires a data row, it searches the index to find the key value and uses the corresponding rowid to locate the requested row. If the table is not indexed, the database server might sequentially read all the rows in the table.

Eventually, a row might outgrow its original storage location. If this occurs, a *forward pointer* to the new location of the data row is left at the position defined by the rowid. The forward pointer is itself a rowid that defines the page and the location on the page where the data row is now stored.

Rows in Fragmented Tables

Unlike rows in a nonfragmented table, the database server does *not* assign a rowid to rows in fragmented tables. If you want to access data by rowid, you must explicitly create a rowid column as described in your *IBM® Informix® Performance Guide*. If user applications attempt to reference a rowid in a fragmented table that does not contain a rowid that you explicitly created, the database server returns an appropriate error code to the application.

- Access to Data in Fragmented Tables with Rowid

Access to Data in Fragmented Tables with Rowid

From the viewpoint of an application, the functionality of a rowid column in a fragmented table is identical to the rowid of a nonfragmented table. However, unlike the rowid of a nonfragmented table, the database server uses an index to map the rowid to a physical location.

When the database server accesses a row in a fragmented table using the rowid column, it uses this index to look up the physical address of the row before it attempts to access the row. For a nonfragmented table, the database server uses direct physical access without an index lookup. As a consequence, accessing a row in a fragmented table using rowid takes slightly longer than accessing a row using rowid in a nonfragmented table. You should also expect a small performance impact on the processing of inserts and deletes due to the cost of maintaining the rowid index for fragmented tables.

Primary-key access can lead to significantly improved performance in many situations, particularly when access is in parallel.

Recommendations on Use of Rowid

It is recommended that application developers use primary keys as a method of access rather than rowids. Because primary keys are defined in the ANSI specification of SQL, using them to access data makes your applications more portable.

For a complete description on how to define and use primary keys to access data, see the *IBM® Informix® Guide to SQL: Reference* and the *IBM Informix Guide to SQL: Tutorial*.

Data-Row Format and Storage

The variable length of a data row has the following consequences for row storage:

- A page might contain one or more whole rows.
- A page might contain portions of one or more rows.
- A page might contain a combination of whole rows and partial rows.
- An updated row might increase in size and become too long to return to its original storage location in a row.

The following paragraphs describe the guidelines that the database server follows during data storage.

Storage of Row

To minimize retrieval time, rows are not broken across page boundaries unnecessarily. Rows that are shorter than a page are always stored as whole rows. A page is considered *full* when the count of free bytes is less than the number of bytes needed to store a row of maximum size.

Location of Rows

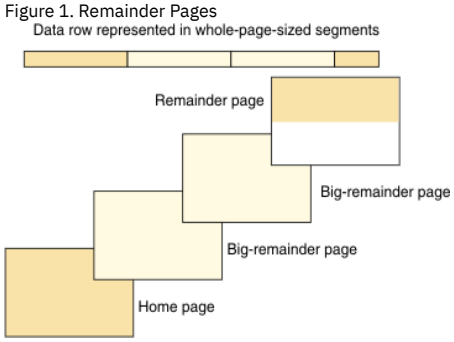
When the database server receives a row that is longer than a page, the row is stored in as many whole pages as required. The database server then stores the trailing portion in less than a full page.

The page that contains the first byte of the row is the row home page. The number of the home page becomes the logical page number contained in the rowid. Each full page that follows the home page is referred to as a big-remainder page. If the trailing portion of the row is less than a full page, it is stored on a remainder page.

After the database server creates a remainder page to accommodate a long row, it can use the remaining space in this page to store other rows.

Figure 1 illustrates the concepts of home page, big-remainder page, and remainder page.

Figure 1. Remainder Pages



Page Compression

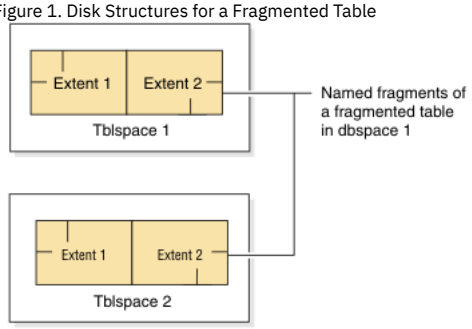
Over time, the free space on a page can become fragmented. When the database server attempts to store data, it first checks row length against the number of free bytes on a page to determine if the row fits. If adequate space is available, the database server checks if the page contains adequate contiguous free space to hold the row (or row portion). If the free space is not contiguous, the database server calls for page compression.

Structure of Fragmented Tables

Although table fragmentation is transparent to applications, as database server administrator you should be aware of how the database server allocates disk space for table fragments and how the database server identifies rows in those fragments.

Each table fragment has its own tblspace with a unique *tblspace_id* or *fragment_id*. Figure 1 shows the disk allocation for a fragmented table that resides in named fragments of the same dbspace.

Figure 1. Disk Structures for a Fragmented Table



Attached Indexes

With an attached index, the index and data are fragmented in the same way. You can decide whether to store the index pages with the corresponding data pages in the same dbspace or store them in separate dbspaces. For information on choosing a fragmentation strategy, see the *IBM® Informix® Performance Guide*.

Detached Indexes

For detached indexes, the table fragment and index fragment are stored in tblspaces in separate dbspaces.

Structure of B-Tree Index Pages

This section provides general information about the structure of B-tree index pages. It is designed as an overview for the interested reader. For more information on B-tree indexes, see your *IBM® Informix® Performance Guide*.

- Definition of B-tree terms
  The database server uses a B-tree structure to organize index information.
- Logical Storage of Indexes
- Functional Indexes

Related reference:
FILLFACTOR configuration parameter

Definition of B-tree terms

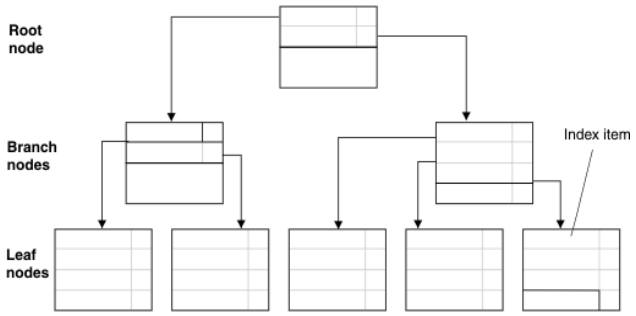
The database server uses a B-tree structure to organize index information.

Figure 1 shows that a fully developed B-tree index is composed of the following three different types of index pages or nodes:

- One *root node*
  A root node contains node pointers to branch nodes.
- Two or more *branch nodes*
  A branch node contains pointers to leaf nodes or other branch nodes.
- Many *leaf nodes*
  A leaf node contains index items and horizontal pointers to other leaf nodes.

Each node serves a different function. The following sections describe each node and the role that it plays in indexing.

Figure 1. Full B-Tree Structure



Index items

The fundamental unit of an index is the *index item*. An index item contains a key value that represents the value of the indexed column for a particular row. An index item also contains rowid information that the database server uses to locate the row in a data page.

Index nodes

A node is an index page that stores a group of index items.

Forest of trees indexes as alternatives to traditional B-Tree indexes

Unlike a traditional B-tree index, a forest of trees index is a large B-tree index that is divided into smaller subtrees with multiple root nodes and fewer levels. You can create a forest of trees index as an alternative to a B-tree index when you want to alleviate root node contention and allow more concurrent users to access the index without waiting.

Related information:
Forest of trees indexes

Logical Storage of Indexes

This section presents an overview of how the database server creates and fills an index.

- Creation of Root and Leaf Nodes
- Creation of branch nodes
- Duplicate Key Values
- Key-Value Locking
- Adjacent Key Locking
- Freed Index Pages
- Filling Indexes
- Calculating the Length of Index Items

Creation of Root and Leaf Nodes

When you create an index for an empty table, the database server allocates a single index page. This page represents the root node and remains empty until you insert data in the table.

At first, the root node functions in the same way as a leaf node. For each row that you insert into the table, the database server creates and inserts an index item in the root node. Figure 1 illustrates how a root node appears before it fills.

Figure 1. Root Node

Root node 1

Albertson	rowid information
Baxter	rowid information
Beatty	rowid information
Currie	rowid information
Keyes	rowid information
Lawson	rowid information
Mueller	rowid information

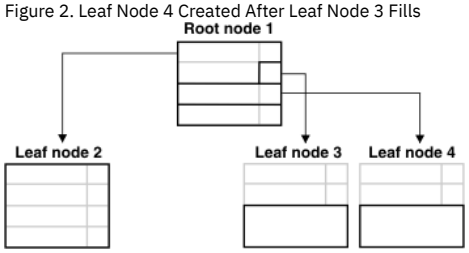
When the root node becomes full of index items, the database server splits the root node by performing the following steps:

- Creates two leaf nodes
- Moves approximately half of the root-node entries to each of the newly created leaf nodes
- Puts pointers to leaf nodes in the root node

As you add new rows to a table, the database server adds index items to the leaf nodes. When a leaf node fills, the database server creates a new leaf node, moves part of the contents of the full index node to the new node, and adds a node pointer to the new leaf node in the root node.

For example, suppose that leaf node 3 in Figure 2 becomes full. When this situation occurs, the database server adds yet another leaf node. The database server moves part of the records from leaf node 3 to the new leaf node, as Figure 2 shows.

Figure 2. Leaf Node 4 Created After Leaf Node 3 Fills



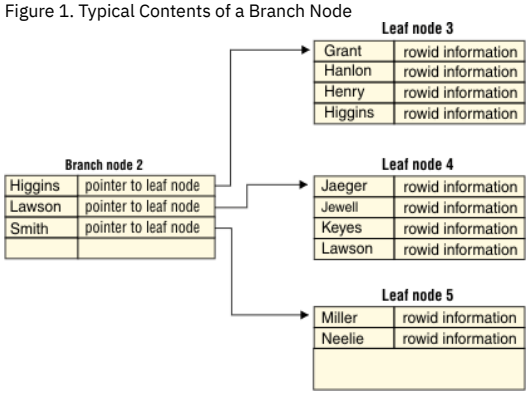
Creation of branch nodes

Eventually, as you add rows to the table, the database server fills the root node with node pointers to all the existing leaf nodes. When the database server splits yet another leaf node, and the root node has no room for an additional node pointer, the following process occurs.

The database server splits the root node and divides its contents among two newly created branch nodes. As index items are added, more and more leaf nodes are split, causing the database server to add more branch nodes. Eventually, the root node fills with pointers to these branch nodes. When this situation occurs, the database server splits the root node again. The database server then creates yet another branch level between the root node and the lower branch level. This process results in a four-level tree, with one root node, two branch levels, and one leaf level. The B-tree structure can continue to grow in this way to a maximum of 20 levels.

Branch nodes can point either to other branch nodes below them (for large indexes of four levels or more) or to leaf nodes. In Figure 1, the branch node points to leaf nodes only. The first item in the left branch node contains the same key value as the largest item in the leftmost leaf node and a node pointer to it. The second item contains the largest item in the next leaf node and a node pointer to it. The third item in the branch node contains only a pointer to the next higher leaf node. Depending on the index growth, this third item can contain the actual key value in addition to the pointer at a later point during the life span of the index.

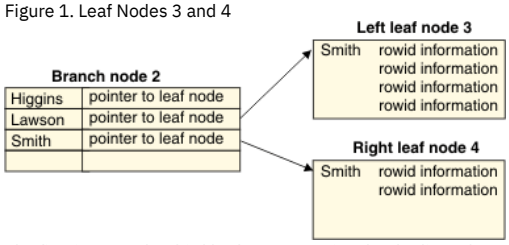
Figure 1. Typical Contents of a Branch Node



Duplicate Key Values

Duplicate key values occur when the value of an indexed column is identical for multiple rows. For example, suppose that the third and fourth leaf nodes of a B-tree structure contain the key value `Smith`. Suppose further that this value is duplicated six times, as Figure 1 illustrates.

Figure 1. Leaf Nodes 3 and 4



The first item on the third leaf page contains the duplicate key value, `Smith`, and the rowid information for the first physical row in the table that contains the duplicate key value. To conserve space, the second item does not repeat the key value `Smith` but instead contains just the rowid information. This process continues throughout the page; no other key values are on the leaf, only rowid information.

The first item on the fourth leaf page again contains the duplicated key value and rowid information. Subsequent items contain only rowid information.

Now consider the branch node. The third item in the branch node contains the same key value and rowid as the largest item in the third leaf node and a node pointer to it. The fourth item would contain only a node pointer to the fourth leaf node, thus saving the space of an additional duplicate key value.

Key-Value Locking

To increase concurrency, the database server supports *key-value* locking in the B-tree index. Key-value locking locks only the value of the key instead of the physical location in the B-tree index.

One of the most important uses for key-value locking is to assure that a unique key remains unique through the end of the transaction that deleted it. Without this protection mechanism, user A might delete a unique key within a transaction, and user B might insert a row with the same key before the transaction commits. This scenario makes rollback by user A impossible. Key-value locking prevents user B from inserting the row until the end of user A's transaction.

Adjacent Key Locking

With Repeatable Read isolation level, the database server is required to protect the *read set*. The read set consists of the rows that meet the filters in the WHERE clause of the query. To guarantee that the rows do not change, the database server obtains a lock on the index item that is adjacent to the right-most item of the read set.

Freed Index Pages

When the database server physically removes an index item from a node and frees an index page, the freed page is reused.

Filling Indexes

When you create an index, you can specify how densely or sparsely filled you want the index. The index fill factor is a percentage of each index page that will be filled during the index build. Use the FILLFACTOR option of the CREATE INDEX statement or the FILLFACTOR configuration parameter to set the fill factor. This option is particularly useful for indexes that you do not expect to grow after they are built. For additional information about the FILLFACTOR option of the CREATE INDEX statement, see the *IBM® Informix® Guide to SQL: Syntax*.

Calculating the Length of Index Items

For data types other than VARCHAR, the length of an index item is calculated by adding the length of the key value plus 5 bytes for each rowid information associated with the key value.

The key values in an index are typically of fixed length. If an index holds the value of one or more columns of the VARCHAR data type, the length of the key value is at least the sum of the length-plus-one of each VARCHAR value in the key.

In Informix®, the maximum length of a key value is 390 bytes. The combined size of VARCHAR columns that make up a key must be less than 390, minus an additional byte for each VARCHAR column. For example, the key length of the index that the database server builds for the following statements equals 390, or `((255+1) + (133+1))`:

```
CREATE TABLE T1 (c1 varchar(255, 10), c2 varchar(133, 10));
CREATE INDEX I1 on T1(c1, c2);
```


Functional Indexes

A *functional index* is one in which all keys derive from the results of a function. If you have a column of pictures, for example, and a function to identify the predominant color, you can create an index on the result of the function. Such an index would enable you to quickly retrieve all pictures having the same predominant color, without re-executing the function.

A functional index uses the same B-tree structure as any other B-tree index. The only difference is that the determining function is applied during an insert or an update whenever the column that is the argument to the function changes. For more information on the nature of functional indexes, refer to your *IBM® Informix® Performance Guide*.

To create a functional index, use the CREATE FUNCTION and CREATE INDEX statements. For more information on these statements, refer to the *IBM Informix Guide to SQL: Syntax*.

Structure of R-Tree Index Pages

An index structure that relies on one-dimensional ordering of key values does not work for spatial data; for example, two dimensional geometric shapes such as circles, squares, and triangles. Efficient retrieval of spatial data, such as the data used in geographic information systems (GIS) and computer-aided design (CAD) applications, requires an access method that handles multidimensional data. The database server implements an R-tree index to access spatial data efficiently. For information about the structure of index pages, refer to the *IBM® Informix® R-Tree Index User's Guide*.

Storage of Simple Large Objects

This section explains the structures and storage techniques that the database server uses to store simple large objects (TEXT or BYTE data).

- Structure of a Blobspace
- Structure of a Dbspace Blobpage
- Simple-Large-Object Storage and the Descriptor
- Blobspace Page Types
- Structure of a Blobspace Blobpage

Structure of a Blobspace

When you create a blobspace, you can specify the effective size of the data pages, which are called blobpages. The blobpage size for the blobspace is specified when the blobspace is created. Blobpage size must be a multiple of page size. (For information on determining database server page size, see the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.) All blobpages within a blobspace are the same size, but the size of the blobpage can vary between blobspaces. Blobpage size can be greater than the page size because data stored in a blobspace is never written to the page-sized buffers in shared memory.

The advantage of customizing the blobpage size is storage efficiency. Within a blobspace, TEXT and BYTE data is stored in one or more blobpages, but simple large objects do not share blobpages. Storage is most efficient when the TEXT or BYTE data is equal to or slightly smaller than the blobpage size.

The blobspace free-map pages and bitmap pages are the size specified as a database server page, which enables them to be read into shared memory and to be logged.

When the blobspace is first created, it contains the following structures:

- Blobspace free-map pages
- The blobspace bitmap that tracks the free-map pages
- Unused blobpages

Structure of a Dbspace Blobpage

TEXT or BYTE data that is stored in the dbspace is stored in a blobpage. The structure of a dbspace blobpage is similar to the structure of a dbspace data page. The only difference is an extra 12 bytes that can be stored along with the TEXT or BYTE data in the data area.

Simple large objects can share dbspace blobpages if more than one simple large object can fit on a single page, or if more than one trailing portion of a simple large object can fit on a single page.

For a discussion of how to estimate the number of dbspace blobpages needed for a specific table, see your *IBM® Informix® Performance Guide*.

Each segment of TEXT or BYTE data stored in a dbspace page might be preceded by up to 12 bytes of information that does not appear on any other dbspace page. These extra bytes are overhead.

Simple-Large-Object Storage and the Descriptor

Data rows that include TEXT or BYTE data do not include the data in the row itself. Instead, the data row contains a 56-byte descriptor with a forward pointer (rowid) to the location where the first segment of data is stored.

The descriptor can point to one of the following items:

- A page (if the data is stored in a dbspace)
- A blobpage (if the data is stored in a blobspace)

- Creation of Simple Large Objects
- Deletion or Insertion of Simple Large Objects
- Size Limits for Simple Large Objects

Creation of Simple Large Objects

When a row that contains TEXT or BYTE data is to be inserted, the simple large objects are created first. After the simple large objects are written to disk, the row is updated with the descriptor and inserted.

Deletion or Insertion of Simple Large Objects

The database server cannot modify simple large objects. It can only insert or delete them. Deleting a simple large object means that the database server frees the space consumed by the deleted object for reuse.

When TEXT or BYTE data is updated, a new simple large object is created, and the data row is updated with the new blob descriptor. The old image of the row contains the descriptor that points to the obsolete value for the simple large object. The space consumed by the obsolete simple large object is freed for reuse after the update is committed. Simple large objects are automatically deleted if the rows that contain their blob descriptors are deleted. (Blobpages that stored a deleted simple large object are not available for reuse until the logical log that contains the original INSERT record for the deleted simple large object is backed up. For more information, see backing up logical-log files to free blobpages in the chapter on what is the logical log in the *IBM® Informix® Administrator's Guide*.)

Size Limits for Simple Large Objects

The largest simple large object that the blob descriptor can accommodate is (2^{31} - 1), or about 2 gigabytes.

Blobspace Page Types

Every blobspace chunk contains three types of pages:

- A blobspace free-map page
- A bitmap page
- Blobpages

Blobspace Free-Map Page

The blobspace free-map page identifies unused blobpages so that the database server can allocate them as part of simple-large-object creation. When a blobpage is allocated, the free-map entry for that page is updated. All entries for a single simple large object are linked.

A blobspace free-map page is the size of one database server page. Each entry on a free-map page is 8 bytes, stored as two 32-bit words, as follows:

- The first bit in the first word specifies whether the blobpage is free or used.
- The next 31 bits in the first word identify the logical-log file that was current when this blobpage was written. (This information is needed for logging TEXT or BYTE data.)
- The second word contains the tblspace number associated with the simple large object stored on this page.

The number of entries that can fit on a free-map page depends on the page size of your computer. The number of free-map pages in a blobspace chunk depends on the number of blobpages in the chunk.

Blobspace Bitmap Page

The blobspace bitmap page tracks the fullness and number of blobspace free-map pages in the chunk. Each blobspace bitmap page is capable of tracking a quantity of free-map pages. The size of the blobspace bitmap page depends on the size of the system page. If the system page is 2K, the blobspace bitmap page can track 2,032,128 blobpages. If the system page is 4K, the blobspace bitmap page can track 8,258,048 blobpages.

Blobpage

The blobpage contains the TEXT or BYTE data. Blobpage size is specified by the database server administrator who creates the blobspace. Blobpage size is specified as a multiple of the page size.

Structure of a Blobspace Blobpage

The storage strategy used to store simple large objects in a blobspace differs from the dbspace storage strategy. The database server does not combine whole simple large objects or portions of a simple large object on a single blobspace blobpage. For example, if blobspace blobpages are 24 kilobytes each, a simple large object that is 26 kilobytes is stored on two 24-kilobyte pages. The extra 22 kilobytes of space remains unused.

The structure of a blobpage includes a blobpage header, the TEXT or BYTE data, and a page-ending time stamp. The blobpage header includes, among other information, the page-header time stamp and the blob time stamp associated with the forward pointer in the data row. If a simple large object is stored on more than one blobpage, a forward pointer to the next blobpage and another blob time stamp are also included in the blobpage header.

Sbspace Structure

An sbspace is similar to a blobspace except that it holds smart large objects.

When an sbspace is created in a database, it contains an sbspace descriptor. Each sbspace chunk contains the following structures:

- Sbspace chunk descriptors
- Chunk free-page list
- An sbspace metadata area (up to one for each chunk)
- Reserved data areas (up to two for each chunk)
- User-data areas (up to two for each chunk)

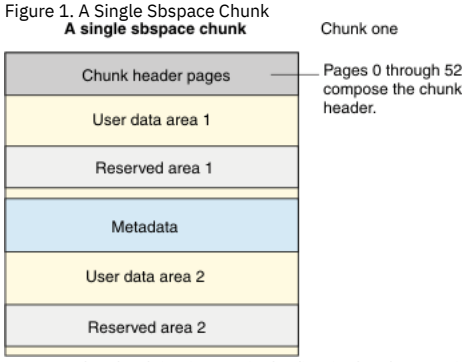
For best performance, it is recommended that the metadata area be located in the middle of the sbspace. The database server automatically places the metadata area in the correct location. However, to specify the location of the metadata area, specify the **-Mo** flag in the **onspaces** command.

If you do not specify the size of the metadata area in the **-Ms** flag of the **onspaces** command, the database server uses the value of AVG_LO_SIZE (defaults to 8 kilobytes) to calculate the size of the metadata area. For more information, see Creating an Sbspace with the -Df option.

Normally, you can let the system calculate the metadata size for you. If you want to estimate the size of the metadata area, see the chapter on table performance considerations in the *IBM® Informix® Performance Guide*.

Figure 1 illustrates the chunk structure of an sbspace as it appears immediately after the sbspace is created. Each reserved area can be allocated to either the user-data or metadata area. Reserved areas are always within the user-data area of the chunk.

Figure 1. A Single Sbspace Chunk



Because the chunk in Figure 1 is the first in the sbspace, it contains an sbspace descriptor. The chunk descriptor tblspace in **chunk one** contains information about chunk one and all chunks added to the sbspace thereafter.

- Structure of the metadata area
  An sbspace contains a metadata area for each chunk in the sbspace.
- Sbpage Structure

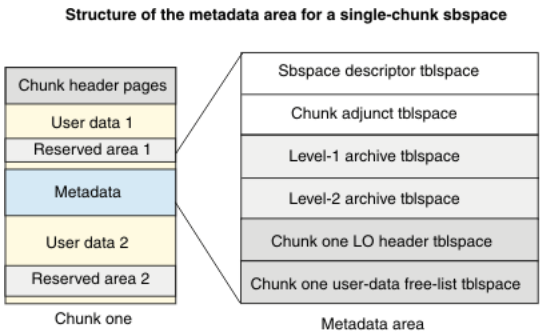
Related reference:
SBSPACENAME configuration parameter
SYSSBSPACENAME configuration parameter

Structure of the metadata area

An sbspace contains a metadata area for each chunk in the sbspace.

As with the chunk header pages, four areas are exclusive to the first chunk in a sbspace: the sbspace descriptor tblspace, the chunk adjunct tblspace, and the level-1 and level-2 archive tblspaces. The tblspace header section contains a tblspace header for each of these tblspaces (notably excluding the tblspace **tblspace**). Figure 1 shows the layout of the metadata in the single-chunk sbspace.

Figure 1. Structure of the metadata area for a single-chunk sbspace



When you specify the sbspace name in the **oncheck -ps** option, you can display the number of pages allocated and used for each tblspace in the metadata area.

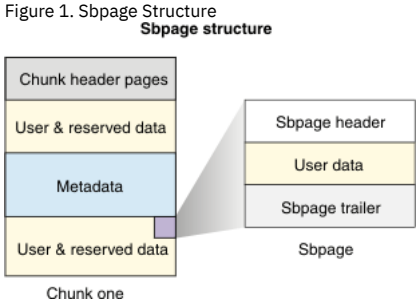
The following items describe how the metadata area grows:

- The sbspace descriptor tblspace does not grow.
- The chunk adjunct tblspace grows as chunks are added.
- The LO header tblspace grows as large objects are added to the chunk.
- The tblspace for user-data free list grows if free spaces in the chunk are heavily fragmented.

Sbpage Structure

Each sbpage is composed of three elements: an sbpage header, the actual user data itself, and an sbpage trailer. Figure 1 shows the structure of an sbpage. The sbpage header consists of the standard page header. The sbpage trailer is used to detect an incomplete write on the page and to detect page corruption.

Figure 1. Sbpage Structure



Time Stamps

The database server uses a time stamp to identify a time when an event occurred relative to other events of the same kind. The time stamp is not a literal time that refers to a specific hour, minute, or second. It is a 4-byte integer that the database server assigns sequentially.

Database and Table Creation: What Happens on Disk

This section explains how the database server stores data related to the creation of a database or table and allocates the disk structures that are necessary to store your data.

- Database Creation
- Table Creation

Database Creation

After the root dbspace exists, users can create a database. The paragraphs that follow describe the major events that occur on disk when the database server adds a new database.

- Disk-Space Allocation for System Catalog Tables
- Tracking of System Catalog Tables

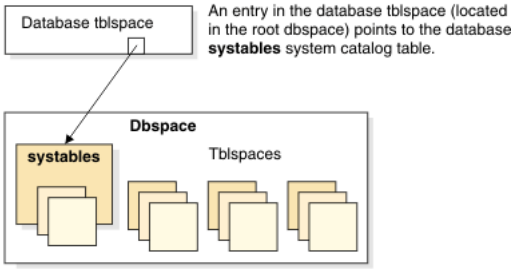
Disk-Space Allocation for System Catalog Tables

The database server searches the chunk free-list pages in the dbspace, looking for free space in which to create the system catalog tables. For each system catalog table, in turn, the database server allocates eight contiguous pages, the size of the initial extent of each system catalog table. The tables are created individually and do not necessarily reside next to each other in the dbspace. They can be located in different chunks. As adequate space is found for the initial extent of each table, the pages are allocated, and the associated chunk free-list page is updated.

Tracking of System Catalog Tables

The database server tracks newly created databases in the database tblspace, which resides in the root dbspace. An entry describing the database is added to the database tblspace in the root dbspace. (See Structure of the Database Tblspace.) For each system catalog table, the database server adds a one-page entry to the tblspace **tblspace** in the dbspace where the database was built. (See Structure of the Tblspace Tblspace.) Figure 1 illustrates the relationship between the database tblspace entry and the location of the **systables** system catalog table for the database.

Figure 1. New Databases



For instructions on how to list your databases after you create them, see monitoring databases in the chapter on managing database-logging status in the *IBM® Informix® Administrator's Guide*.

Table Creation

After the root dbspace exists, and a database has been created, users with the necessary SQL privileges can create a database table. When users create a table, the database server allocates disk space for the table in units called extents (see what is an extent in the chapter on where data is stored in the *IBM® Informix® Administrator's Guide*). The paragraphs that follow describe the major events that occur when the database server creates a table and allocates the initial extent of disk space.

- Disk-Space Allocation
- Entry in the Tblspace Tblspace
- Entries in the System Catalog Tables
- Creation of a Temporary Table

Disk-Space Allocation

The database server searches the chunk free-list pages in the dbspace for contiguous free space equal to the initial extent size for the table. When adequate space is found, the pages are allocated, and the associated chunk free-list page is updated.

If the database server cannot find adequate contiguous space anywhere in the dbspace, it allocates to the table the largest available amount of contiguous space. No error message is returned if an allocation is possible, even when the amount of space allocated is less than the requested amount. If the minimum extent size cannot be allocated, an error is returned. (Extents cannot span two chunks.)

Entry in the Tblspace Tblspace

The database server adds a one-page entry for this table to the tblspace **tblspace** in this dbspace. The tblspace number assigned to this table is derived from the logical page number in the tblspace **tblspace** where the table is described. See Tblspace Numbers.

The tblspace number indicates the dbspace where the tblspace is located. Tblspace extents can be located in any of the dbspace chunks.

If you must know exactly where the tblspace extents are located, execute the **oncheck -pe** command for a listing of the dbspace layout by chunk.

Entries in the System Catalog Tables

The table itself is fully described in entries stored in the system catalog tables for the database. Each table is assigned a table identification number or *tabid*. The tabid value of the first user-defined table object in a database is always `100`. (The object whose tabid = 100 might also be a view, synonym, or a sequence.) For a complete discussion of the system catalog, see the *IBM® Informix® Guide to SQL: Reference*.

A table can be located in a dbspace that is different than the dbspace that contains the database. The tblspace itself is the sum of allocated extents, not a single, contiguous allocation of space. The database server tracks tblspaces independently of the database.

Creation of a Temporary Table

The tasks involved in creating temporary tables are similar to the tasks that the database server performs when it adds a new permanent table. The key difference is that temporary tables do not receive an entry in the system catalog for the database. For more information, see the section defining a temporary table, in the chapter on where data is stored in the *IBM® Informix® Administrator's Guide*.

Interpreting Logical-Log Records

In This Chapter

To display the logical-log records that the logical-log files contain, use the **onlog** utility.

This chapter provides the following information:

- Brief guidance on reading logical-log records
- A listing of the different logical-log record types

In general, you do not need to read and interpret your logical-log files. However, **onlog** output is useful in debugging situations. For example, you might want to use **onlog** to track a specific transaction or to see what changes the database server made to a specific tblspace. You can also use **onlog** to investigate the cause of an error that occurs during a rollforward. For more information, see onlog: Display Logical-Log Contents.

About Logical-Log Records

Most SQL statements generate multiple logical-log records. Interpreting logical-log records is more complicated when the database server records the following events in the logical log:

- A transaction that drops a table or index
- A transaction that rolls back
- A checkpoint in which transactions are still active
- A distributed transaction

The following sections discuss the logical-log records for these events.

Transactions That Drop a Table or Index

Once the database server drops a table or index from a database, it cannot roll back that drop operation. If a transaction contains a DROP TABLE or DROP INDEX statement, the database server handles this transaction as follows:

1. The database server completes all the other parts of the transaction and writes the relevant logical-log records.
2. The database server writes a BEGCOM record to the logical log and the records associated with the DROP TABLE or DROP INDEX (DINDEX, for example).

3. The database server writes a COMMIT record.

If the transaction is terminated unexpectedly after the database server writes the BEGCOM record to the logical log, the database server rolls *forward* this transaction during recovery because it cannot roll back the drop operation.

Transactions That Are Rolled Back

When a rollback occurs, the database server generates a compensation-log record (CLR) for each record in the logical log that is rolled back. The database server uses the CLRs if a system failure takes place *during a rollback*. The CLRs provide the database server with information on how far the rollback progressed before the failure occurred. In other words, the database server uses the CLRs to log the rollback.

If a CLR contains the phrase `includes next record`, the next log record that is printed is included within the CLR log record as the compensating operation. Otherwise, you must assume that the compensating operation is the logical undo of the log record to which the **link** field of the CLR points.

Checkpoints with Active Transactions

If any transactions are active at the time of a checkpoint, checkpoint records include subentries that describe each of the active transactions using the following columns:

- Log begin (decimal format)
- Transaction ID (decimal format)
- Unique log number (decimal format)
- Log position (hexadecimal format)
- User name

Distributed Transactions

When distributed transactions (transactions that span multiple database servers) generate log records, they are slightly different than nondistributed transactions. You might need to read and interpret them to determine the state of the transaction on both database servers if a failure occurs as a transaction was committing.

The following log records are involved in distributed transactions:

- BEGPREP
- ENDTRANS
- HEURTX
- PREPARE
- TABLOCKS

For more information about this type of logical-log record, see the material on two-phase commit and logical-log records in the *IBM® Informix® Administrator's Guide*.

If you are performing distributed transactions with TP/XA, the database server uses an XAPREPARE record instead of a PREPARE record.

Logical-Log Record Structure

Each logical-log record has *header* information. Depending on the record type, additional columns of information also appear in the output, as explained in Logical-log record types and additional columns.

- Logical-Log Record Header
- Logical-log record types and additional columns
  In addition to the six header columns that display for every record, some record types display additional columns of information. The information that appears varies, depending on record type.
- Log Record Types for Smart Large Objects

Logical-Log Record Header

Table 1 contains sample output to illustrate the header columns that display for a logical-log record.

Table 1. Sample Output from onlog

addr	len	type	xid	id	link
2c018	32	BEGIN	6	3	0
2c038	140	HDELETE	6	0	2c018
2c0c4	64	DELITEM	6	0	2c038
2c104	40	DELITEM	6	0	2c0c4
2c12c	72	HDELETE	6	0	2c104
2c174	44	DELITEM	6	0	2c12c
2c1a0	72	HDELETE	6	0	2c174
2c1e8	44	DELITEM	6	0	2c1a0
2c214	64	HDELETE	6	0	2c1e8
2c254	56	DELITEM	6	0	2c214
2c28c	48	DELITEM	6	0	2c254
2c2bc	24	PERASE	6	0	2c28c
2c2d4	20	BEGCOM	6	0	2c2bc
2c2e8	24	ERASE	6	0	2c2d4
2c300	28	CHFREE	6	0	2c2e8
2c31c	24	COMMIT	6	0	2c300

Table 2 defines the contents of each header column.

Table 2. Definition of onlog Header Columns

Header Field	Contents	Format
addr	Log-record address (log position)	Hexadecimal
len	Record length in bytes	Decimal
type	Record-type name	ASCII
xid	Transaction number	Decimal
id	Logical-log number	Decimal
link	Link to the previous record in the transaction	Hexadecimal

Logical-log record types and additional columns

In addition to the six header columns that display for every record, some record types display additional columns of information. The information that appears varies, depending on record type.

The following table lists all the record types and their additional columns.

The **Action** column indicates the type of database server action that generated the log entry. The **Additional Columns** and **Format** columns describe what information appears for each record type in addition to the header described in Logical-Log Record Header.

Table 1. Logical-Log Record Types

Record Type	Action	Additional Columns and Format
ADDCHK	Add chunk.	• chunk number - Decimal • chunk name - ASCII
ADDDBS	Add dbspace.	• dbspace name - ASCII
ADDITEM	Add item to index.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • logical page - Decimal • key number - Decimal • key length - Decimal
ADDLOG	Add log.	• log number - Decimal • log size (pages) - Decimal • pageno - Hexadecimal

Record Type	Action	Additional Columns and Format
ALLOCGENPG	Allocate a generic page.	• tblspace ID - Decimal • rowid - Decimal • slot flags and length - Decimal • page version if delete - Decimal • flags, vimage record - Decimal • rowid for previous - Decimal • data - ASCII
ALTERDONE	Alter of fragment complete.	• tblspace ID - Hexadecimal • physical page number previous page - Hexadecimal • logical page number - Decimal • version of alter - Decimal
ALTSPCOLSNEW	Changed columns in an alter table.	• number of columns - Decimal • special column list - array
ALTSPCOLSOLD	Changed columns in an alter table.	• number of columns - Decimal • special column list - array
BADIDX	Bad index	• tblspace ID - Hexadecimal
BEGCOM	Begin commit.	• (None) - (None)
BEGIN	Begin work.	• date - Decimal • time - Decimal • SID - Decimal • user - ASCII
BEGPREP	Written by the coordinator database server to record the start of the two-phase commit protocol.	• flags - Decimal (Value is 0 in a distributed transaction.) • number of participants - Decimal
BEGWORK	Begin a transaction.	• begin transaction time - Decimal • user ID - Decimal • process ID - Decimal
BFRMAP	Simple-large-object free-map change.	• tblspace ID - Hexadecimal • bpageno - Hexadecimal • status USED/FREE log ID - Decimal • prev page - Hexadecimal
BLDCL	Build tblspace.	• tblspace ID - Hexadecimal • fextsize - Decimal • nextsize - Decimal • row size - Decimal • ncolumns - Decimal • table name - ASCII
BMAPFULL	Bitmap modified to prepare for alter.	• tblspace ID - Hexadecimal • bitmap page num - Decimal
BMAP2TO4	2-bit bitmap altered to two 4-bit bitmaps.	• tblspace ID - Hexadecimal • 2-bit bitmap page number - Decimal • flags - Decimal
BSPADD	Add blobspace.	• blobspace name - ASCII
BTCPYBCK	Copy back child key to parent.	• tblspace ID - Hexadecimal • parent logical page - Decimal • child logical page - Decimal • slot - Decimal • rowoff - Decimal • key number - Decimal
BTMERGE	Merge B-tree nodes.	• tblspace ID - Hexadecimal • parent logical page - Decimal • left logical page - Decimal • right logical page - Decimal • left slot - Decimal • left rowoff - Decimal • right slot - Decimal • right rowoff - Decimal • key number - Decimal

Record Type	Action	Additional Columns and Format
BTSHUFFL	Shuffle B-tree nodes.	• tblspace ID - Hexadecimal • parent logical page - Decimal • left logical page - Decimal • right logical page - Decimal • left slot - Decimal • left rowoff - Decimal • key number - Decimal • flags - Hexadecimal
BTSPLIT	Split B-tree node.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • parent logical page - Decimal • left logical page - Decimal • right logical page - Decimal • infinity logical page - Decimal • rootleft logical page - Decimal • midsplit - Decimal • key number - Decimal • key length - Decimal
CDINDEX	Create detached index.	• database name - ASCII • owner - ASCII • table name - ASCII • index name - ASCII
CDR	Captures the set of table columns modified by an update statement such as a *bitvector*. This log record allows Enterprise Replication to capture only the changed data to avoid transmitting the unchanged columns to a target site. In the example, the first six columns of the table are unchanged (6 leftmost bits in the **bitvector** are 0), the seventh and eighth columns have been updated (seventh and eighth bits are 1), and so on. The onlog output displays as many bits of bitvector as fit in a single line of the output. To see the entire **bitvector** displayed in hexadecimal, use the **onlog -l** command.	• name of CDR record - ASCII • partition number - Hexadecimal • bitvector - Binary Sample **onlog** output for CDR log record: `adr len type xid id link` `40 36 CDR 14 0 18` `name partno bitvector` `UPDCOLS 10009a 000000110100110100`
CHALLOC	Chunk extent allocation.	• pageno - Hexadecimal • size - Hexadecimal
CHCOMBINE	Chunk extent combine.	• pageno - Hexadecimal
CHFREE	Chunk extent free.	• pageno - Hexadecimal • size - Hexadecimal
CHKADJUP	Update chunk adjunct on disk. The database server writes this record when it moves space from the reserved area to the metadata or user-data area or when the user adds an sbspace chunk.	• chunk number - Integer • ud1_start_page - Integer • ud1_size - Integer • md_start_page - Integer • md_size - Integer • ud2_start_page - Integer • ud2_size - Integer • flags - Hexadecimal
CHPHYLOG	Change physical-log location.	• pageno - Hexadecimal • size in kilobytes - Hexadecimal • dbspace name - ASCII
CHRESERV	Reserve extent for metadata stealing. This record is written when you add an sbspace chunk.	• chunk number - Integer • page number - Integer • length - Integer
CHSPLIT	Chunk extent split.	• pageno - Hexadecimal
CINDEX	Create index.	• tblspace ID - Hexadecimal • low rowid - Decimal • high rowid - Decimal • index descriptor - ASCII

Record Type	Action	Additional Columns and Format
COARSELOCK	Coarse-grain locking	• tblspace ID - Hexadecimal • old coarse-locking flag value - Decimal • new coarse-locking flag value - Decimal
CKPOINT	Checkpoint.	• max users - Decimal • number of active transactions - Decimal
CLR	Compensation-log record; created during a rollback.	• (None) - (None)
CLUSIDX	Create clustered index.	• tblspace ID - Hexadecimal • key number - Decimal
COLREPAI	Adjust BYTE, TEXT, or VARCHAR column.	• tblspace ID - Hexadecimal • number of columns adjusted - Decimal
COMMIT	Commit work.	• date - Decimal • time - Decimal
COMTAB	Compact slot table on a page.	• logical page number - Decimal • number slots moved - Decimal • compressed slot pairs - ASCII
COMWORK	End a transaction and commit work.	• end transaction time - Decimal • begin transaction time - Decimal
DELETE	Delete before-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal
DELITEM	Delete item from index.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • logical page - Decimal • key number - Decimal • key length - Decimal
DERASE	Drop tblspace in down dbspace.	• tblspace number - Hexadecimal • table lock number - Decimal
DFADDEXT	New extent is added.	• partnum - Hexadecimal • offset of extent entry in list - Hexadecimal • extent size in pages - Decimal • physical address of extent - Offset and chunk no-hex
DFDRPEXT	Drop the original extent.	• partnum - Hexadecimal • offset of extent entry in list - Hexadecimal • original size of this extent - Decimal • physical address - offset and chunk no-hex
DFEND	End of defragment operation.	• partnum - Hexadecimal
DFMVPG	Move page from old extent to new extent.	• partnum - Hexadecimal • offset of new extent - Hexadecimal • logical page number of source - Hexadecimal • physical address of destination - Offset and chunk no-hex • physical address of source - Offset and chunk no-hex
DFREMDUM	Remove the dummy entries.	• partnum - Hexadecimal
DFSTART	Start of defragment operation.	• partnum - Hexadecimal
DINDEX	Drop index.	• tblspace ID - Hexadecimal • key number - Decimal
DRPBSP	Drop blobspace.	• blobspace name - ASCII
DRPCHK	Drop chunk.	• chunk number - Decimal • chunk name - ASCII
DRPDBS	Drop dbspace.	• dbspace name ASCII

Record Type	Action	Additional Columns and Format
DRPLOG	Drop log.	• log number - Decimal • log size (pages) - Decimal • pageno - Hexadecimal
ENDTRANS	Written by both the coordinator and participant database servers to record the end of the transaction. ENDTRANS instructs the database server to remove the transaction entry from its shared-memory transaction table and close the transaction. In the coordinator logical log, each BEGPREP that results in a committed transaction is paired with an ENDTRANS record. If the final decision of the coordinator is to roll back the transaction, no ENDTRANS record is written. In the participant logical log, each ENDTRANS record is paired with a corresponding HEURTX record.	• (None) - (None)
ERASE	Drop tblspace.	• tblspace ID -Hexadecimal
FREE_RE	Allocate extent from reserve extent to metadata or user-data area of an sbspace chunk.	• chunk number - Integer • page number - Integer • length - Integer • flag - Hexadecimal
HDELETE	Delete home row.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal
HEURTX	Written by a participant database server to record a heuristic decision to roll back the transaction. It should be associated with a standard ROLLBACK record indicating that the transaction was rolled back.	• flag - Hexadecimal (Value is always 1.)
HINSERT	Home row insert.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal
HUPAFT	Home row update, after-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal
HUPBEF	Home row update, before-image. In addition, the flag field of the HUPBEF header may include the following values: LM_PREVLSN Confirms that an LSN exists. LM_FIRSTUPD Confirms that this is the first update for this rowID by this transaction.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal • LSN (optional) - Decimal
HUPDATE	If the home row update before-images and after-images can both fit into a single page, the database server writes a single HUPDATE record. In addition, the flag field of the HUPDATE log may include the following values: LM_PREVLSN Confirms that an LSN exists. LM_FIRSTUPD Confirms that this is the first update for this rowID by this transaction.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • forward ptr rowid - Hexadecimal • old slotlen - Decimal • new slotlen - Decimal • number of pieces - Decimal • LSN (optional) - Decimal
IDXFLAGS	Index flags.	• tblspace ID - Hexadecimal • key number - Hexadecimal
INSERT	Insert after-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal
ISOSPCOMMIT	Log an isolated save-point commit.	• end transaction time - Decimal • begin transaction time - Decimal

Record Type	Action	Additional Columns and Format
LCKLVL	Locking mode (page or row).	• tblspace ID - Hexadecimal • old lockmode - Hexadecimal • new lockmode - Hexadecimal
LG_ADDBPOOL	Add a buffer pool online.	• page size in bytes - Decimal • number of buffers in the pool - Decimal • number of lru queues - Decimal • percent of lru_max_dirty - Decimal • percent of lru_min_dirty - Decimal
PTRUNCATE	Identifies an intention to truncate a table. The partitions are marked to be dropped or reused, according to the specified command option.	• tblspace ID - Hexadecimal
TRUNCATE	TRUNCATE has freed the extents and the transaction will be committed.	• tblspace ID - Hexadecimal
MVIDXND	Index node moved to allow for 2-bit to 4-bit bitmap conversion.	• tblspace ID - Hexadecimal • old page number - Decimal • new page number - Decimal • parent page number - Decimal • parent slot number - Decimal • parent slot offset - Decimal • key number - Decimal
PBDELETE	Delete tblspace blobpage.	• bpageno - Hexadecimal • status USED/FREE unique ID - Decimal
PBINSERT	Insert tblspace blobpage.	• bpageno - Hexadecimal • tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal • pbrowid - Hexadecimal
PDINDEX	Predrop index.	• tblspace ID - Hexadecimal
PGALTER	Page altered in place.	• tblspace ID - Hexadecimal • physical page number - Hexadecimal
PGMODE	Page mode modified in bitmap.	• tblspace ID - Hexadecimal • logical page number - Decimal • old mode - Hexadecimal • new mode - Hexadecimal
PERASE	Preerase old file. Mark a table that is to be dropped. The database server frees the space on the commit.	• tblspace ID - Hexadecimal
PNGPALIGN8	Use the pages in this tblspace as generic pages.	• None
PNLOCKID	Change tblspaces lockid.	• tblspace ID - Hexadecimal • old lock ID - Hexadecimal • new lock ID - Hexadecimal
PNSIZES	Set tblspace extent sizes.	• tblspace ID - Hexadecimal • fextsize - Decimal • nextsize - Decimal
PREPARE	Written by a participant database server to record the ability of the participant to commit the transaction, if so instructed.	• DBSERVERNAME of coordinator - ASCII
PTADESC	Add alter description information.	• tblspace ID - Hexadecimal • physical page number of previous page - Hexadecimal • logical page number - Decimal • number of columns added - Decimal
PTALTER	Alter of fragment begun.	• tblspace ID - Hexadecimal • physical page number previous page - Hexadecimal • logical page number - Decimal • alter desc page number - Decimal • num columns added - Decimal • version of alter - Decimal • added rowsize - Decimal

Record Type	Action	Additional Columns and Format
PTALTNEWKEYD	Update key descriptors in a tblspace header after an alter table command.	• bytes in key descriptor - Decimal • data in key descriptor - ASCII
PTALTOLDKEYD	Update key descriptors after an alter table command.	• bytes in key descriptor - Decimal • data in key descriptor - ASCII
PTCOLUMN	Add special columns to fragment.	• tblspace ID - Hexadecimal • number of columns - Decimal
PTEXTEND	Tblspace extend.	• tblspace ID - Hexadecimal • last logical page - Decimal • first physical page - Hexadecimal
PTRENAME	Rename table.	• tblspace ID - Hexadecimal • old table name - ASCII • new table name - ASCII
RDELETE	Remainder page delete.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal • hrowid (optional) - Decimal • poffset (optional) - Decimal
RENDBS	Rename dbspace.	• new dbspace name - ASCII
REVERT	Logs the reversion of a database space to a database space of an earlier version.	• type of reversion event - Decimal • arg1 - Decimal • arg2 - Decimal • arg3 - Decimal
RINSERT	Remainder page insert.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal • hrowid (optional) - Decimal • poffset (optional) - Decimal
ROLLBACK	Rollback work.	• date - Decimal • time - Decimal
ROLWORK	End a transaction and roll back work.	• end transaction time - Decimal • begin transaction time - Decimal
RSVEXTEND	Logs the extension to the reserved pages.	• number of pages - Decimal • physical page number of extent - Hexadecimal
RTREE	Logs inserts and deletions for R-tree index pages. (Other operations on R-tree indexes are physically logged.) The record subtypes are: • LEAFINS - insert item in a leaf page • LEAFDEL - delete item from leaf page	• record subtype - ASCII • index page rowid - Hexadecimal • tuple length - Decimal • base table rowid - Decimal • base table fragid - Decimal • delete flag - Decimal
RUPAFT	Remainder page update, after-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal
RUPBEF	Remainder page update, before-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • slotlen - Decimal • hrowid (optional) - Decimal • poffset (optional) - Decimal
RUPDATE	If the remainder page update before-images and after-images can both fit into a single page, the database server writes a single RUPDATE record.	• tblspace ID - Hexadecimal • rowid - Hexadecimal • forward ptr rowid - Hexadecimal • old slotlen - Decimal • new slotlen - Decimal • number of pieces - Decimal • hrowid (optional) - Decimal • poffset (optional) - Decimal

Record Type	Action	Additional Columns and Format
SBLOB	Indicates a subsystem log record for a smart large object. The various record subtypes are: • CHALLOC • CHCOMBINE • CHFREE • CHSPLIT • CREATE • DELETES • EXTEND • HDRUPD • PDELETE • PTRUNC • REFCOUNT • UDINSERT • UDINSERT_LT • UDUPAFT • UDUPAFT_LT • UDUPAFT • UDUPAFT_LT • UDWRITE • UDWRITE_LT	• Varies For more information, see Log Record Types for Smart Large Objects. Varies
SYNC	Written to a logical-log file if that log file is empty and administrator instructs the database server to switch to next log file.	• (None) - (None)
TABLOCKS	Written by either a coordinator or a participant database server. It is associated with either a BEGPREP or a PREPARE record and contains a list of the locked tblspaces (by tblspace number) held by the transaction. (In a distributed transaction, transactions are shown as the owners of locks.)	• number of locks - Decimal • tblspace number - Hexadecimal
UDINSERT	Append new user data.	• number of locks - Decimal • tblspace number - Hexadecimal
UDUPAFT	Update user data after-image if a UDWRITE is too expensive.	• chunk - Decimal • page within chunk - Hexadecimal • offset within page - Hexadecimal • data length - Hexadecimal
UDUPBEF	Update user-data before-image if a UDWRITE is too expensive.	• chunk - Decimal • page within chunk- Hexadecimal • offset within page - Hexadecimal • data length - Hexadecimal
UDWRITE	Update user data (difference image).	• chunk - Decimal • page within chunk - Hexadecimal • offset within chunk - Hexadecimal • length before write - Hexadecimal • length after write - Hexadecimal
UNDO	Header record to a series of transactions to be rolled back.	• count - Decimal
UNDOBLDC	This record is written if a CREATE TABLE statement should be rolled back but cannot be because the relevant chunk is down. When the log file is replayed, the table will be dropped.	• tblspace number - Hexadecimal
UNIQID	Logged when a new SERIAL value is assigned to a row.	• tblspace ID - Hexadecimal • unique ID - Decimal
UNIQ8ID	Logged when a new SERIAL8 value is assigned to a row.	• tblspace ID - Hexadecimal • unique ID - Decimal
UPDAFT	Update after-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal
UPDBEF	Update before-image.	• tblspace ID - Hexadecimal • rowid - Hexadecimal

Record Type	Action	Additional Columns and Format
XAPREPARE	Participant can commit this XA transaction.	• (None) - (None)

Log Record Types for Smart Large Objects

All smart-large-object log records are the SBLOB type. Each smart-large-object log record contains six header columns, described in Logical-Log Record Header; the record subtype; and additional information. The information that appears varies, depending on record subtype.

Table 1 lists all the smart-large-object record types. The **Subtype** column describes the smart-large-object record type. The **Action** column indicates the type of database server action that generated the log entry. The **Additional Columns** and **Format** columns describe what information appears for each record type.

Table 1. Record Subtypes for Smart Large Objects

Record Subtype	Action	Additional Columns	Format
CHALLOC	Allocate chunk extent.	extent [chk, page, len]	Decimal
		flags	Hexadecimal
CHCOMBINE	Combine two pages in the user-data extent list.	chunk number	Decimal
		first page	Decimal
		second page	Decimal
CHFREE	Frees chunk extent.	extent [chk, page, len]	Decimal
CHSPLIT	Split a page in the user-data extent list.	chunk number	Decimal
		UDFET page to split	Decimal
CREATE	Create smart large object.	smart-large-object ID [sbs, chk, page, oid]	Decimal
		number of extents in lomaphdr	Decimal
DELETE	Delete a smart large object at commit.	smart-large-object ID [sbs, chk, page, oid]	Decimal
EXTEND	Add extent to an extent list of a smart large object.	smart-large-object ID [sbs, chk, page, oid]	Decimal
		extent [chk, page, len]	Decimal
		lomap overflow page number	Decimal
HDRUPD	Update smart-large-object header page.	smart-large-object ID [sbs, chk, page, oid]	Decimal
		old EOF offset	String
		new EOF offset	String
		old times	Decimal
		new times	Decimal
PDELETE	Queue a smart large object for deletion at commit.	smart-large-object ID [sbs, chk, page, oid]	Decimal
PTRUNC	Queue a smart large object for truncation at commit.	smart-large-object ID [sbs, chk, page, oid]	Decimal
		old offset	String
		new offset	String
REFCOUNT	Increment or decrement the reference count of a smart large object.	smart-large-object ID [sbs, chk, page, oid]	Decimal
		1 if increment; 0 if decrement	Decimal
UDINSERT,	Append new user data.	chunk	Decimal
UDINSERT_LT		page within chunk	Decimal
		offset within page	Decimal
		data length	Decimal
UDUPAFT,	Update user-data after-image if a UDWRITE is too expensive.	chunk	Decimal
UDUPAFT_LT		page within chunk	Decimal
		offset within page	Decimal
		data length	Decimal
UDUPBEF,	Update user-data beforeimage if a UDWRITE is too expensive.	chunk	Decimal
UDUPBEF_LT		page within chunk	Decimal
		offset within page	Decimal
		data length	Decimal
UDWRITE,	Update user data (difference image).	chunk	Decimal
UDWRITE_LT		page within chunk	Decimal
		offset within page	Decimal

Record Subtype	Action	Additional Columns	Format
		length before write	Decimal
		length after write	Decimal
		number of different image pieces	Decimal

For an example of smart-large-object records in **onlog** output, see smart-large-object log records in the chapter on what is the logical log in the *IBM® Informix® Administrator's Guide*.

Figure 1 shows an example of smart-large-object records in **onlog** output. The first two records show that an extent was freed. The next group of records, flanked by BEGIN and COMMIT, shows the allocation of storage and creation of the smart large objects.

Figure 1. Smart-Large-Object Records in onlog Output

```
addr    len   type    xid       id  link     subtype   specific-info

4e8428   40   SBLOB     8        0  4e7400   CHFREE    (2,53,421)
4e8450   40   SBLOB     8        0  4e8428   CHFREE    (2,579,421)

c8018    40   BEGIN     8        3  0        07/13/98 10:23:04 34       informix
c8040    264  SBLOB     8        0  c8018    CREATE    [2,2,1,900350517]  10
c8148    44   SBLOB     8        0  c8040    CHALLOC   (2,53,8) 0x1
c8174    68   SBLOB     8        0  c8148    EXTEND    [2,2,1,900350517] (2,53,8) -1
c81b8    264  SBLOB     8        0  c8174    CREATE    [2,2,2,900350518]  10
c82c0    44   SBLOB     8        0  c81b8    CHALLOC   (2,61,1) 0x1
c82ec    68   SBLOB     8        0  c82c0    EXTEND    [2,2,2,900350518] (2,61,1) -1
c8330    56   SBLOB     8        0  c82ec    REFCOUNT  [2,2,1,900350517] 1
c8368    56   SBLOB     8        0  c8330    REFCOUNT  [2,2,2,900350518] 1
c83a0    36   COMMIT    8        0  c8368    07/13/98 10:23:05

c83c4    40   BEGIN     8        3  0        07/13/98 10:23:05 34       informix
c83ec    264  SBLOB     8        0  c83c4    CREATE    [2,2,3,900350519]  10
c84f4    44   SBLOB     8        0  c83ec    CHALLOC   (2,62,1) 0x1
c8520    68   SBLOB     8        0  c84f4    EXTEND    [2,2,3,900350519] (2,62,1) -1
c8564    56   SBLOB     8        0  c8520    REFCOUNT  [2,2,3,900350519] 1
c859c    36   COMMIT    8        0  c8564    07/13/98 10:23:05
```


Administrative Utilities

- Overview of Utilities
  The Informix database server utilities allow you to perform administrative tasks directly from the command line.
- The finderr utility
  Use the **finderr** utility to view additional information on Informix error messages. On UNIX and Linux platforms, the information appears on the command line. On Windows platforms, the information appears in the Error Messages program.
- The genoncfg Utility
  Use the **genoncfg** utility to expedite the process of customizing the default Informix configuration file (**onconfig.std**) to the host environment and your planned usage of a database server instance.
- The oncheck Utility
  Use the **oncheck** utility to check specified disk structures for inconsistencies, repair inconsistent index structures, and display information about disk structures.
- The onclean utility
  Use the **onclean** utility to force a shut down of the database server when normal shut down with the **onmode** utility fails or when you cannot restart the server. The **onclean** utility attempts to clean up shared memory, semaphores, and stops database server virtual processes.
- The oncmsm utility
  Use the **oncmsm** utility to start or shut down a Connection Manager, load a new configuration file into a Connection Manager to modify the Connection Manager's settings, or update the format of a configuration file.
- The onconfig_diff utility
  Use the **onconfig_diff** utility to compare two onconfig files.
- The ondblog utility
  Use the **ondblog** utility to change the logging mode for one or more databases.
- The oninit utility
  The **oninit** utility starts the database server.
- The onkstash Utility
  Use the **onkstash** utility to create a password stash file for an existing PKCS#12 keystore.
- The onkstore Utility
- The onlog utility
  The **onlog** utility displays the contents of a logical-log file, either on disk or on backup.
- The onmode utility
  Use the **onmode** utility to change the database server operating mode and perform various other operations on shared memory, sessions, transactions, parameters, and segments.
- The onparams Utility
  Use the **onparams** utility to add or drop a logical-log file, change physical-log parameters, and add a new buffer pool.
- The onpassword utility
  Use the **onpassword** utility to encrypt and decrypt a password file. Connection Manager and Enterprise Replication utilities require a password file to connect to database servers over an untrusted network.
- The ifxclone utility
  You use the **ifxclone** utility to create a server clone from a snapshot of an existing database server.
- The onspaces utility
  Use the **onspaces** utility to manage the storage spaces in your database.
- The onstat utility
  The **onstat** utility reads shared-memory structures and provides statistics about the database server at the time that the command runs.

Overview of Utilities

The Informix® database server utilities allow you to perform administrative tasks directly from the command line.

For a complete list of server utilities, see Informix administrative utilities and applications.

The database server utilities support multibyte command-line arguments. For a complete list of the utilities that support multibyte command-line arguments, see the Locale-specific support for utilities.

The database server must be online before you execute a utility, with the following exceptions:

- **oninit**
- Some **onlog** options
- Some **oncheck** options

Note: When using utilities, do not use the UNIX command `CTRL-C` to send an interrupt signal to a process because it might produce an error.

- Obtaining utility version information
  Use the **-V** and **-version** options of many Informix command-line utilities to obtain version, primarily for debugging.
- Setting local environment variables for utilities
  On UNIX operating systems, you can start certain utilities without setting local environment variables in your shell environment. You can set local environment variables in the onconfig file. When you run the command to start the utility, use the **-FILE** option to point to the onconfig file.

Obtaining utility version information

Use the **-V** and **-version** options of many Informix® command-line utilities to obtain version, primarily for debugging.

The **-V** option displays the software version number and the serial number.

The **-version** option extends the **-V** option to display additional information about the build operation system, build number, and build date.

```
>>-utility--+-utility specific options-+----------------------><
            +- -V---------------------+
            '- -version----------------'
```

The **-V** and **-version** options cannot be used with any other utility options. For example, the **onstat -version** command might display the following output.

```
onstat -version
```

```
        Program:          onstat
    Build Version:    11.70.FC1
    Build Host:       connla
    Build OS:         SunOS 5.6
    Build Number:     009
    Build Date:       Sat Nov 20 03:38:27 CDT 2011
    GLS Version:      glslib-4.50.xC2
```

The **onstat -V** command might display the following information:

```
IBM Informix Version 11.70.FC1   Software Serial Number
 RDS#N000000
```


Setting local environment variables for utilities

On UNIX operating systems, you can start certain utilities without setting local environment variables in your shell environment. You can set local environment variables in the onconfig file. When you run the command to start the utility, use the **-FILE** option to point to the onconfig file.

Before you begin, ensure that these prerequisites are met:

- The path to the executable program for the utility is part of the existing shell environment.
- If you want to run commands on a remote computer, a remote shell utility such as SSH is configured.

1. Add values for one or more environment variables to the onconfig file. Use the following format for each directive: #$*variable_name value*
2. When you run the command to start the utility, use the **-FILE** option to specify the full or relative path to the onconfig file. Review the syntax, usage, and examples in the reference information for the **-FILE** option.

The utility reads and sets the environment variables that are specified in the onconfig file, and those values take precedence over values that are set in the local shell environment.

Related concepts:
onconfig file

Related reference:
Database server files
The -FILE option

The finderr utility

Use the **finderr** utility to view additional information on Informix® error messages. On UNIX and Linux platforms, the information appears on the command line. On Windows platforms, the information appears in the Error Messages program.

Syntax

```
            .- - .
>>-finderr--+----+--error_number------------------------------><
            '- + '
```

Table 1. finderr element

Element	Purpose	Key Considerations
error_number	The error message number for which to provide additional information	**On UNIX or Linux:** If you do not include a minus sign (-) or plus sign (+) and both a positive and a negative version of the error message exists, the negative version of the message is displayed. To display the information about an error message number that is positive, preface the error number with a plus sign. **On Windows:** If you do not include a minus sign or plus sign and both a positive and a negative version of the error message exists, you must choose which message you want to view in the Error Messages program.

Usage

Error messages that are printed in the message log include a message number and a short message description. Use the message number with the **finderr** command to look up a more detailed description of the cause of the error and possible user actions to correct or prevent the error.

On Windows, you can open the Error Messages program directly by choosing Error Messages from the database server program group.

Examples

The following command on a UNIX or Linux platform displays information about the error message -201:

```
finderr 201

-201     A syntax error has occurred.

This general error message indicates mistakes in the form of an SQL
statement. Look for missing or extra punctuation (such as missing or
extra commas, omission of parentheses around a subquery, and so on),
keywords misspelled (such as VALEUS for VALUES), keywords misused (such
as SET in an INSERT statement or INTO in a subquery), keywords out of
sequence (such as a condition of "value IS NOT" instead of "NOT value
IS"), or a reserved word used as an identifier.

Database servers that provide full NIST compliance do not reserve any
words; queries that work with these database servers might fail and
return error -201 when they are used with earlier versions of IBM Informix
database servers.

The cause of this error might be an attempt to use round-robin syntax with
CREATE INDEX or ALTER FRAGMENT INIT on an index. You cannot use round-robin
indexes.

The error may also occur if an SQL statement uses double quotation marks
around input strings and the environment variable DELIMIDENT is set.
If DELIMIDENT is set, strings that are surrounded by double quotation
marks are regarded as SQL identifiers rather than string literals. For
more information on the usage of DELIMIDENT, see the IBM Informix Guide to
SQL: Reference.
```

The following command displays information about the error message 100, which corresponds to the SQLCODE value of 100:

```
finderr +100

100     No matching records found.

The database server did not find any more data. This message is an ANSI-
standard SQLCODE value. If you attempted to select or fetch data, you
encountered the end of the data, or no data matched the criteria in the
WHERE clause. Check for an empty table. Use this SQLCODE value to determine
when a statement reaches the end of the data. For more information, see the
discussion of SQLCODE in the IBM Informix ESQL/C Programmer's Manual. The
database server can return this SQLCODE value to a running program.
For the High-Performance Loader (HPL), this message can indicate that the
map might be from a project other than the default project. Use the -p option
in the onpload command line to provide a project name for mappings.
```

The genoncfg Utility

Use the **genoncfg** utility to expedite the process of customizing the default Informix® configuration file (**onconfig.std**) to the host environment and your planned usage of a database server instance.

Syntax

```
>>-genoncfg--+-input_file--+-------------+-+-------------------><
             |             '-informixdir-' |
             +- -h-------------------------+
             +- -V-------------------------+
             '- -version-------------------'
```

Element	Purpose	Key Considerations
input_file	Name of the input file containing your parameter settings.	
informixdir	Path to the Informix installation that you want to configure.	You can omit the installation path if the **INFORMIXDIR** environment variable is set. If the **INFORMIXDIR** variable is already set and you enter an installation path on the command line, the utility runs with the command-line path.
-h	Help information about the **genoncfg** utility.	
-V	Displays short version information and exits the command-line utility.	
-version	Displays extended version information and exits the command-line utility.	

Usage

Log in to the host computer as root or user **informix** before you run this utility.

You must set parameters that are valid for your host environment in an input file before you can successfully run the **genoncfg** utility. For all environments, the parameter `disk` is required in the input file. You can also enter directives in the input file. The directives are not required to run the utility, but they can be helpful in some circumstances.

The utility does not read or modify any existing configuration file. If you have a pre-existing ONCONFIG file in the host environment, none of its parameter values are changed when you run the utility. Therefore, you can review the recommended configuration settings before you put them in effect on a database server instance.

To use the genoncfg utility:

1. Create the input file containing your values for the parameters that the **genoncfg** utility processes with a text editor.
2. Run the utility with your input file. The configuration file (named **onconfig**) is generated and saved in the working directory.
3. *Optional:* Rename the generated configuration file.
4. If you want to run a database server instance with the generated configuration file, copy the file to `$INFORMIXDIR/etc` and update the **ONCONFIG** environment variable accordingly.

Input File for the genoncfg Utility

Use the input file to specify the following information about the database server instance:

- number of anticipated online transaction processing (OLTP) connections
- number of anticipated decision-support systems (DSS) connections
- disk space
- CPU utilization
- network connection settings
- recovery time

The input file is an ASCII text file. There is no required order for the parameters. The following is an example of an input file:

```
cpus 1
memory 1024 m
connection name demo_on onsoctcp 9088
servernum 1
oltp_connections 10
dss_connections 2
disk /opt/IBM/informix/demo/server/online_root 0 k 300 m
directive one_crit
directive debug
```

Table 1. Parameters of the Input File for the **genoncfg** Utility

Parameter	Description

Parameter	Description
connection	Server connection parameters: • `name` or `alias`, depending on whether the connection functions with a specific server name (the DBSERVERNAME parameter of the configuration file) or with an alternative server name (using the DBSERVERALIASES parameter of the configuration file) • name for the connection • type of server connection (equivalent to NETTYPE in the configuration file) • port number for the service Example: `connection name demo_on onsoctcp 9088`
cpus	Number of central processing units (CPUs) to allocate the instance. Example: `cpus 1`
directive	Directives that can be used with the **genoncfg** utility. • one_crit: Configures the database server to store physical logs, logical logs, and data in the root dbspace only. • debug: Displays information in real time about the host environment and actions done on the configuration file. Example: `directive one_crit` This information can be helpful in troubleshooting problems with database server configuration. One scenario is that the debug directive can result in saving time. In this scenario, you read the displayed information and notice that the utility is creating an onconfig file that you do not want or that will not function. You stop the utility while it is still running, adjust the input file settings, and then rerun the utility with the modified input file.
disk	Disk storage space settings for the instance: • location of the root dbspace • size of offset, in megabytes (m) or kilobytes (k) • size of root dbspace, in megabytes (m) or kilobytes (k) Example: UNIX: `/opt/IBM/dbspace/rootdbs` Windows: `d:\INFXDATA\rootdbs` Important: If you enter a path location that is the root dbspace of a working instance, the instance is overwritten and made unusable.
dss_connections	Estimated number of decision-support systems (DSS) connections to the instance. For example, a query client or other application that obtains result sets for business intelligence can be a DSS connection. Example: `dss_connections 2`
memory	Amount of memory, in megabytes (m), for the instance. Example: `memory 1024 m`
oltp_connections	Estimated number of online transaction processing (OLTP) connections to the instance. Typically, an application that modifies the state of databases in the instance is an OLTP connection. Example: `oltp_connections 10`
rto_server_restart	Specifies the amount of time, in seconds, that the database server has to recover from a problem after you restart Informix and bring it into online or quiescent mode. The value can be set either to 0 to disable the configuration parameter or to a value between 60 and 1800 to enable the parameter and indicate the number of seconds. Example: `rto_server_restart 100` specifies the recovery time objective as 100 seconds.
servernum	Unique ID of the database server instance. Example: `servernum 1`

Related tasks:
Modifying the onconfig file

The oncheck Utility

Use the **oncheck** utility to check specified disk structures for inconsistencies, repair inconsistent index structures, and display information about disk structures.

The **oncheck** utility requires sort space when examining an index. The amount of sort space required is the same as that needed to build the index. For information about calculating the amount of temporary space needed, see Estimating temporary space for index builds. If you receive the error "`no free disk space for sort`," you must estimate the amount of temporary space needed and make that space available.

You can use SQL administration API commands that are equivalent to some **oncheck** commands.

- oncheck Check-and-Repair
  The **oncheck** utility repairs disk structures.
- oncheck utility syntax
  The **oncheck** utility checks specified disk structures for inconsistencies, repairs inconsistent index structures, and displays information about disk structures.
- oncheck -cc and-pc: Check system catalog tables
  The **-cc** option checks system catalog tables for information about database tables, columns, indexes, views, constraints, stored procedures, and privileges.
- oncheck -cd and oncheck -cD commands: Check pages
  Use the **oncheck -cd** and **oncheck -cD** commands to check each page for consistency. Use the **oncheck -cd -y** or **oncheck -cD -y** command to repair inconsistencies.
- oncheck -ce, -pe: Check the chunk-free list
- oncheck -ci and -cI: Check index node links
  Use the **oncheck -ci** and **oncheck -cI** commands to check the ordering of key values and the consistency of horizontal and vertical node links for all indexes associated with the specified table.
- oncheck -cr and -cR: Check reserved pages
- oncheck -cs, -cS, -ps, -pS: Check and display sbspaces
- oncheck -pB: Display blobspace statistics

- oncheck -pd and pD: Display rows in hexadecimal format
- oncheck -pk, -pK, -pl, -pL: Display index information
- oncheck -pp and -pP: Display the contents of a logical page
- oncheck -pr and pR: Display reserved-page information
  The -pr option performs the same checks as **oncheck -cr** command and displays the reserved-page information.
- oncheck -pt and -pT: Display tblspaces for a Table or Fragment
  The **oncheck -pt** and **oncheck -pT** options print a tblspace report for a specific table or fragment. The only difference between these options is that **oncheck -pT** prints more information, including some index-specific information.
- Turn On Locking with -x
- Send Special Arguments to the Access Method with -u
- Return Codes on Exit

oncheck Check-and-Repair

The **oncheck** utility repairs disk structures.

The **oncheck** utility can repair the following types of disk structures:

- Partition page statistics
- Bitmap pages
- Partition blobpages
- Blobspace blobpages
- Indexes
- Sbspace pages
- Metadata partitions for sbspaces

If **oncheck** detects inconsistencies in other structures, messages alert you to these inconsistencies, but **oncheck** cannot resolve the problem. For more information, see the chapter on consistency checking in the *IBM® Informix® Administrator's Guide* and Disk Structures and Storage.

- What Does Each Option Do?
  The **oncheck** options fall into three categories: check, repair, and display.
- Using the -y Option to Perform Repairs
  Use the **-y** option to instruct **oncheck** to perform repairs automatically.
- Repairing Indexes in Sbspaces and External Spaces
  The **oncheck** utility can repair an index in an sbspace or external space if the index is created using an access method that supports the **oncheck -y** option.
- Locking and oncheck
  The **oncheck** utility places a shared lock on a table, so no other users can perform updates, inserts, or deletes until the check has completed.

What Does Each Option Do?

The **oncheck** options fall into three categories: check, repair, and display.

The display or print options (those prefixed with the letter **p**) are identical in function to the **-c** options, except that the **-p** options display additional information about the data that is being checked as the **oncheck** utility executes. You cannot combine **oncheck** option flags except as the following paragraphs describe.

In general, the **-c** options check for consistency and display a message on the screen only if they find an error or inconsistency.

Any user can execute the check options. On UNIX platforms, you must be user **informix** or **root** to display database data or initiate repair options. On Windows, you must be a member of the **Informix-Admin** group to display database data or initiate repair options.

Table 1 associates **oncheck** options with their function. It also shows the SQL administration API *command* strings that are equivalent to the **oncheck -c** options.

Table 1. oncheck Options and Their Function

Object	**Check**	**SQL administration API *command* string**	**Repair**	**Display**
Blobspace simple large objects				**-pB**
System catalog tables	**-cc**			**-pc**
Data rows, no simple large objects or smart large objects	**-cd**			**-pd**
Data rows, simple large objects but no smart large objects	**-cD**			**-pD**
Table with a user-defined access method	**-cd, -cD**	CHECK DATA		
Chunks and extents	**-ce**	CHECK EXTENTS		**-pe**
Index (key values)	**-ci, -cix**		**-ci -y -pk -y, -pkx -y**	**-pk**
Index (keys plus rowids)	**-cI, -cIx**		**-cI -y -pK -y, -pKx -y**	**-pK**
Index with a user-defined access method	**-ci, -cI**			
Index (leaf key values)			**-pl -y, -plx -y**	**-pl**
Index (leaf keys plus rowids)			**-pL -y, -pLx -y**	**-pL**

Object	Check	SQL administration API *command* string	Repair	Display
Pages (by table or fragment)				**-pp**
Pages (by chunk)				**-pP**
Root reserved pages	**-cr, -cR**			**-pr, -pR**
Metadata for smart large objects	**-cs, -cS**			**-ps, -pS**
Space usage (by table or fragment)		CHECK PARTITION PRINT PARTITION		**-pt**
Space usage (by table, with indexes)				**-pT**

Using the -y Option to Perform Repairs

Use the **-y** option to instruct **oncheck** to perform repairs automatically.

If you do not use the **-y** option, **oncheck** prompts you when it encounters an inconsistency and allows you to request a repair. If you specify option **-n**, **oncheck** does not prompt you because this option instructs **oncheck** to not perform repairs.

The following examples show automatic repair commands for the **oncheck** utility:

```
oncheck -cd -y
oncheck -cD -y
oncheck -ci -y
oncheck -cI -y
```


Repairing Indexes in Sbspaces and External Spaces

The **oncheck** utility can repair an index in an sbspace or external space if the index is created using an access method that supports the **oncheck -y** option.

Although the **oncheck** utility does not repair fragmented indexes, user-defined access methods can repair them. For more information about the **oncheck** options that access methods support, see the *IBM® Informix® DataBlade API Programmer's Guide* or the *IBM Informix Virtual-Index Interface Programmer's Guide*.

Locking and oncheck

The **oncheck** utility places a shared lock on a table, so no other users can perform updates, inserts, or deletes until the check has completed.

The **oncheck** utility places a shared lock on a table during the following operations:

- When it checks data
- When it checks indexes (with **-ci**, **-cI**, **-pk**, **-pK**, **-pl**, or **-pL**) and the table uses page locking
- When you specify the **-x** option with **-ci**, **-cI**, **-pk**, **-pK**, **-pl**, or **-pL** and the table uses row locking

If the table does not use page locking, the database server does not place a shared lock on the table when you check an index with the **oncheck -ci**, **-cI**, **-pk**, **-pK**, **-pl**, or **-pL** options. When no shared lock is on the table during an index check, other users can update rows during the check.

By not placing a shared lock on tables using row locks during index checks, the **oncheck** utility cannot be as accurate in the index check. For absolute assurance of a complete index check, you can execute **oncheck** with the **-x** option. With the **-x** option, **oncheck** places a shared lock on the table, and no other users can perform updates, inserts, or deletes until the check has completed.

The **oncheck** utility returns unreliable results when run on secondary servers in a high-availability cluster.

For more information about the **-x** option, refer to Turn On Locking with -x. For information on shared locks and intent shared locks, see the *IBM® Informix® Performance Guide*.

The **oncheck** utility places a shared lock on system catalog tables when they are checked. It places an exclusive lock on a table when it executes repair options.

oncheck utility syntax

The **oncheck** utility checks specified disk structures for inconsistencies, repairs inconsistent index structures, and displays information about disk structures.

```
>>-oncheck--+------------------------+------------------------>
```

Notes:

1. See The -FILE option.

Element	Purpose	Key Considerations
-cc	Checks system catalog tables for the specified database	See oncheck -cc and-pc: Check system catalog tables.
-cd	Reads all pages except simple large objects from the tblspace for the specified database, table, or fragment and checks each page for consistency Also checks tables that use a user-defined access method	Does not check simple or smart large objects. See oncheck -cd and oncheck -cD commands: Check pages.
-cD	Same as **-cd** but also reads the header of each blobpage and checks it for consistency	Checks simple large objects but not smart large objects. See oncheck -cd and oncheck -cD commands: Check pages .
-ce	Checks each chunk-free list and corresponding free space and each tblspace extent. Also checks smart-large-object extents and sbspace metadata	The **oncheck** process verifies that the extents on disk correspond to the current control information that describes them. See oncheck -ce, -pe: Check the chunk-free list. For background information, see Next-Extent Allocation.
-ci	Checks the ordering of key values and the consistency of horizontal and vertical node links for all indexes associated with the specified table Also checks indexes that use a user-defined access method	See oncheck -ci and -cI: Check index node links.
-cI	Same as **-ci** but also checks that the key value tied to a rowid in an index is the same as the key value in the row	See oncheck -ci and -cI: Check index node links.
-cr	Checks each of the root dbspace reserved pages for several conditions	See oncheck -cr and -cR: Check reserved pages.
-cR	Checks the root dbspace reserved pages, physical-log pages, and logical-log pages	See oncheck -cr and -cR: Check reserved pages
-cs	Checks smart large object and sbspace metadata for an sbspace	See oncheck -cs, -cS, -ps, -pS: Check and display sbspaces.
-cS	Checks smart large object and sbspace metadata for an sbspace as well as extents	See oncheck -cs, -cS, -ps, -pS: Check and display sbspaces.

Element	Purpose	Key Considerations
sbspace	Indicates optional sbspace name If not supplied, all sbspaces are checked.	None.
-n	Indicates that no index repair should be performed, even if errors are detected	Use with the index repair options (**-ci**, **-cI**, **-pk**, **-pK**, **-pl**, and **-pL**).
-pB	Displays statistics that describe the average fullness of blobspace blobpages in a specified table	These statistics provide a measure of storage efficiency for individual simple large objects in a database or table. If a table or fragment is not specified, statistics are displayed for the entire database. See oncheck -pB: Display blobspace statistics. For information about optimizing blobspace blobpage size, see the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-pc	Same as **-cc** but also displays the system catalog information as it checks the system catalog tables, including extent use for each table	None.
-pd	Displays rows in hexadecimal format	See oncheck -pd and pD: Display rows in hexadecimal format.
-pD	Displays rows in hexadecimal format and simple-large-object values stored in the tblspace or header information for smart large objects stored in an sbspace sbpage and simple large objects stored in a blobspace blobpage	See oncheck -pd and pD: Display rows in hexadecimal format.
-pe	Same as **-ce** but also displays the chunk and tblspace extent information as it checks the chunk free list, the corresponding free space, and each tblspace extent	See oncheck -ce, -pe: Check the chunk-free list.
-pk	Same as **-ci** but also displays the key values for all indexes on the specified table as it checks them	See oncheck -pk, -pK, -pI, -pL: Display index information.
-pK	Same as **-cI** but also displays the key values and rowids as it checks them	See oncheck -pk, -pK, -pI, -pL: Display index information.
-pl	Same as **-ci** but also displays the key values. Only leaf-node index pages are checked	See oncheck -pk, -pK, -pI, -pL: Display index information.
-pL	Same as **-cI** but also displays the key values and rowids for leaf-node index pages only	See oncheck -pk, -pK, -pI, -pL: Display index information.
-pp	Displays contents of a logical page	See oncheck -pp and -pP: Display the contents of a logical page.
-pP	Same as **-pp** but requires a chunk number and logical page number or internal rowid as input	See oncheck -pp and -pP: Display the contents of a logical page.
-pr	Same as **-cr** but also displays the reserved-page information as it checks the reserved pages	See oncheck -pr and pR: Display reserved-page information.
-pR	Same as **-cR** but also displays the information for the reserved pages, physical-log pages, and logical-log pages	See oncheck -pr and pR: Display reserved-page information.
-ps	Checks and displays smart-large-object and sbspace metadata for an sbspace	See oncheck -cs, -cS, -ps, -pS: Check and display sbspaces.
-pS	Checks and displays smart-large-object and sbspace metadata. Lists extents and header information for individual smart large objects	See oncheck -cs, -cS, -ps, -pS: Check and display sbspaces.
-pt	Displays tblspace information for a table or fragment	See oncheck -pt and -pT: Display tblspaces for a Table or Fragment.
-pT	Same as **-pt** but also displays index-specific information and page-allocation information by page type (for dbspaces)	See oncheck -pt and -pT: Display tblspaces for a Table or Fragment.
-q	Suppresses all checking and validation message	None.
-x	Places a shared lock on the table when you check and print an index	Use with the **-ci**, **-cI**, **-pk**, **-pK**, **-pl**, or **-pL** options. For complete information, see Turn On Locking with -x.
-y	Repairs indexes when errors are detected	None.
-V	Displays the software version number and the serial number	See Obtaining utility version information.
-version	Displays the build version, host, OS, number and date, as well as the GLS version	See Obtaining utility version information.
chunknum	Specifies a decimal value that you use to indicate a particular chunk	Value must be an unsigned integer greater than 0. Chunk must exist. Execute the **-pe** option to learn which chunk numbers are associated with specific dbspaces, blobspaces or sbspaces.
database	Specifies the name of a database that you want to check for consistency	Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*.
db1	Specifies the local database that contains a data type that you want to check	Optionally specify the local database server name using the format **db1@server1**.
db2	Specifies the remote database that contains a data type that you want to check	Optionally specify the remote database server name using the format **db2@server2**.

Element	Purpose	Key Considerations
frag_dbs	Specifies the name of a dbspace that contains a fragment you want to check for consistency	Dbspace must exist and contain the fragment that you want to check for consistency. Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*.
index_name	Specifies the name of the index that you want to check for consistency	Index must exist on table and in database specified. Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*.
logical pagenum	Specifies an integer value that you use to indicate a particular page in a tblspace	Value must be an unsigned integer between 0 and 16,777,215, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.
object	Specifies the name of the DataBlade, cast, operator , user-defined data type, or UDR that you want to check	If you do not specify an object name, the database server compares all objects of the same type with the same name and owner.
owner	Specifies the owner of a table	You must specify the current owner of the table. Syntax must conform to the Owner Name segment; for more information, see *IBM Informix Guide to SQL: Syntax*.
pagenum	Indicates the page number of the sbspace metadata portion to check and display	None.
partnum	Identifies the sbspace metadata partition to check and display	None.
rowid	Identifies the rowid of the row whose contents you want to display. The rowid is displayed as part of **oncheck -pD** output	Value must be an unsigned integer between 0 and 4,277,659,295, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.
sbspace	Specifies the name of the sbspace that you want to check for consistency	None.
server	Specifies the database server name	If you omit the database server name, **oncheck** uses the name that INFORMIXSERVER specifies.
table	Specifies the name of the table that you want to check for consistency	Table exists when you execute the utility. Syntax must conform to the Table Name segment; for more information, see *IBM Informix Guide to SQL: Syntax*.
tblspacenum	Identifies the tblspace whose contents you want to display	Value must be an unsigned integer between 0 and 208,666,624, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.

oncheck -cc and-pc: Check system catalog tables

The **-cc** option checks system catalog tables for information about database tables, columns, indexes, views, constraints, stored procedures, and privileges.

```
Syntax:

>>-oncheck----+- -cc-+--database-------------------------------><
              '- -pc-'
```

The **oncheck -cc** command checks the following tables:

- **systables**
- **syscolumns**
- **sysindices**
- **systabauth**
- **syscolauth**
- **sysdepend**
- **syssyntable**
- **sysviews**
- **sysconstraints**
- **sysams**

If you do not specify a database name in the **oncheck -cc**, the command checks the listed system catalog tables for all databases.

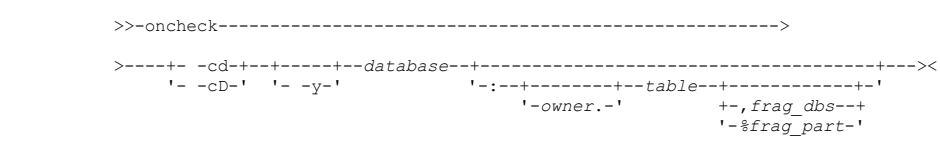
The **-pc** option performs the same checks on system catalog tables and also displays the system catalog information, including the physical address, type of locking used, row size, number of keys, extent use, the number of pages allocated and used, tblspace partnum, and index use for each table.

Before you execute **oncheck -cc** or **oncheck -pc**, execute the SQL statement UPDATE STATISTICS to ensure that an accurate check occurs. To check a table, **oncheck** compares each system catalog table to its corresponding entry in the tblspace.

oncheck -cd and oncheck -cD commands: Check pages

Use the **oncheck -cd** and **oncheck -cD** commands to check each page for consistency. Use the **oncheck -cd -y** or **oncheck -cD -y** command to repair inconsistencies.

Syntax:



The **oncheck -cd** command reads all pages, except for blobpages and sbpages, from the tblspace for the specified database, table, fragment, or multiple fragments (fragparts), and checks each page for consistency. This command compares entries in the bitmap page to the pages to verify mapping.

The **oncheck -cD** command performs the same checks as the **oncheck -cd** command, and also checks the header of each blobpage for consistency. The **oncheck -cD** command does not compare the beginning time stamps stored in the header with the ending time stamps stored at the end of a blobpage. Use the **oncheck -cD -y** command to clean up orphaned simple large objects in blobspaces, which can occur after a rollback across several log files.

If the database contains fragmented tables, but no fragment is specified, the **oncheck -cd** command checks all fragments in the table. If you do not specify a table, the command checks all of the tables in the database. By comparison, the **oncheck -pd** command displays a hexadecimal dump of specified pages but does not check for consistency.

For both the **oncheck -cd** and **oncheck -cD** commands, the **oncheck** utility locks each table as it checks the indexes for the table. To repair the pages, use **oncheck -cd -y** or **oncheck -cD -y**.

If tables are fragmented on multiple partitions in the same dbspace, the **oncheck -cd** and **oncheck -cD** commands show the partition names. The following example shows typical output for a table that has fragments in multiple partitions in the same dbspace:

```
TBLspace data check for multipart:informix.t1
              Table fragment partition part_1 in DBspace dbs1
              Table fragment partition part_2 in DBspace dbs1
              Table fragment partition part_3 in DBspace dbs1
              Table fragment partition part_4 in DBspace dbs1
              Table fragment partition part_5 in DBspace dbs1
```

When you use the **oncheck -cd** or **oncheck -cD** command, you can specify either the *frag_dbs* or the *%frag_dbs* option but not both:

- When you use the *frag_dbs* option, the utility checks all fragments in the dbspace `frag_dbs`.
- When you use the *%frag_dbs* option, the utility checks only the fragment named *frag_part*, if the PARTITION syntax was used when the fragment or table was created.

While it is possible to fragment an index with the PARTITION syntax, it is not possible to limit an index check to just one fragment or partition. For example, you can specify **oncheck -cDI my_db:my_tab,data_dbs1** or **oncheck -cDI my_db:my_tab%part1**. The **D** (data) portion of the check is limited according to the specification, however the **I** (index) check is not limited.

Examples

The following example checks the data rows, including simple large objects and smart large objects, in the **catalog** table:

```
oncheck -cD superstores_demo:catalog
```

If you specify a single fragment, the **oncheck** utility displays a single header for that fragment. For fragmented tables, one header is displayed for each fragment:

```
TBLspace data check for stores_demo:informix.tab1
 Table fragment in DBspace db1
```

Messages

If the **oncheck** utility finds no inconsistencies, a header displays for each table that the utility. For example:

```
TBLSPACE data check for stores_demo:informix.customer
```

If the **oncheck** utility finds an inconsistency, a message displays. For example:

```
BAD PAGE 2:28: pg_addr 2:28 != bp-> bf_pagenum 2:69
```

The physical address `2:28` represents page 28 of chunk number 2.

If an index that uses an access method provided by a DataBlade module cannot find the access method, you receive the following message:

```
-9845 Access method access_method_name does not exist in database.
Ensure that the DataBlade installation was successful.
```

Reference

To monitor blobspace blobpages, see oncheck -pB: Display blobspace statistics.

Related reference:
check data argument: Check data consistency (SQL administration API)

oncheck -ce, -pe: Check the chunk-free list

Syntax:

```
>>-oncheck----+- -ce-+-----------------------------------------><
              '- -pe-'
```

The **-ce** option checks each chunk-free list and corresponding free space and each tblspace extent. For more information, refer to Next-Extent Allocation and Structure of the Chunk Free-List Page, respectively. The **oncheck** process verifies that the extents on disk correspond to the current control information that describes them.

The **-pe** option performs the same checks and also displays the chunk and tblspace extent information during the check. The **-ce** and **-pe** options also check blobspaces, smart-large-object extents, and user-data and metadata information in sbspace chunks.

For information about using **oncheck -ce** and **-pe**, see managing disk space in the *IBM® Informix® Administrator's Guide*.

Use CHECK EXTENTS as the SQL administration API *command* string for **oncheck -ce**.

Related reference:
check extents argument: Check extent consistency (SQL administration API)

oncheck -ci and -cI: Check index node links

Use the **oncheck -ci** and **oncheck -cI** commands to check the ordering of key values and the consistency of horizontal and vertical node links for all indexes associated with the specified table.

The **oncheck -cI** command also checks that the key value tied to a rowid in an index is the same as the key value in the row. The **-cI** option does not cross-check data on a functional index.

Syntax:

```
>>-oncheck------------------------------------------------------>

>----+- -ci +--database--+-------------------------------------+--><
     '- -cI '            '-:--+--------+--table--+----------+-'
                              '-owner.-'         '- #index -'
```

If you do not specify an index, the option checks all indexes. If you do not specify a table, the option checks all tables in the database.

The same **-ci** repair options are available with **-cI**. If **oncheck -ci** or **oncheck -cI** detects inconsistencies, it prompts you for confirmation to repair the problem index. If you specify the **-y** (yes) option, indexes are automatically repaired. If you specify the **-n** (no) option, the problem is reported but not repaired; no prompting occurs.

If **oncheck** does not find inconsistencies, the following message appears:

```
validating indexes......
```

The message displays the names of the indexes that **oncheck** is checking.

Note: Using **oncheck** to rebuild indexes can be time consuming. Processing is usually faster if you use the SQL statements DROP INDEX and CREATE INDEX to drop and re-create the index.
The following example checks all indexes on the **customer** table:

```
oncheck -cI -n stores_demo:customer
```

The following example checks the index **zip_ix** on the **customer** table:

```
oncheck -cI -n stores_demo:customer#zip_ix
```

If indexes are fragmented on multiple partitions in the same dbspace, the **oncheck -ci** and **oncheck -cI** commands show the partition names. The following example show typical output for an index that has fragments in multiple partitions in the same dbspace:

```
Validating indexes for multipart:informix.t1...
                Index idx_t1
                  Index  fragment partition part_1 in DBspace dbs1
                  Index  fragment partition part_2 in DBspace dbs1
                  Index  fragment partition part_3 in DBspace dbs1
                  Index  fragment partition part_4 in DBspace dbs1
                  Index  fragment partition part_5 in DBspace dbs1
```

By default, the database server does not place a shared lock on the table when you check an index with the **oncheck -ci** or **oncheck -cI** commands unless the table uses page locking. For absolute assurance of a complete index check, you can execute **oncheck -ci**or **oncheck -cI** with the **-x** option. With the **-x** option, **oncheck** places a shared lock on the table, and no other users can perform updates, inserts, or deletes until the check has completed. For more information about using **oncheck -ci** and **oncheck -cI** with the **-x** option, Turn On Locking with -x.

When you execute **oncheck** on an external index, the user-defined access method is responsible for checking and repairing an index. If an index that employs a user-defined access method cannot find the access method, the database server reports an error. The **oncheck** utility does not repair inconsistencies in external indexes. You should not use **oncheck -cI** on a table that contains more than one type of index.

The **oncheck** utility requires sort space when examining an index. The amount of sort space required is the same as that needed to build the index. For information about calculating the amount of temporary space needed, see Estimating temporary space for index builds. If you receive the error "`no free disk space for sort`," you must estimate the amount of temporary space needed and make that space available.

For more information about indexes, see Structure of B-Tree Index Pages.

oncheck -cr and -cR: Check reserved pages

Syntax:

```
>>-oncheck----+- -cr-+-----------------------------------------><
              '- -cR-'
```

The -cr option checks each of the root dbspace reserved pages as follows:

- It validates the contents of the ONCONFIG file with the PAGE_CONFIG reserved page.
- It ensures that all chunks can be opened, that chunks do not overlap, and that chunk sizes are correct.

The -cR option performs the same checking and validation, and also checks all logical-log and physical-log pages for consistency. The -cr option is considerably faster because it does not check the log-file pages.

If you have changed the value of a configuration parameter (either through **onparams**, **onmonitor**, **onspaces**, or by editing the configuration file), but you have not yet reinitialized shared memory, **oncheck -cr** and **oncheck -cR** detect the inconsistency and return an error message.

If **oncheck -cr** does not display any error messages after you execute it, you can assume that all three items in the preceding list were checked successfully.

For more information on reserved pages, see Reserved Pages.

Related reference:
oncheck -pr and pR: Display reserved-page information

oncheck -cs, -cS, -ps, -pS: Check and display sbspaces

Syntax:

```
>>-oncheck--+-+- -cs-+--+---------+------------------+--------><
            | '- -cS-'  '-sbspace-'                  |
            '-+- -ps-+--+--------------------------+-'
              '- -pS-'  '-sbspace--partnum--pagenum-'
```

The **-cs** option checks sbspaces. The **-ps** option checks sbspaces and extents.

The **-cS** option validates and displays metadata for an sbspace.

The **-ps** option checks sbspaces and extents. If you do not specify the sbspace name, these options check all sbspaces.

The **-pS** option validates and displays metadata for an sbspace and also lists extents and header information for smart large objects.

If you do not specify the sbspace name, all sbspaces will be checked. The following example checks and displays metadata for **test_sbspace**:

```
oncheck -ps test_sbspace
```

If you specify **rootdbs** as the sbspace name with the **-cs** or **-ps** options, **oncheck** checks the root dbspace.

For more information about using the **-cs, -cS, -ps,** and**-pS** options, see the *IBM® Informix® Administrator's Guide*.

oncheck -pB: Display blobspace statistics

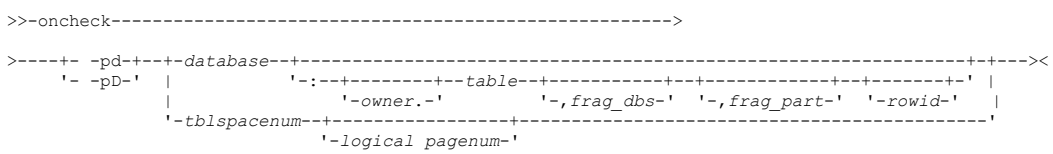
Syntax:

```
>>-oncheck---- -pB----database--+-------------------------------------+-><
                                '-:--+--------+--table--+-----------+-'
                                     '-owner.-'         '-,frag_dbs-'
```

The **-pB** option displays statistics that describe the average fullness of blobspace blobpages in a specified table. These statistics provide a measure of storage efficiency for individual simple large objects in a database or table. If you do not specify a table or fragment, the option displays statistics for the entire database. For more information, see optimizing blobspace blobpage size in the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.

oncheck -pd and pD: Display rows in hexadecimal format

Syntax:



The **-pd** option takes a database, a table, a fragment, a fragment partition (fragpart), and a specific rowid or tblspace number and logical page number as input. In every case, **-pd** prints page-header information and displays the specified rows for the database object (database, table, fragment, internal rowid, or page number) that you specify in hexadecimal and ASCII format. No checks for consistency are performed.

Element	Purpose	Key Considerations
database	Specifies the name of a database that you want to check for consistency	Syntax must conform to the Identifier segment; see *IBM® Informix® Guide to SQL: Syntax*.
frag_dbs	Specifies the name of a dbspace that contains a fragment you want to check for consistency	Dbspace must exist and contain the fragment that you want to check for consistency. Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*.
frag_part	Specifies the fragment partition	For fragmented tables or an index that use expression-based or round-robin distribution schemes, you can create multiple partitions, which are collections of pages for a table or index, within a single dbspace. This partition is referred to as a *fragment partition* or *fragpart*.
logical pagenum	Specifies an integer value that you use to indicate a particular page in a tblspace	Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier. Value must be an unsigned integer between 0 and 16,777,215, inclusive.
owner	Specifies the owner of a table	You must specify the current owner of the table. Syntax must conform to the Owner Name segment; see *IBM Informix Guide to SQL: Syntax*.
rowid	Identifies the rowid of the row whose contents you want to display. The rowid is displayed as part of **oncheck -pD** output	Value must be an unsigned integer between 0 and 4,277,659,295, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.
table	Specifies the name of the table that you want to check for consistency	Table exists when you execute the utility. Syntax must conform to the Table Name segment; see *IBM Informix Guide to SQL: Syntax*.
tblspacenum	Identifies the tblspace whose contents you want to display	Value must be an unsigned integer between 0 and 208,666,624, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.

If you specify an internal rowid (expressed as a hexadecimal value), the rowid maps to a particular page, and all rows from that page are printed.

If you specify a logical page number (expressed as a decimal), all the rows of the tblspace number with the logical page number are printed.

If you specify a fragment, all the rows in the fragment are printed, with their rowids, forward pointers, and page type.

If you specify a table, all the rows in the table are printed, with their rowids, forward pointers, and page type.

If you specify a database, all the rows in all the tables in the database are printed. TEXT and BYTE column descriptors stored in the data row are printed, but TEXT and BYTE data itself is not.

The **-pD** option prints the same information as **-pd**. In addition, **-pD** prints TEXT and BYTE values stored in the tblspace or header information for simple large objects stored in a blobspace blobpage. The following example show different options for the **oncheck -pd** and **oncheck -pD** commands:

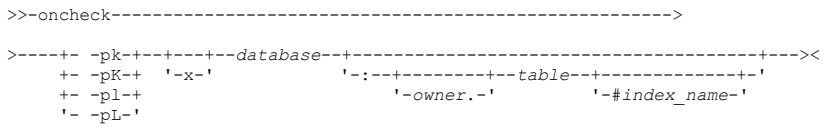
```
oncheck -pd stores_demo:customer,frgmnt1
oncheck -pd stores_demo:customer
oncheck -pD stores_demo:customer 0x101
```

The following example shows a partial output of an **oncheck -pD** command:

```
oncheck -pD multipart:t1 :

TBLspace data check for multipart:informix.t1
                Table fragment partition part_1 in DBspace dbs1
page_type  rowid    length fwd_ptr
HOME       101      24     0
    0:  0  0  0  a 47 48 49 20 20 20 20 20 20 20 20 20    ....GHI
   16: 20 20 20 20 20 20 20 20
                                        ........
```


oncheck -pk, -pK, -pl, -pL: Display index information

Syntax:



The **-pk** option performs the same checks as the **-ci** option and in addition, displays the key values for all indexes on the specified table as it checks them.

The **-pK** option performs the same checks as the **-cI** option and in addition, displays the key values and rowids as it checks them.

The **-pl** option performs the same checks as the **-ci** option and displays the key values, but checks only leaf-node index pages. It ignores the root and branch-node pages.

The **-pL** option performs the same checks as the **-cI** option and displays the key values and rowids, but checks only leaf-node index pages. It ignores the root and branch-node pages.

Element	Purpose	Key Considerations
database	Specifies the name of a database that you want to check for consistency	Syntax must conform to the Identifier segment; see *IBM® Informix® Guide to SQL: Syntax*.
index_name	Specifies the name of the index that you want to check for consistency	Index must exist on table and in database specified. Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*.
owner	Specifies the owner of a table	You must specify the current owner of the table. Syntax must conform to the Owner Name segment; see *IBM Informix Guide to SQL: Syntax*.
table	Specifies the name of the table that you want to check for consistency	Table exists when you execute the utility. Syntax must conform to the Table Name segment; see *IBM Informix Guide to SQL: Syntax*.
-x	Places a shared lock on the table when you check and print an index	For complete information, see Turn On Locking with -x.

If any of the **oncheck** options detect inconsistencies, you are prompted for confirmation to repair the problem index. If you specify the **-y** (yes) option, indexes are automatically repaired. If you specify the **-n** (no) option, the problem is reported but not repaired; no prompting occurs.

The following example displays information about all indexes on the **customer** table:

```
oncheck -pl -n stores_demo:customer
```

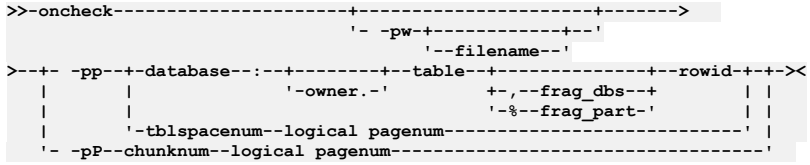
The following example displays information about the index **zip_ix**, which was created on the **customer** table:

```
oncheck -pl -n stores_demo:customer#zip_ix
```

By default, the database server does not place a shared lock on the table when you check an index with the **oncheck -pk, -pK, -pl,** or **-pL** options unless the table uses page locking. For absolute assurance of a complete index check, you can execute **oncheck –pk**, **oncheck -pK** , **oncheck -pl** , or **oncheck -pL** with the **-x** option. With the **-x** option, **oncheck** places a shared lock on the table, and no other users can perform updates, inserts, or deletes until the check has completed. For more information on using the **-x** option, Turn On Locking with -x.

For more information on **oncheck -ci**, see oncheck -ci and -cI: Check index node links. For more information index pages, see Structure of B-Tree Index Pages.

oncheck -pp and -pP: Display the contents of a logical page



Element	Purpose	Key Considerations
database	Specifies the name of a database that you want to check for consistency	Syntax must conform to the Identifier segment; see *IBM® Informix Guide to SQL: Syntax*.
chunknum	Specifies a decimal value that you use to indicate a particular chunk	Value must be an unsigned integer greater than 0. Chunk must exist.
frag_dbs	Specifies the name of a dbspace that contains a fragment you want to check for consistency	Dbspace must exist and contain the fragment that you want to check for consistency. Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*.
frag_part	Specifies the partition name of the fragment to be checked. This is useful in cases where more than one fragment of a table was created in the same dbspace.	For fragmented tables or an index that use expression-based or round-robin distribution schemes, you can create multiple partitions, which are collections of pages for a table or index, within a single dbspace. This partition is referred to as a *fragment partition* or *fragpart*.

Element	Purpose	Key Considerations
logical pagenum	Specifies an integer value that you use to indicate a particular page in a tblspace	Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier. Value must be an unsigned integer between 0 and 16,777,215, inclusive.
owner	Specifies the owner of a table	You must specify the current owner of the table. Syntax must conform to the Owner Name segment; see *IBM Informix Guide to SQL: Syntax.*
rowid	Identifies the rowid of the row whose contents you want to display. The rowid is displayed as part of **oncheck -pD** output	Value must be an unsigned integer between 0 and 4,277,659,295, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.
table	Specifies the name of the table that you want to check for consistency	Table exists when you execute the utility. Syntax must conform to the Table Name segment; see *IBM Informix Guide to SQL: Syntax.*
tblspacenum	Identifies the tblspace whose contents you want to display	Value must be an unsigned integer between 0 and 208,666,624, inclusive. Value can be expressed as an unsigned integer or hexadecimal that begins with 0x identifier.

The **-pp** option has the following syntax variations:

Invocation	Explanation
oncheck -pp tblspc lpn <pages>	Displays the contents of a logical page using a tblspace number and logical page number. You can also specify an optional parameter specifying the number of pages to be printed.
oncheck -pp *tblspc lpn -h*	Displays only the header of a logical page using a tblspace number and logical page number.
oncheck -pp *database:table rowid*	Displays the contents of a logical page using a database name, table name, and the Informix® internal rowid. You can obtain this internal rowid with the **oncheck -pD** command. This internal rowid is not the serial rowid that is assigned in tables created with the CREATE TABLE tabname WITH ROWIDS statement. For more information, see Definition of Rowid

The page contents appear in ASCII format. The display also includes the number of slot-table entries on the page. The following example shows different invocations of the **oncheck -pp** command:

```
oncheck -pp stores_demo:orders 0x211  # database:owner.table,   # fragment rowid
oncheck -pp stores_demo:informix.customer,frag_dbspce1 0x211
oncheck -pp 0x100000a 25 # specify the tblspace number and # logical page number
```

The **-pP** option provides the following syntax variations:

Invocation	Explanation
oncheck -pP chunk# offset pages	Displays the contents of a logical page using a chunk number and an offset. You can also specify an optional parameter specifying the number of pages to be printed.
oncheck -pP *chunk# offset -h*	Displays only the header of a logical page using a chunk number and an offset.

Note: The output for chunk page displays both the `start` and the `length` fields in decimal format.
Note:
The -pw option is required only when the Storage space encryption feature is enabled and no stash file is in use. Supply an optional path to a file containing the keystore password, otherwise oncheck will prompt for a password before displaying the requested page(s).

The following example shows typical output using the **onstat -pP** command:

```
oncheck -pP 1 5 2
addr          stamp          nslots          flag       type          frptr       frcnt       next       prev
stamp          100005          250181          2          1000       ROOTRSV          320       1716       0
0       250181       slot       ptr       len       flg
...
addr       stamp          nslots          flag       type          frptr       frcnt       next       prev
stamp       100005    6       250182       2       1000       ROOTRSV       128       1908       0       0
250182       slot       ptr       len       flg       1       24       56       0
2       80       48         0
```


oncheck -pr and pR: Display reserved-page information

The -pr option performs the same checks as **oncheck -cr** command and displays the reserved-page information.

```
>>-oncheck----+- -pr-+---+----------------------+------------><
              '- -pR-'   '- -pw-+------------+--'
                                '--filename--'
```

The -pR option performs the same checks as the **oncheck -cR** command, displays the reserved-page information, and displays detailed information about logical-log and physical-log pages, including marking the start and end of the active physical-log pages.

The following example show output of the **oncheck -pr** command:

```
Validating IBM Informix Dynamic Server reserved pages

    Validating PAGE_PZERO...

    Identity                        IBM Informix Dynamic Ser
                                    ver Copyright 2001, 2016
```

```
                                  IBM Corporation
    Database system state          0
    Database system flags          0xc039
        64-bit server
        BigChunk page flags are not in use
        Encryption-at-rest is enabled using cipher 'aes192'
        The ROOT Dbspace is encrypted
    Page Size                      2048 (b)
    Date/Time created              05/12/2016 18:01:09
    Version number of creator      28
    UID of rootdbs creator         200
    Index Page Logging             OFF
    HA Disk Owner                  <null>
```

If you have changed the value of a configuration parameter, but you have not yet reinitialized shared memory, the **oncheck -pr** and **oncheck -pR** commands detect the inconsistency and return an error message.

Note:
The -pw option is required only when the Storage space encryption feature is enabled and no stash file is in use. Supply an optional path to a file containing the keystore password, otherwise oncheck will prompt for a password before displaying the requested page(s).

Related concepts:
Reserved Pages
Related reference:
oncheck -cr and -cR: Check reserved pages
DISK_ENCRYPTION configuration parameter

oncheck -pt and -pT: Display tblspaces for a Table or Fragment

The **oncheck -pt** and **oncheck -pT** options print a tblspace report for a specific table or fragment. The only difference between these options is that **oncheck -pT** prints more information, including some index-specific information.

Syntax

```
>>-oncheck-----------------------------------------------------><

>----+- -pt-+--database--+--------------------------------------+---><
     '- -pT-'            '-:--+--------+--table--+------------+-'
                              '-owner.-'         '-,frag_dbs-'
```

Table 1. Options of the oncheck -pt and oncheck -pT commands

Element	Purpose	Key Considerations
database	Specifies the name of a database that you want to check for consistency	Syntax must conform to the Identifier segment; see Identifier.
frag_dbs	Specifies the name of a dbspace that contains a fragment you want to check for consistency	The dbspace must exist and contain the fragment that you want to check for consistency. Syntax must conform to the Identifier segment; see Identifier.
owner	Specifies the owner of a table	You must specify the current owner of the table. Syntax must conform to the Owner Name segment; see Owner name.
table	Specifies the name of the table that you want to check for consistency	The table must exist. Syntax must conform to the Identifier segment; see Identifier.

The **-pt** option prints a tblspace report for the table or fragment with the specified name and database. If you do not specify a table, the option displays this information for all tables in the database. The report contains general allocation information, including the maximum row size, the number of keys, the number of extents, their sizes, the pages allocated and used per extent, the current serial value, and the date that the table was created. The **-pt** output prints the page size of the tblspace, the number of pages (allocated, used, and data) in terms of logical pages.

The **TBLspace Flags** field shows information about the tblspace configuration, including whether the tblspace is used for Enterprise Replication or time series data.

The **Extents** fields list the physical address for the tblspace **tblspace** entry for the table and the address of the first page of the first extent. The extent list shows the number of logical and physical pages in every extent.

The **-pT** option prints the same information as the **-pt** option. In addition, the **-pT** option displays:

- Index-specific information
- Page-allocation information by page type (for dbspaces)
- The number of any compressed rows in a table or table fragment and the percentage of table or table-fragment rows that are compressed
  If table or fragment rows are not compressed, the "Compressed Data Summary" section does not appear in the output.

Plan when you want to run the **-pT** option, because it forces a complete scan of partitions.

Output for both **-pt** and **-pT** contains listings for **Number of pages used**. The value shown in the output for this field is never decremented because the disk space allocated to a tblspace as part of an extent remains dedicated to that extent even after you free space by deleting rows. For an accurate count of the number of pages currently used, see the detailed information about tblspace use (organized by page type) that the **-pT** option provides.

Example of oncheck -pt Output

The following example shows output of the **oncheck -pt** command:

```
TBLspace Report for testdb:tab1

Physical Address                2:10
Creation date                   10/07/2004 17:01:16
TBLspace Flags                  801         Page Locking
                                                TBLspace use 4 bit bit-maps
Maximum row size                14
Number of special columns       0
Number of keys                  0
Number of extents               1
Current serial value            1
Pagesize (k)                    4
First extent size               4
Next extent size                4
Number of pages allocated       340
Number of pages used            337
Number of data pages            336
Number of rows                  75806
Partition partnum               2097154
Partition lockid                2097154

Extents
       Logical Page     Physical Page         Size Physical Pages
                  0             2:106         340         680
```

Example of oncheck -pT Output

The following example shows output of the **oncheck -pT** command:

```
TBLspace Report for database_a:nilesh.table_1a

                  Table fragment partition dbspace1 in DBspace dbspace1

    Physical Address                3:5
    Creation date                   03/21/2009 15:35:47
    TBLspace Flags                  8000901     Page Locking
                                                TBLspace contains VARCHARS
                                                TBLspace use 4 bit bit-maps
                                                TBLspace is compressed
    Maximum row size                80
    Number of special columns       1
    Number of keys                  0
    Number of extents               1
    Current serial value            100001
    Current SERIAL8 value           1
    Current BIGSERIAL value         1
    Current REFID value             1
    Pagesize (k)                    2
    First extent size               8
    Next extent size                8
    Number of pages allocated       24
    Number of pages used            22
    Number of data pages            14
    Number of rows                  500
    Partition partnum               3145730
    Partition lockid                3145730

    Extents
         Logical Page     Physical Page         Size Physical Pages
                    0           3:16053           24          24

Type                    Pages      Empty  Semi-Full       Full  Very-Full
    ---------------- ---------- ---------- ---------- ---------- ----------
    Free                        9
    Bit-Map                     1
    Index                       0
    Data (Home)                14
    Data (Remainder)            0          0          0          0          0
                     ----------
    Total Pages                24

    Unused Space Summary

        Unused data bytes in Home pages                1177
        Unused data bytes in Remainder pages              0

    Home Data Page Version Summary

                Version                                 Count

                    0 (current)                          14

    Compressed Data Summary

    Number of compressed rows and percentage of compressed rows  500 100.00
```

Note: oncheck -p[tT] now indicates the last time each index fragment was used for a query. This access time is stored on the partition page on disk, it will survive an instance restart.

Related reference:

TBLTBLFIRST configuration parameter

check partition argument: Check partition consistency (SQL administration API)

print partition argument: Print partition information (SQL administration API)
Tblspace tblspace entries
Related information:
Performance of in-place alters for DDL operations
Resolve outstanding in-place alter operations
Monitor simple large objects in a dbspace with oncheck -pT

Turn On Locking with -x

The **-x** option can be appended to the **-ci**, **-cI**, **-pk**, **-pK**, **-pl**, and **-pL** options to place a shared lock on affected tables. While the table is locked, no other users can perform inserts, updates, and deletions while **oncheck** checks or prints the index. Without the **-x** option for tables with row locking, **oncheck** only places an IS (intent shared) lock on the table, which prevents actions such as dropping the table or the indexes during the check.

For example, the following sample command instructs **oncheck** to lock indexes for the **customer** table while it validates the order of key values, validates horizontal links, and ensures that no node appears twice in the index:

```
oncheck -cix stores_demo:customer
```

When you specify option **-x**, **oncheck** locks indexes for tables that use row locking. If **oncheck** detects page-lock mode, it displays a warning message and places a shared lock on the table regardless.

Send Special Arguments to the Access Method with -u

You can use the **-u** option to send special arguments to the access method. The possible arguments depend on the access method. For example, the R-tree access method supports the **display** option, as the following example shows:

```
oncheck -pl -u "display"
```

Use commas to separate multiple arguments in the argument string.

For information on valid arguments for your access method, refer to the user manual for your access method.

Return Codes on Exit

The **oncheck** utility returns the following codes on exit.

```
GLS failures:-1
Invalid srial/key:2
Onconfig access error:2
Invalid onconfig settings:2
Invalid arguments to oncheck:2
Error connecting database server:1
Warning reported by oncheck:1
error detected by oncheck:2
no errors detected by oncheck:0
```

Windows only:

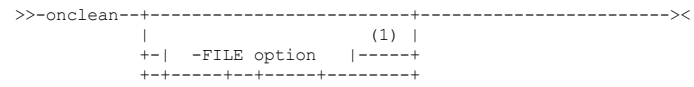
```
Not properly installed:1
Authentication error:2
```


The onclean utility

Use the **onclean** utility to force a shut down of the database server when normal shut down with the **onmode** utility fails or when you cannot restart the server. The **onclean** utility attempts to clean up shared memory, semaphores, and stops database server virtual processes.

Syntax

On UNIX and Linux, you must be user **root** or **informix** to run the **onclean** command. On Windows, you must be in the **Informix-Admin** group to run the command.



```
| '- -k-'  '- -y-'        |
'-+-----------+-----------'
  +- -V-------+
  '- -version-'
```

Notes:

1. See The -FILE option.

Table 1. Syntax Elements of the **onclean** Command

Element	Purpose
-k	Shuts down a server that is online by stopping database server virtual processes and attempting to clean up the remaining semaphores and shared-memory segments, even if they are still running.
-V	Displays short version information.
-version	Displays full version information.
-y	Does not prompt for confirmation.

Usage

Use the **onclean** utility to stop the database server only if the **onmode** utility is unable to shut it down or you cannot restart the server. Perhaps the database server shut down in an uncontrolled way and cannot recover, or it is hung. If the database server fails to restart, the previous instance of the database server is still attached to the shared-memory segments. Check the message log to see if the database server shut down abnormally. The **onclean** utility stops all **oninit** processes and attempts to remove all shared-memory segments and semaphores that are recorded in the **$INFORMIXDIR/etc/.conf.$INFORMIXSERVER** file.

Attention: Use the **onclean** utility with caution. When you run **onclean**, any pending transactions and processes fail to complete, and user sessions are disconnected abruptly. However, the database server rolls back transactions when it restarts.
The INFORMIXDIR, INFORMIXSERVER, INFORMIXSQLHOSTS, and ONCONFIG environment variables must be set with valid values to run this utility.

The **onclean** command that you use depends on the situation:

- If you are not sure whether the database server is offline, use the **onclean** command without options. If the database server is still online, a message appears directing you to run the **onclean -k** command.
- If the database server is offline, use the **onclean** command.
- If the database server is online and you are sure that you want to force it to shut down, use the **onclean -k** command.

You can use the **onclean** utility only to shut down the local database server; you cannot use it to shut down a remote database server. The **onclean** utility should not be used to shut down an entire high-availability cluster or a remote database server.

The **onclean** utility might not be able to clean up shared memory segments that were in use by the database server in every situation. The **onclean** utility attempts to terminate only **oninit** processes. The **onclean** utility does not succeed in the following situations:

- If a non-database server process is attached to the shared memory segment before running the **onclean** command, the **onclean** utility does not stop this process to remove the shared memory segment.
- The **onclean** might not be able to guarantee a clean server startup is when an application or database server utility is connected to a network port. If the user tries to initialize a database server instance on the same network port, then the database server cannot start the listener thread and fails to start. The **onclean** utility does not stop the application to free the network port.

You can automate shutting down the database server with the **onshutdown** script, which calls the **onclean -ky** command if necessary.

Return Codes

0
: Successful

1
: Failure because of one of the following problems:

  - Incorrect environment variable settings
  - Incorrect privileges to run the **onclean** command
  - Incorrect command syntax
  - Corrupted information
  - Running the **onclean** command without the **-k** option on a server that is still online

2
: Failure because one or more OS system calls used by **onclean** returned an error.

- The onshutdown script
  Use the **onshutdown** script to automate shutting down the database server. The script attempts to shut down the server normally. If the server has not shut down after a specified time, the script forces the server to shut down.

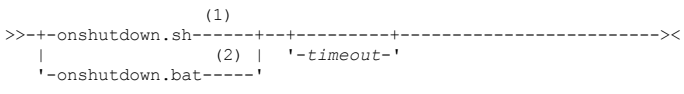
Related reference:
Taking the Database Server to Offline Mode with the -k Option

The onshutdown script

Use the **onshutdown** script to automate shutting down the database server. The script attempts to shut down the server normally. If the server has not shut down after a specified time, the script forces the server to shut down.

Syntax

The **onshutdown** script first runs the **onmode -ky** command. After a specified wait time, the script runs the **onclean -ky** command.
On UNIX and Linux, you must be user **root** or **informix** to run the **onshutdown** script. On Windows, you must be in the **Informix-Admin** group to run the **onshutdown** script.



Notes:

1. UNIX
2. Windows

Table 1. Syntax Elements of the **onshutdown** Script

Element	Purpose
timeout	The number of seconds after the **onmode -ky** command has been run before running the **onclean -ky** command. Must be a positive integer from 10 to 60. The default value is 30 seconds.

Usage

Use the **onshutdown** script only when forcing the database server to shut down would be appropriate.

Attention: Use the **onshutdown** script with caution. If the script needs to run the **onclean -ky** command, any pending transactions and processes fail to complete, and user sessions are disconnected abruptly. However, the database server rolls back transactions when it restarts.
The INFORMIXDIR, INFORMIXSERVER, INFORMIXSQLHOSTS, and ONCONFIG environment variables must be set with valid values to run this utility.

You can only use the **onshutdown** script to shut down the local database server; you cannot use it to shut down a remote database server. The **onshutdown** script should not be used to shut down an entire high-availability cluster or a remote database server instance.

The **onshutdown** script has a 10 second time period during which it can be aborted.

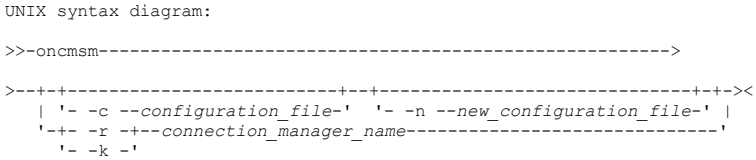
Related reference:
Taking the Database Server to Offline Mode with the -k Option

The oncmsm utility

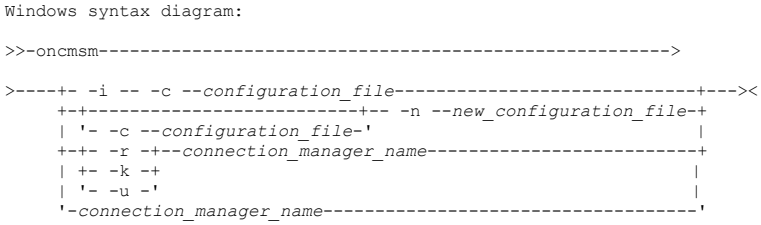
Use the **oncmsm** utility to start or shut down a Connection Manager, load a new configuration file into a Connection Manager to modify the Connection Manager's settings, or update the format of a configuration file.

Syntax

UNIX syntax diagram:



Windows syntax diagram:



Element	Purpose	Key considerations

Element	Purpose	Key considerations
-c	Starts the Connection Manager or converts a configuration file to the current Connection Manager format.	
connection_manager_name	Specifies the name of a Connection Manager instance.	
-i	Installs the Connection Manager as a Windows service.	This option is valid for Windows platforms only.
-k	Shuts down a specific instance of the Connection Manager.	
-n	Specifies the name of a converted configuration file.	
new_configuration_file	The name of file that is output to the $INFORMIXDIR/etc directory as part of the format-conversion process.	
configuration_file	The name of the configuration file located in the $INFORMIXDIR/etc directory.	If the configuration file is not specified, the Connection Manager attempts to load $INFORMIXDIR/etc/cmsm.cfg.
-r	Reloads the Connection Manager settings without stopping and restarting the Connection Manager.	
-u	Uninstall the Connection Manager Windows service.	This option is valid for Windows platforms only.

Usage

Run the **oncmsm** utility from the command line to initialize the Connection Manager. You can add, change, or delete Service Level Agreements (SLAs) while the Connection Manager is running and then reload the configuration file.

The Connection Manager configuration file in versions of IBM® Informix® Client Software Development Kit (Client SDK) prior to version 3.70.xC3 are incompatible with the current version of the Connection Manager. You must convert configuration files from versions prior to 3.70.xC3. You must have read permission on the configuration file you want to convert and write permission on the configuration file you want to create.

UNIX Only: The following users can run the **oncmsm** utility:

- User **informix**
- User **root**, if the user has privileges to connect to the **sysadmin** database
- A member of the **DBSA** group, if the user has privileges to connect to the **sysadmin** database

Windows Only: The following users can run the **oncmsm** utility:

- A member of the **Informix-Admin** group
- User **administrator**, if the user has privileges to connect to the **sysadmin** database
- A member of the **DBSA** group, if the user has privileges to connect to the **sysadmin** database

You must install the **oncmsm** utility as a service before you can start it.

The **oncmsm** utility can be started two ways:

- Run an **oncmsm** command.
- Click Start > Control Panel > Administrative Tools > Services and then start **oncmsm**.

If you are using multiple Connection Managers, you can run **onstat -g cmsm** to display the names of Connection Manager instances.

Example 1: Starting a Connection Manager (UNIX)

For the following example the Connection Manager's configuration_file_1 exists in the $INFORMIXDIR/etc directory. To start the Connection Manager, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -c configuration_file_1
```

The Connection Manager starts.

Example 2: Starting a Connection Manager (Windows)

For the following example the Connection Manager's configuration_file_1 exists in the $INFORMIXDIR/etc directory. To start the Connection Manager, run the following commands on the computer that the Connection Manager is installed on:

```
oncmsm -i -c configuration_file_2
oncmsm connection_manager_2
```

The Connection Manager named **connection_manager_2** starts.

Example 3: Stopping a Connection Manager

To stop the Connection Manager, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -k connection_manager_3
```

The Connection Manager named **connection_manager_3** stops.

Example 4: Reloading Connection Manager settings

For the following example, $INFORMIXDIR\etc\configuration_file_4 for a Connection Manager named **connection_manager_4** has changed. To update the Connection Manager's settings, run the following command on the computer that **connection_manager_4** is installed on:

```
oncmsm -r connection_manager_4
```

Example 5: Converting a Connection Manager configuration file to a current format

For the following example the Connection Manager's configuration file that is named cmsm.cfg exists in the $INFORMIXDIR/etc directory. To start the Connection Manager, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -n configuration_file_5
```

The **oncmsm** utility converts cmsm.cfg to the current configuration file format, and then outputs a file named configuration_file_5 into $INFORMIXDIR/etc/.

Example 6: Converting a specific Connection Manager configuration file to a current format

For the following example the Connection Manager's configuration file that is named configuration_file_4 exists in the $INFORMIXDIR/etc directory. To start the Connection Manager, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -c configuration_file_6 -n configuration_file_7
```

The **oncmsm** utility converts configuration_file_6 to the current configuration file format and then outputs a file named configuration_file_7 into $INFORMIXDIR/etc/.

Example 7: Uninstalling a Connection Manager (Windows)

For the following example, you have installed a Connection Manager named **connection_manager_4** as a Windows service. To unistall the Connection Manager, run the following command on the computer that the Connection Manager is installed on:

```
oncmsm -u connection_manager_4
```

The **oncmsm** utility uninstalls the Connection Manager.

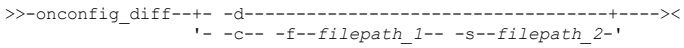
Related information:

CMCONFIG environment variable
Connection management through the Connection Manager
Starting Connection Managers on UNIX and Linux
Starting Connection Managers on Windows

The onconfig_diff utility

Use the **onconfig_diff** utility to compare two onconfig files.

Syntax



Element	Description
-d	Compares the current onconfig settings to the default settings.
-c	Compares one onconfig file to another.
-f *filepath_1*	Specifies the first file name to compare. Provide the path to the file unless the file is in the $INFORMIXDIR/bin directory.
-s *filepath_2*	Specifies the second file name to compare. Provide the path to the file unless the file is in the $INFORMIXDIR/bin directory.

Usage

Run the **onconfig_diff** utility to compare two different onconfig files. The **onconfig_diff** utility is in $INFORMIXDIR/bin.

The two files that you want to compare must be in the same directory.

Here are some ways that you can use the utility:

- Compare your current onconfig with the onconfig.std of same version.
- Compare your current onconfig with the onconfig.std of a newer version.
- Compare two onconfig files from different servers.

Example

In this example, the onconfig.std file is compared against the onconfig.production file:

```
$ onconfig_diff -c -f onconfig.std -s onconfig.production
```

Here is the output from this command:

```
=========================================
File 1: onconfig.std
File 2: onconfig.production
=========================================
Parameters Found in File 1, not in File 2
=========================================

FULL_DISK_INIT  0

NETTYPE         ipcshm,1,50,CPU

NUMFDSERVERS    4
...
=============================================
Parameters Found in File 2, not in File 1
=============================================

JVPJAVAHOME    $INFORMIXDIR/extend/krakatoa/jre

...
=============================================
Parameters Found in both files, but different
=============================================
ROOTPATH

File 1: $INFORMIXDIR/tmp/demo_on.rootdbs
File 2: /usr2/support/grantf/g1150fc8/rootdbs

LOGFILES

File 1: 6
File 2: 10

LOGSIZE

File 1: 10000
File 2: 3000
...
```

Related tasks:
Modifying the onconfig file

The ondblog utility

Use the **ondblog** utility to change the logging mode for one or more databases.

```
>>-ondblog--+-buf-------+--+-------------+---------------------><
            +-unbuf-----+  | .-,-------. |
            +-nolog-----+  | V         | |
            +-ansi------+  +---db_list-+-+
            +-cancel----+  '- -f--dbfile-'
            +- -V-------+
            '- -version-'
```

Element	Purpose	Key Considerations
buf	Sets the logging mode so that transaction information is written to a buffer before it is written to a logical log	None.
unbuf	Sets the logging mode so that data is not written to a buffer before it is written to a logical log	None.
nolog	Sets the logging mode so that no database transactions are logged	None.
ansi	Changes database logging to be ANSI compliant	Once you create or convert a database to ANSI mode, you cannot change it back to any of the other logging modes.
cancel	Cancels the logging-mode change request before the next level-0 backup occurs	None.
-f *dbfile*	Changes the logging status of the databases that are listed (one per line) in the text file whose pathname is given by *dbfile*	This command is useful if the list of databases is long or used often.
db_list	Names a space-delimited list of databases whose logging status is to be changed	If you do not specify anything, all databases that the database server manages are modified.

Usage

If you turn on transaction logging for a database, you must create a level-0 backup of all of the storage spaces that contain data in the database before the change takes effect.

For more information and examples of logging modes, see Modify the database-logging mode with ondblog.

Alternatively, you can change the logging mode by using an SQL administration API command with the alter logmode argument.

You cannot use the **ondblog** utility on High-Availability Data Replication (HDR) secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

Return codes

The **ondblog** utility logs messages in the BAR_ACT_LOG file.

For many of the return codes, you can check the ON-Bar logs to find the source of the problem:

1. Check the BAR_ACT_LOG file for accompanying messages.
2. Set the BAR_DEBUG configuration parameter to a positive integer and retry the operation.
3. Check the ON-Bar debug log file.

Table 1. Return codes for the ondblog utility

Return code	Description	User action
1	An error reading the onconfig file.	Check that the onconfig file is in the $INFORMIXDIR/etc/$ONCONFIG directory. If the BAR_ACT_LOG and BAR_DEBUG_LOG configuration parameters are set, make sure that the files are valid.
2	A linked list error.	See the accompanying message.
3	The user is not authorized to run the command.	Run the **ondblog** commands as the **root** user, user **informix**, as a Windows administrator, or as the owner of the database server.
4	Failed to set the INFORMIXSHMBASE environment variable to -1.	Contact Software Support.
5	The database server is not online.	Start the database server.
6	The command option is invalid.	Correct the spelling of the option.
7	Failed to communicate with the backup utility.	Check that the database server is online and that ON-Bar is configured.
9	Failed to allocate memory.	Check the ON-Bar logs for more information. You might need to ask your System Administrator to either increase your swap space or to install more memory in your system.
16	Failed to open the file.	Check the ON-Bar logs for more information.
17	Cannot change to the specified logging mode.	Check the ON-Bar logs for more information.
18	Failed to change the logging mode.	Check the ON-Bar logs for more information.
19	An SQL error occurred.	Check the ON-Bar logs for more information.
20	An empty list problem occurred.	Check the ON-Bar logs for more information.

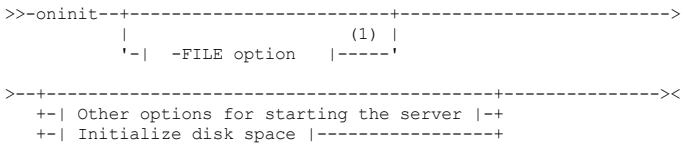
The oninit utility

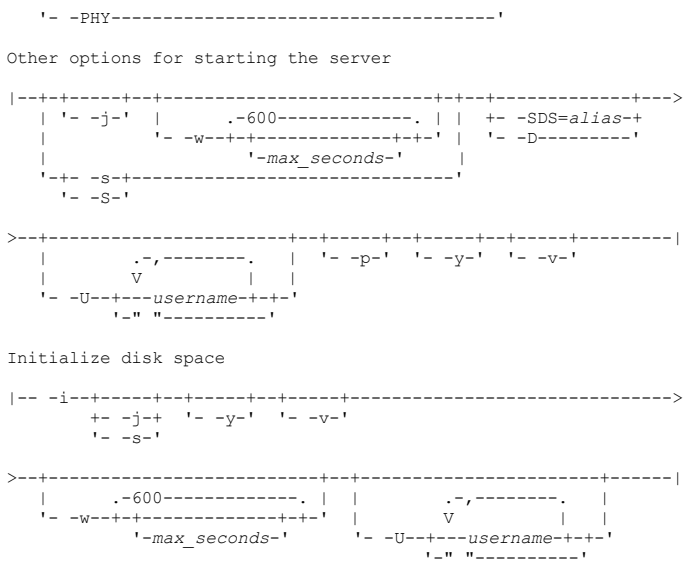
The **oninit** utility starts the database server.

On UNIX, Linux, and Mac OS X, you must be logged in as user **root**, user **informix**, or the non-root database server owner to run the **oninit** utility. User **informix** should be the only member of the group **informix**. Run the **oninit** command from the command line. You can allow users who belong to the DBSA group to run the **oninit** command. See Allow DBSA group users to run the oninit command (UNIX).

On Windows, IBM® Informix® runs as a Windows service. Any user who has appropriate permissions to start a Windows service is able to start the IBM Informix service. The Services control application runs the **oninit** utility with any options that you supply.

Syntax





Notes:

1. See The -FILE option.

Table 1. **oninit** command elements

Element	Purpose	Key Considerations
-D	Starts the database server with Enterprise Replication and high-availability cluster replication disabled.	
-i	Initializes disk space for the root dbspace so that it can be used by the database server and starts the database server.	Disk space needs to be initialized only once to prepare data storage for the server. By default, to prevent data loss, you cannot reinitialize disk space. To reinitialize disk space for an existing root dbspace, you must set the FULL_DISK_INIT configuration parameter to `1` and then run the **oninit -i** command. See Initialize disk space for the root dbspace.
-j	Starts the server in administration mode.	See Start the server in administration mode.
-p	Starts the database server without deleting temporary tables.	If you use this option, the database server starts more rapidly, but space used by temporary tables left on disk is not reclaimed.
-PHY	Starts the server as of most current checkpoint. The **-PHY** option is used to tell the server to do only physical recovery without logical recovery.	This option is normally used to start a secondary server. You must run one of the following commands to connect the secondary server to the primary server: `onmode -d secondary` `onmode -d RSS` The connection of the secondary server to the primary server fails if the most recent checkpoint on the primary server was not performed on the secondary server.
-s	Starts the server in quiescent mode.	The database server must be shut down when you use this option. When the database server is in quiescent mode, only the user **informix** can access the database server.
-S	Starts database server in quiescent mode as a standard server with high-availability data replication disabled.	When the database server is in quiescent mode, only the user **informix** can access the database server.
-SDS=*alias*	For shared disk servers, starts the current server and specifies the primary server with the alias name.	When both the primary server and all of the SDS servers are down, use the **-SDS=***alias* option to start the designated SDS server as the primary server. The **-SDS=***alias* flag cannot be combined with the **-i** flag.
-U *username*	Specifies which users can access the server in administration mode for the current session.	The **informix** user and members of the DBSA group are always administration mode users. See Start the server in administration mode.
-v	Displays verbose informational messages while the server is starting.	
-w *max_seconds*	Starts the database server and waits to indicate success or failure until the server is completely started in online mode or the number of seconds specified by *max_seconds* elapses.	The default number of seconds to wait is 600. This option is not valid on secondary servers in a high-availability cluster. See Start the server with a script.
-y	Prevents verification prompts.	The **-y** option automatically answers yes to all the verification prompts.

Usage

By default, the **oninit** utility shows verification prompts during server startup. You can suppress verification prompts by including the **-y** option. You can view verbose informational messages by including the **-v** option. On UNIX, Linux, and Mac OS X, **oninit** output is shown to standard output. On Windows, you can view **oninit** output by setting the ONINIT_STOUT environment variable to save the output to a file.

You can start the server in different operating modes. By default, if you run the **oninit** command without options, the server starts in online mode. When the database server is in online mode, all authorized users can access the server.

If you run an **oninit -FILE** command, you do not need to set local environment variables before you start the database server. The database server automatically uses the environment variables that are set as values in the **onconfig** file.

Start the server in administration mode

Administration mode is an administrator-only mode you can use to perform maintenance operations including those that require running SQL or DDL commands. When in administration mode, the database server only accepts connection requests from the following users:

- The **informix** user
- Members of the DBSA group
- Users specified by the **oninit -U** command or the **onmode -j -U** command, for the current session. The **-U** option overrides any users listed by the ADMIN_MODE_USERS configuration parameter in the onconfig file.
- Users specified by the ADMIN_MODE_USERS configuration parameter

Use the **-U** option with a list of comma-separated user names to add administration mode users, such as: `Karin,Sarah,Andrew`.

Use the **-U " "** option to remove all administration mode users except the **informix** user and members of the DBSA group: **oninit -U " "**.

Initialize disk space for the root dbspace

The first time you install IBM Informix on your system, disk space for the root dbspace for the database server needs to be initialized. The root dbspace is specified by the ROOTPATH configuration parameter.

If you performed a typical installation and chose to create a database server or you performed a customer installation, disk space was automatically initialized. Otherwise, you must initialize disk space by running the **oninit -i** command.

If the DISK_ENCRYPTION configuration parameter is set when you initialize the root dbspace, the root dbspace is encrypted.

If necessary, you can reinitialize disk space. Reinitializing disk space destroys all existing data managed by the database server. The database server must be offline when you reinitialize.

By default, you cannot reinitialize a root dbspace that is being used by the database server. Disk initialization fails if a page zero exists at the root path location (at the first page of the first chunk location). You can allow disk reinitialization of an existing root dbspace by setting the FULL_DISK_INIT configuration parameter to `1`.

Start the server with a script

You can use the **oninit -w** command in customized startup scripts and to automate startup. The **-w** option forces the server to wait until startup is completely successfully before indicating that the server is in online mode by returning to the shell prompt with a return code of `0`. If the server is not in online mode within the timeout period, the server returns a return code of `1` to the shell prompt and writes a warning message in the online log.

The default timeout is 600 seconds (10 minutes), which you can modify to any integer value.

After running the following command, if the server fails to start within 60 seconds, a code of `1` is returned to the prompt:

```
oninit -w 60
```

To determine the reason for the server failing to start, check the online log. You might need to increase the timeout value. When you use the **oninit -w** command in a script, you can check whether the server is online with the **onstat -** (Print output header) command.

Allow DBSA group users to run the oninit command (UNIX)

To allow users who belong to the DBSA group, other than the user **informix**, to run the **oninit** command, log in as the user **root** and change the permissions on the **oninit** utility in the $INFORMIXDIR/bin directory from 6754 to 6755.

- The -FILE option
  On UNIX, you can use the **-FILE** option to run certain IBM Informix utilities with the local environment variables that you set in your onconfig file. You do not have to set local environment variables before you run the command to start the utilities.
- Return codes for the oninit utility
  If a **oninit** command encounters an error, the database server returns an error message and a return code value.

Related reference:
ADMIN_MODE_USERS configuration parameter
FULL_DISK_INIT configuration parameter
Database server files
SDS_ALTERNATE configuration parameter
DISK_ENCRYPTION configuration parameter
Related information:
Initialization process
Database server operating modes

The -FILE option

On UNIX, you can use the **-FILE** option to run certain IBM® Informix® utilities with the local environment variables that you set in your onconfig file. You do not have to set local environment variables before you run the command to start the utilities.

You can use the **-FILE** option when you start the following utilities: **oninit**, **oncheck**, **onclean**, **onload**, **onunload**, **onlog**, **onmode**, **onparams**, **onspaces**, **onstat**, and **ontape**.

Syntax

```
-FILE option

|--+---------+--+-----------+-----------------------------------|
   '- -FILE=-'  '-file_name-'
```

Table 1. **-FILE** option

Element	Purpose	Key Considerations
-FILE=*file_name*	Specifies the full path or relative path to the onconfig file that contains the environment information.	The **-FILE=***file_name* option must be the first argument in the command.

Usage

Before you run a command with the **-FILE** option, you must add directives to your onconfig file in the following format:

```
#$variable_name value
```

Any environment variables that are set in the onconfig file take precedence over the same environment variables that are set in the system or shell.

When you start a utility with the **-FILE** option, specify the full path or the relative path to the onconfig file. For example, both of the following examples start the database server with the environment information in the onconfig.serv1 file:

Full path

```
oninit -FILE=/opt/IBM/inf/etc/onconfig.serv1
```

Relative path

```
oninit -FILE=etc/onconfig.serv1
```

If the INFORMIXDIR environment variable is not set in the user system, the shell, or in the onconfig file, the value of INFORMIXDIR is set to the PATH of the executable program, with the assumption that the executable program is in a subdirectory of INFORMIXDIR. For example, you can run the **oninit -FILE=etc/onconfig.myserv** command when the **oninit** utility is in the /opt/IBM/informix/bin directory. If the INFORMIXDIR environment variable is not set in the shell or in the onconfig.myserv file, the value of INFORMIXDIR is set to /opt/IBM/informix.

If you use a form of remote execution, such as `ssh`, use the **-FILE** option to specify the path to the onconfig file on the remote computer.

Example

Suppose that you specified values for the INFORMIXSERVER, DBDATE, and SERVER_LOCALE environment variables in the onconfig file for the js_3 instance:

```
#onconfig.js_3
#
# *** Start environment settings for js_3
#
#$INFORMIXSERVER server3
#$DBDATE MDY4/
#$SERVER_LOCALE en_us.utf8
#
# *** End environment settings for js_3
```

The other important environment variables (INFORMIXDIR, INFORMIXSQLHOSTS, ONCONFIG) for running the utility are specified in the user environment. The path to the **oninit** executable program is part of the user environment and the onconfig file is in the current directory.

You can run the `oninit -FILE=onconfig.js_3` command from the current directory to start the database server, and automatically set the values for the INFORMIXSERVER, DBDATE, and SERVER_LOCALE environment variables.

Related tasks:
Setting local environment variables for utilities

Return codes for the oninit utility

If a **oninit** command encounters an error, the database server returns an error message and a return code value.

The following table contains the return codes, message text, and user actions for the **oninit** utility.

Table 1. Return codes for the oninit utility

Return Code	Message Text	User Action
0	The database server was initialized successfully.	The database server started.
1	Server initialized has failed. Look at any error messages written to stderr or the online message log.	Take the appropriate action based on the error messages written to stderr or the online message log.
87	The database server has detected security violations or certain prerequisites are missing or incorrect.	(UNIX and Mac OS only) Check if user and group **informix** exists. Check if the server configuration file (onconfig) and sqlhosts file exists and has the correct permissions. Check if the environment variables INFORMIXDIR, ONCONFIG, and SQLHOSTS have a valid value and their length does not exceed 255 characters. Check if the environment variable INFORMIXDIR specifies an absolute path and does not have any spaces, tab, new lines, or other incorrect characters. Check if role separation-related subdirectories under the $INFORMIXDIR directory, such as aaodir and dbssodir, have the correct ownership. Run the **onsecurity** utility to diagnose and fix any issues.
170	The database server failed to initialize the dataskip structure.	Free some physical memory on the system and try to start the database server again.
172	The database server failed to initialize the listener threads.	Free some system resources, check the configuration parameter values for the number of listener threads to start when the database server starts up, and try to start the database server again.
173	The database server failed to initialize data replication.	Free some physical memory in the system and try to start the database server again.
174	The database server failed to start fast recovery threads.	Free some physical memory in the system and try to start the database server again.
175	The database server failed to initialize the root dbspace.	Check the root dbspace related parameters in server configuration file (onconfig) to make sure that the path for the root dbspace is valid.
176	Shared disk secondary server initialization failed.	Check the entries in sqlhosts file (UNIX) or SQLHOSTS registry key (Windows) to make sure that you are using the value of the DBSERVERNAME configuration for the primary server correctly. Check if the value for the SDS_PAGING configuration parameter in the server configuration file (onconfig) is correct. Free some system resources and try to start the database server again.
177	The database server failed to start the main_loop thread.	Free some physical memory on the system and try to start the database server again.
178	The database server failed to initialize the memory required for page conversion.	Free some physical memory on the system and try to start the database server again.
179	The database server was unable to start CPU VPs.	Free some physical memory on the system and try to start the database server again.
180	The database server was unable to start the ADM VP.	Free some physical memory on the system and try to start the database server again.
181	The database server failed to initialize kernel AIO.	Free some physical memory on the system and try to start the database server again.
182	The database server was unable to start IO VPs.	Free some physical memory on the system and try to start the database server again.
183	The database server failed to initialize the memory required for asynchronous I/O operations.	Free some physical memory on the system and try to start the database server again.
184	The database server failed to initialize memory required for parallel database queries. (PDQ)	Free some physical memory on the system and try to start the database server again.
185	The database server failed to initialize various SQL caches.	Free some physical memory on the system and try to start the database server again.
186	The database server failed to initialize the Global Language Support (GLS) component.	Free some physical memory on the system and try to start the database server again.
187	The database server failed to initialize the Associated Service Facility (ASF) components.	Check the entries in sqlhosts file.
188	The database server was unable to start the CRYPTO VP.	Free some physical memory on the system and try to start the database server again.
189	The database server was unable to initialize the alarm program.	Free some physical memory on the system and try to start the database server again.
190	The database server failed to initialize the auditing component.	Free some physical memory on the system and try to start the database server again.
192	The database server failed to restore the Window station and desktop.	(Windows only) Try to shut down the database server after freeing some system resources.
193	The database server failed to create daemon processes.	(UNIX and Mac OS only) Free some system resources and try to startup the database server once again.
194	The database server failed to redirect the file descriptors properly.	(UNIX and Mac OS only) Check the availability of the /dev/null device and try to start the database server again.

Return Code	Message Text	User Action
195	The database server failed to initialize the current directory for use.	Check the validity of the current working directory from where the database server is being initialized.
196	The database server failed to initialize the /dev/null device.	(AIX® only) Check the validity of the /dev/null device.
197	The database server failed to find the password information for the user trying to initialize the database server.	Verify that the user password is valid.
198	The database server failed to set the resource limits.	(UNIX and Mac OS only) Verify, and if required, increase the resource limits for processes on the host computer.
200	The database server did not have enough memory to allocate structures during initialization.	Free some physical memory on the system and try to start the database server again.
206	The database server could not allocate the first resident segment.	Check the values of the BUFFERPOOL and LOCKS configuration parameters in the server configuration file (onconfig) to make sure that they can be accommodated with the available memory on the host computer.
207	The database server failed to initialize shared memory and disk space.	Free some physical memory in the system, check the validity of all the chunks in the database server, and try to start the database server again.
208	The database server failed to allocate structures from shared memory.	Free some system resources and try to start the database server again.
209	The database server encountered a fatal error during the creation of shared memory.	Free some physical memory in the system and try to start the database server again.
210	The database server requested memory for the resident segment that exceeded the maximum allowed.	Reduce the size of the resident segment by lowering the values of the BUFFERPOOL and LOCKS configuration parameters.
220	The database server failed to read the audit configuration file.	Check that the audit configuration file (adtcfg) exists and is valid.
221	The database server could not detect the default directory for DUMPDIR. Usually it is the $INFORMIXDIR/tmp directory.	Create the $INFORMIXDIR/tmp directory if it is not present.
222	The database server detected an error in the value of the DBSERVERALIASES configuration parameter in the server's configuration file.	Verify that the values for the DBSERVERALIASES configuration parameter are valid and they have corresponding entries in the sqlhosts file (UNIX) or SQLHOSTS registry key (Windows).
223	The database server detected an error with the value of the DBSERVERNAME configuration parameter in the server's configuration file.	Verify that the value of the DBSERVERNAME configuration parameter is valid and it has a corresponding entry in the sqlhosts file (UNIX) or SQLHOSTS registry key (Windows).
224	The database server detected an error with the value of the HA_ALIAS configuration parameter in the server's configuration file.	Correct the value of the HA_ALIAS configuration parameter in the server configuration file (onconfig).
225	The database server detected too many entries for the NETTYPE configuration parameter or the DBSERVERALIASES configuration parameter in the server's configuration file.	Reduce the number of instances of the NETTYPE or DBSERVERALIASES configuration parameters in server configuration file (onconfig) and try to start the database server again.
226	The database server could not find an entry for the DBSERVERNAME configuration parameter in the sqlhosts file or the contents of the sqlhosts file are not valid.	Check the entries in the sqlhosts file.
227	Incorrect serial number.	Reinstall the database server.
228	The user does not have the necessary DBSA privileges to invoke the executable.	The user must have DBSA privileges or be a part of the Informix®-Admin group (Windows).
229	The database server could not initialize the security sub-system.	(Windows only) The user does not the necessary user rights on the host or is not part of the Informix-Admin group.
230	The database server, if started as a process on Windows platform , timed out while trying to build the required system databases during initialization. (Windows only)	Check the event log on the host to determine why the service could not be opened or could not be started. The database server might have timed out while trying to build the system databases. Free some system resources and try to start the database server again.
231	Informix service startup failed when the “oninit -w” command was run as a process on the command line.	(Windows only) Check the event log on the host to determine why the service start has failed.
233	The database server failed to initialize the Pluggable Authentication Module (PAM).	Check the configuration for the PAM library on the system.
235	The database server detected errors for certain configuration parameter values in the server's configuration file.	Inspect the server configuration file (onconfig) for any errors.
236	The database server detected an error while trying to restrict the allowable values for the Informix edition in use.	Check if the SDS_ENABLE configuration parameter is set to 1 in the server configuration file (onconfig). Check if the server name specified with the **oninit -SDS** command matches the value of the HA_ALIAS or DBSERVERNAME configuration parameter. Check if the shared disk used is part of an existing shared disk cluster.
237	The database server could not find the server configuration file.	Ensure that the server configuration file exists and is valid.
238	The database server detected an incorrect value for the INFORMIXSERVER environment variable or the value did not match the value of the DBSERVERNAME configuration parameter in the server's configuration file.	(Windows only) Check the value of the INFORMIXSERVER environment variable and the corresponding entry in the registry.

Return Code	Message Text	User Action
239	The database server detected an incorrect or non-existent value for the INFORMIXDIR environment variable.	(Windows only) Check the value of the INFORMIXDIR environment variable.
240	Incorrect command-line options were issued to the database server.	Correct the command-line options issued to the database server at startup.
248	The database server failed to create the Informix loader domain file.	(AIX only) Check if the /var/adm/ifx_loader_domain file is present.
249	The database server failed to dynamically load the PAM library.	The PAM library is not available for the database server. Install the PAM libraries.
250	The database server failed to dynamically load the ELF library.	The ELF library is not available to the database server. Install the **libelf** packages.
255	There was an internal error during server initialization. Look at any error messages written to stderr or to the online message log.	Take the appropriate action based on the error messages written to stderr or the online message log.

The onkstash Utility

Use the **onkstash** utility to create a password stash file for an existing PKCS#12 keystore.

A password stash file allows database clients or the database server itself access to their respective keystore without the inconvenience for the user to supply the password every time.

The **onkstash** utility accepts the file name of a PKCS#12 keystore (ending with extension ".p12") and the password for this keystore. It writes the password in an encrypted format to the password stash file. The name of this stash file is same as the keystore filename, but with the extension ".stl".

If the password for a keystore gets changed, the new password must be stashed again using the **onkstash** utility. If a password stash file exist with the old keystore password, then it is overwritten with the new password in an encrypted format.

Syntax

```
onkstash <keystore file> <password>
```

where **<keystore file>** is the name of the PKCS#12 keystore file, and **<password>** is the current password for the keystore.

Usage

The **onkstash** utility determines the file name for the password stash file from the name of the keystore file. It checks if the given password is correct and then writes it in an encrypted format to the stash file.

If the password stash file gets created by **onkstash**, the file access permissions are set to 600. If the password stash file already exists, the permissions are not changed. It is recommended to check the permissions for the keystore file as well as for the password stash file, and correct them if deemed necessary.

In addition to stashing the keystore password, the **onkstash** utility can also provide the version of the encryption library used. Run the command "onkstash -version" to determine, whether GSKit or OpenSSL is used as encryption library and the version of that library.

The onkstore Utility

Use the **onkstore** utility to create and manage password stash files for use with storage space encryption and the integrated backup encryption features.

The onkstore utility will create a password stash file in the $INFORMIXDIR/etc directory by default, but this file may be created and used from any location accessible by the database server as long as that directory has secure permissions.

With its informix/informix ownership and 600 permissions, the password stash file can be read only by users root or informix in UNIX/Linux and the creator of the keystore in Windows. In addition, the file is itself encrypted using a password. The admin must specify this *keystore password* when creating the keystore. By default that password will be stored (as an obfuscated value) in a stash file along side the keystore file. Do not remove the stash file or allow it to be separated from the keystore file. If you do not want the password to be stashed, use the option "-nostash" when creating the keystore. In that case the password may be supplied interactively to oninit and utilities such as *oncheck*, *onlog*, *ontape*, or *onbar*.

The onkstore utility can create different types of keystores. A keystore can contain either:

1. A *Master Encryption Key* (MEK) that is used as a “seed” by the server to encrypt storage spaces when using the Storage Space Encryption feature.
2. A set of credentials to access a Remote Key Server that stores the Master Encryption Key for the Storage Space Encryption (DISK_ENCRYPTION) or a set of credentials to access a Remote Key Server that stores the Remote Master Encryption Key used by the Integrated Backup Encryption feature (BAR_ENCRYPTION).

The onkstore utility has the following usage:

Table 1. onkstore usage

-file <fn>	name of keystore to create/list/convert.
-type	type of keystore to create: local, AWS-EAR, AWS-BAR, KMIP, AZURE-EAR, AZURE-BAR

-create	create a new keystore. By default stash the password in a stash file. Use option "-nostash" if this is not desired.
-pw <fn>	file with cleartext keystore password. If not provided and the password is not stashed already, it is prompted for interactively.
-list	list the contents of the file.
-cipher	cipher the server will use: aes128, aes192, aes256
-credential <fn>	file that contains credentials in json format.
-verify	verify the keystore.
-convert	convert keystore from one type to another.
-changepw [<fn>]	change the password for the keystore. <fn> is the file containing the new cleartext keystore password.
-nostash	upon creation of a keystore do not stash the password.
-help	print this message.

Note: -pw is not needed if your password is stashed.
Use the onkstore utility to perform the following tasks:

- Create a Keystore with onkstore
- Creating an AWS type keystore
- Creating an Azure type keystore
- Creating a KMIP type keystore
- Verifying a Keystore File
- Changing the Password for a Keystore File
- Converting a Keystore File
- List the contents of a Keystore File

Create a Keystore with onkstore

A keystore file is required by any instance that has the storage space encryption feature enabled. This keystore file has a ".p12" extension. It may also have an associated password stash file whose extension is ".sth".

When referring to a keystore file with onkstore or in the value of the DISK_ENCRYPTION configuration parameter, always omit the ".p12" extension.

A keystore file that contains your instance's encryption key is called a local keystore file. The simplest way to create a local keystore file is as follows:

```
onkstore -create -file my_keystore -type local -cipher aes128
```

The result of that command is a file located in the $INFORMIXDIR/etc directory called my_keystore.p12, which contains a 128-bit (16 byte) encryption key. That p12 file is encrypted using a password, which must be provided interactively when prompted for. By default, the password is stored in a stash file. The path to the stash file is $INFORMIXDIR/etc/my_keystore.sth.

To provide the new password on the command line when creating a new keystore, use these commands instead:

```
echo "sample_password" > pw_file
onkstore -file my_keystore -type local -cipher aes128 -pw pw_file
rm pw_file
```

The password must be at least 8 characters long. In this case "sample_passwd" would also be stashed encrypted in $INFORMIXDIR/etc/my_keystore.sth.

As the encryption password is known, the admin has the option of removing the stash file and supplying the password to **oninit** manually each time the server is booted:

```
oninit -pw
Please enter current encryption password: sample_password
```

Instead of supplying the password interactively, it may be passed to oninit using a file:

```
touch /tmp/mypassword
chmod 660 /tmp/mypassword
echo "sample_password" > /tmp/mypassword
oninit -pw /tmp/mypassword
rm /tmp/mypassword
```

The keystore file will be located in $INFORMIXDIR/etc by default, but you can also move or create it elsewhere by specifying a full path (minus the .p12 extension):

```
onkstore -create -file /work/KEYSTORES/my_keystore -type local -cipher aes128
```

If your keystore file is not located in $INFORMIXDIR/etc you must use the full path in your DISK_ENCRYPTION setting:

```
DISK_ENCRYPTION keystore=/work/KEYSTORES/my_keystore
```

Like $INFORMIXDIR/etc, the directory containing your keystore file must have ownerships of informix/informix.

When creating a keystore file with onkstore you must specify which of the three supported ciphers you wish to use: aes128, aes192, and aes256. By default the server assumes you are using aes128, but if not, the admin must specify the cipher in the DISK_ENCRYPTION setting:

```
DISK_ENCRYPTION keystore=my_keystore,cipher=aes256
```

The DISK_ENCRYPTION setting consists of comma-separated attributes and may contain no quotes or spaces.

A keystore file that contains AWS (Amazon Web Services) credentials instead of an encryption key is called a remote keystore file. Run the following command to create a remote keystore file interactively:

```
onkstore -create -file my_aws_keystore -type AWS_EAR -cipher aes192
```

onkstore will then prompt you for AWS credentials and other information that will identify the key you want to either create or use. For example:

```
 $ onkstore -create -file my_aws_keystore -type AWS_EAR -cipher aes192
Creating AWS EAR Keystore
AWS Key Id
>AKCAIPP520LF4AJBOTXA
AWS Key Secret
>TCEmlasjdflkjbasNHFAI6BHOwj4XHe50ic7LCt9
AWS Region
>us-east-1
AWS CMK Id
>16fd15d9-db8b-4cb7-9d99-d3070df97b58
SSM Key Location
>/informix/keys/aes192/key1
```

This is not your actual encryption key. They are merely pieces of information that when put together allow the server to access a particular encryption key stored in AWS. If the terms “CMK Id” and “AWS Region” are not familiar to you, it is because you do not yet have an AWS account set up. Familiarity with an AWS account you are able to manage is a prerequisite for creating a remote keystore file using onkstore.

Rather than providing these details to onkstore interactively you have the option of feeding a json file to the utility instead:

```
onkstore -create -file my_ks -cipher aes192 -credential /tmp/my_creds.json
```

In this case the /tmp/my_creds.json file would contain something like this:

```
{
"Credentials" :
{
"Type" : "aws-ear",
"AWS Key Id" : "AKCAIPP520LF4AJBOTXA",
"AWS Key Secret" : "TCEmlasjdflkjbasNHFAI6BHOwj4XHe50ic7LCt9",
"AWS Region" : "us-east-1",
"AWS CMK Id" : "16fd15d9-db8b-4cb7-9d99-d3070df97b58",
"SSM Key Location" : "/informix/keys/aes192/key1"
}
}
```

If this command is run and the master encryption key does not exist in AWS at the specified location (/informix/keys/aes192/key1), onkstore will attempt to generate one and store it there. If the credentials point to an existing key, onkstore will create the keystore file and leave the key as-is.

The -pw argument works the same way with remote keystore file creation as it does with local keystore creation.

Do not use the AWS-BAR type when creating a keystore for use with the storage space encryption feature. This type of keystore is used with the Integrated Backup Encryption feature.

Creating an AWS type keystore

If your remote key server is Amazon Web Services Key Management Service (AWS-KMS), you can create two types of keystore : “AWS-EAR” to be used by the Storage Space Encryption feature, or “AWS-BAR” to be used by the Integrated Backup Encryption feature.

The only difference between these credentials is that the AWS-EAR requires, also, access to the AWS Secrets Manager (AWS-SSM) where the IDS Master Encryption Key is stored.

When asked to create an AWS keystore, the following information must be readily available to the operator:

- AWS Key Id, this is not an encryption key, this is the AWS access key (the equivalent of a username) to get access to the AWS infrastructure. You can generate a “AWS Key Id”/”AWS Key Secret” pair (**AWS Services -> IAM -> Users -> “User Name” -> Security Credentials -> Access Keys**).
- AWS Key Secret, this is not an encryption key, this is the AWS secret key (the equivalent of a password) to get access to the AWS infrastructure. You can generate a “AWS Key Id”/”AWS Key Secret” pair (**AWS Services -> IAM -> Users -> “User Name” -> Security Credentials -> Access Keys**).
- AWS Region, this is the AWS region, all keys and CMKs are region bound, you must provide the region where you have your keys (ie us-east-1).
- AWS CMK Id, this is the id (or name) of the AWS Customer Master Key (Remote Master Encryption Key). This key never leaves the AWS infrastructure and it is used by onkstore to generate the master encryption key used by IDS with the Storage Spaces Encryption feature or the Backup Encryption Keys used by the Integrated Backup Encryption feature.
- SSM Key Location, this is needed only for AWS-EAR types of keystores. This is the hierarchical path where the IDS Master Encryption Key will be stored after being generated by onkstore. The MEK is encrypted using the CMK before being stored here.

To use a JSON file as input for onkstore, create a file with the following structure:

```
            {
              "Credentials" :
                {
                "Type" : "..." ,
                "AWS Key Id" : "...",
                "AWS Key Secret" : "...",
                "AWS Region" : "...",
                "AWS CMK Id" : "...",
                "SSM Key Location" : "..."
                }
            }
          where the value for "Type" is either "aws-ear" or "aws-bar".
```

Creating an Azure type keystore

If your remote key server is Microsoft Azure Key Vault you can create two types of keystore : “AZURE-EAR” to be used by the Storage Space Encryption feature, or “AZURE-BAR” to be used by the Integrated Backup Encryption feature.

The only difference between these credentials is that the AZURE-EAR requires, also, access to the Azure Secrets inside the Key Vault, where the IDS Master Encryption Key is stored.

When asked to create an AZURE keystore, the following information must be readily available to the operator:

- Azure Vault Url, this is the URL to access your key vault. This value is generated by the Azure system when a new keyvault is created (**Home -> All resources -> “Key Vault Name” -> Overview -> DNS Name**).
- Azure Client Id, the usage of this feature requires access to the Active directory infrastructure in Azure,for that you need to create a Web Application under your username and provide the “Application Id” (**Active Directory -> Users -> All User -> “User Id” -> Applications -> “Application Name” -> Application Id**).
- Azure Client Secret, When the Web Application is created, you will be provided with both the Application Id and the Application Secret. The application secret cannot be recovered after the application was created.
- Azure Directory Id, Your KeyVault is created under an Active Directory, you need to provide the Directory Id (**Home -> All resources -> “Key Vault Name” -> Overview -> Directory ID**).
- Azure Key Name, this is the name or full id of the Azure Key (Remote Master Encryption Key). This Key never leaves the Azure infrastructure and it is used by onkstore to encrypt the locally generated Master Encryption Key used by the IDS Storage Space Encryption feature. It is also used by the On-Bar/ontape utilities to encrypt the Backup Encryption Keys used by the Integrated Backup Encryption Feature. In Azure, you can provide a simple name for this key (ie “MY_IDS_MEK”) in which case we will use the LATEST key available (Each time the key is rotated a newer Id is available), or, you can specify the Id of the key you want to use (ie “ MY_IDS_MEK/wsdd6405fb584cf9a3c63f6926d2e92e”) in which case we will keep using the same key even if it is rotated.
- Azure Encrypt Algorithm, when you create the RMEK in Azure Key Vault, it allows you to select among several types of keys and depending on the type of key, you can select different algorithms to encrypt data with it. Select here a valid algorithm name for the type of RMEK you created.
- Azure Secret Name, The name of the secret where we will store the IDS MEK. This is used only if you create a AZURE-EAR type keystore. If you provide a simple name (ie “ INFORMIX-256BIT”) a new MEK will be generated and stored, the ID of the newly stored key will recorded. If you provide a full ID for the secret (“ INFORMIX-256BIT/ 284ded569a8b40be8e4de2254ddeedd7”), then we will try to retrieve the secret, if not present we will return an error.

To use a JSON file as input for onkstore, create a file with the following structure:

```
{
  "Credentials" :
    {
    "Type" : "...",
    "Azure Vault Url" : "...",
    "Azure Client Id" : "...",
    "Azure Client Secret" : "...",
    "Azure Directory Id" : "...",
    "Azure Key Name" : "...",
    "Azure Encrypt Algorithm" : "...",
    "Azure Secret Name" : "..."
    }
}
```

```
where the value for "Type" is either "azure-ear" or "azure-bar".
```


Creating a KMIP type keystore

If your remote key server is located in a server/cluster supporting the KMIP standard you can create a single type of keystore (KMIP). At this moment, the same keystore type can be used by both the Storage Space Encryption and Integrated Backup Encryption features.

For Integrated Backup Encryption, this type of keystore works similarly with Azure and AWS: We provide the Key name of a RMEK that is used to encrypt the Backup Encryption Keys.

For Storage Space Encryption, the Key Name provided is the IDS MEK.

When asked to create a KMIP type keystore, following information must be readily available to the operator:

- KMIP Server, the IP address or hostname where the KMIP server is listening for request. If the port where the server listens is different from the default (5696), the port must be specified (ie “myserver.hcl.com:2356).
- KMIP Username, username to access the KMIP server. This is optional since in most cases, the access to the server is done by using SSL certificates.
- KMIP Password, password for the given username. This is also optional.
- KMIP Client Certificate File, a file containing the certificate for the client, The file must also contain the Private Key matching the certificate. The private key is expected to be a PKCS#8 key. The certificate is expected to have Authentication extensions.
- KMIP CA Certificate File, a file containing the root CA used to sign both the KMIP Client Certificate File and the KMIP Server Certificate File.
- KMIP Key Name, The name of the KMIP Key used as MEK by the Storage Spaces Encryption feature or as RMEK by the Integrated backup Encryption Feature. It is optional. If not present, onkstore will generate a new key and report its Id to the operator.

To use a JSON file as input for onkstore, create a file with the following structure:

```
{
  "Credentials" :
    {
    "Type" : "..." ,
    "KMIP Server" : "...",
    "KMIP Username" : "...",
    "KMIP Password" : "...",
    "KMIP Client Certificate File" : "...",
```

```
                "KMIP CA Certificate File" : "...",
                "KMIP Key Name" : "..."
                }
            }

          where the value for "Type" is "kmip".
```


Verifying a Keystore File

After creating a network keystore file and before deploying it, it is important to verify that its credentials work correctly. To do this, run the following command:

onkstore -file my_keystore -verify

onkstore will use the credentials contained in your keystore file to communicate with AWS and report success or failure:

```
$ onkstore -file my_keystore -verify
Keystore Verify Successful.
Key exists in AWS SSM /informix/keys/aes192/key1 for cipher aes192.
```


Changing the Password for a Keystore File

At any time using onkstore the admin may change the password of a keystore:

onkstore -file my_keystore -pw /tmp/old_password -changepw /tmp/new_password

If your current password is stashed (contained in the keystore's associated .sth file), then you do not need to pass it to onkstore via the -pw argument, but if you have removed the stash file you must provide the current password to onkstore before it can be changed to a new one.

You can also perform this same function using the "master_key reset" sysadmin command:

```
dbaccess sysadmin -<<END
execute function task("master_key reset","new_sample_pw");
END
```

This method has the advantage of not requiring the current password in order to change to a new one. The current password was provided to the server at boot time.

Converting a Keystore File

The convert feature is currently used only for EAR types of keystores. It supports to download the Master Encryption Key contained in the Remote Key Server (ie a KMIP server) to the local keystore. The old keystore containing the credentials to the RKS will be renamed and will be replaced with a new one of type "local".

Since the Integrated backup encryption feature does not store a Master Encryption Key at the RKS and does not support keystore of type "local", this option is not needed/supported for credentials of type AWS-BAR and AZURE-BAR.

```
$ onkstore -file my_keystore -convert
Which type of keystore would you like to create:
1 - Local Keystore
2 - AWS EAR Keystore
3 - AWS BAR Keystore
4 - KMIP EAR Keystore
5 - AZURE EAR Keystore
6 - AZURE BAR Keystore

Conversion complete for /vobs/tristarp/sqldist/etc/my_keystore.p12
```

Currently, only option 1 (converting to a local keystore file) is supported. The original keystore file is copied to a backup file (my_keystore.p12.bak#) before being overwritten during the conversion.

Note: By downloading your MEK to a local machine, you are increasing the chances of exposing that key, which is the reason to use a RKS in the first place.

List the contents of a Keystore File

This command will not display your encryption key or your AWS credentials. It displays the kinds of objects stored in the file. For example:

```
$ onkstore -file /work3/keystores/test_keystore -list
List the contents of keystore /work3/keystores/test_keystore.p12
KeystoreType
AWS Key Id
```

```
AWS Key Secret
AWS Region
AWS CMK Id
SSM Key Location
```

An admin may list the basic contents of a file as a sort of sanity check.

The onlog utility

The **onlog** utility displays the contents of a logical-log file, either on disk or on backup.

onlog: Display Logical-Log Contents

The **onlog** output is useful in debugging situations when you want to track a specific transaction or see what changes have been made to a specific tblspace. (For information about interpreting the logical-log file contents, see Interpreting Logical-Log Records.)

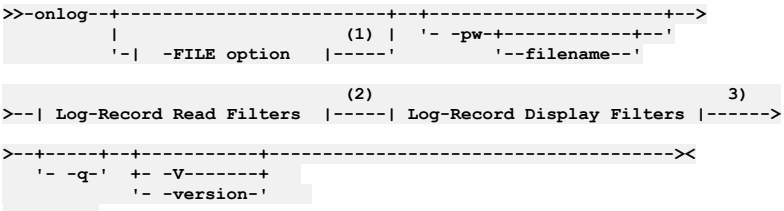
Any user can run all of the **onlog** options except the **-l** option. Only user **informix** on UNIX or a member of the **Informix-Admin** group on Windows can run the **-l** option.

If the database server is in offline mode when you execute **onlog**, only the files on disk are read. If the database server is in quiescent or online mode, **onlog** also reads the logical-log records stored in the logical-log buffers in shared memory (after all records on disk have been read).

When the database server reads a logical-log file with status U from disk while in online mode, the database server denies all access to the logical-log files, effectively stopping database activity for all sessions. (For more information, see onstat -l command: Print physical and logical log information.) For this reason, it is recommended that you wait until the files have been backed up and then read the contents of the logical-log files from backup.

The **onlog** utility does not have a functionally equivalent SQL administration API *command* string.

onlog Syntax



Note: The -pw option is required only when the Storage space encryption feature is enabled, stash file is in use, and the server is off-line. Supply an optional path to a file containing the keystore password, otherwise onlog will prompt for a password before displaying the requested log(s).

Element	Purpose	Key Considerations
-q	Suppresses the initial header and the one-line header that appears every 18 records by default	None.
-V	Displays the software version number and the serial number	See Obtaining utility version information.
-version	Displays the build version, host, OS, number and date, as well as the GLS version	See Obtaining utility version information.

You direct **onlog** to read the following portions of the logical log as it searches for records to display:

- Records stored on disk
- Records stored on backup media
- Records from the specified logical-log file

By default, **onlog** displays the logical-log record header, which describes the transaction number and the record type. The record type identifies the type of operation performed.

In addition to the header, you can use the read filters to direct **onlog** to display the following information:

- Logical-log record header and data (including copies of simple large objects stored in a dbspace or tblspace)
- Copies of blobpages from blobspaces
  They are copied from the logical-log backup only. They are not available from disk.

You can display every logical-log record header, or you can specify output based on the following criteria:

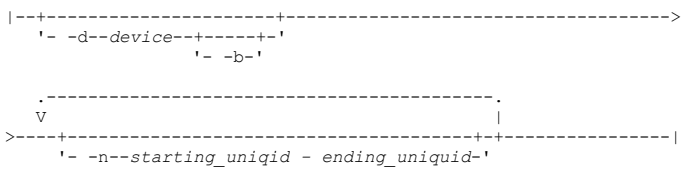
- Records associated with a specific table
- Records initiated by a specific user
- Records associated with a specific transaction

If **onlog** detects an error in the log file, such as an unrecognizable log type, it displays the entire log page in hexadecimal format and terminates.

Log-Record Read Filters

The **onlog** utility uses the pathnames that are stored in the root dbspace reserved pages to locate the logical-log files. If you use ON-Bar to back up the logical logs, **onlog** asks the storage manager to retrieve the appropriate logical-log records from the backup media.

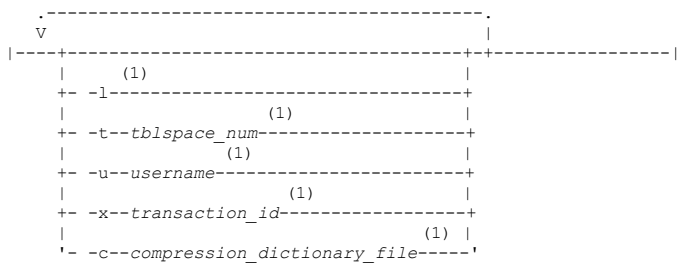
Syntax



Element	Purpose	Key Considerations
-b	Displays logical-log records associated with blobspace blobpages	The database server stores these records on the logical-log backup media as part of blobspace logging.
-d *device*	Names the pathname of the storage device where the desired logical-log backup is mounted	If you use **ontape**, the device that you name must be the same as the pathname of the device assigned to the configuration parameter LTAPEDEV. If the **-d** option is not used, **onlog** reads the logical-log files stored on disk, starting with the logical-log file with the lowest *logid*. If you use ON-Bar to back up logical logs, use the **onbar -P** command to view the contents of a logical-log file. See the *IBM® Informix Backup and Restore Guide*. For pathname syntax, see your operating-system documentation.
-n *starting_uniqid-ending_uniqid*	Directs **onlog** to read all the logical-log records contained in the log file that you specified from *starting uniqid* to the *ending uniqid*.	The *starting_uniqid* and the *ending_uniqid* are the unique ID numbers of the logical log. To determine the *uniqid* of a particular logical-log file, use the **onstat -l** command. If you do not use the **-n** option, **onlog** reads all the logical-log files that are available (either on disk or on tape). For information about the **onstat** utility, see Monitor the database server status.

Log-Record Display Filters

Syntax



Notes:

1. Only one occurrence of this item allowed

Element	Purpose	Key Considerations
-l	Displays the long listing of the logical-log record.	The long listing of a log record includes a complex hexadecimal and ASCII dump of the entire log record. The listing is not intended for casual use.
-t*tblspace_num*	Displays records associated with the tblspace that you specify.	Unsigned integer. Number, greater than 0, must be in the **partnum** column of the **systables** system catalog table. Specify this value as either an integer or hexadecimal value. (If you do not use a 0x prefix, the value is interpreted as an integer.) To determine the tblspace number of a particular tblspace, query the **systables** system catalog table as described in Tblspace Numbers.
-u *username*	Displays records for a specific user.	User name must be an existing login name. User name must conform to operating-system-specific rules for login name.
-x *transaction_id*	Displays only records associated with the transaction that you specify.	Value must be an unsigned integer between 0 and TRANSACTIONS - 1, inclusive. You should need to use the **-x** option only in the unlikely case that an error is generated during a rollforward. When this situation occurs, the database server sends a message to the message log that includes the transaction ID of the offending transaction. You can use this transaction ID with the **-x** option of **onlog** to investigate the cause of the error.

Element	Purpose	Key Considerations
-c *compression_ dictionary_file*	Uses the compression dictionary to expand compressed data and display uncompressed data.	If the **onlog** command contains the **-l** option and the **-c** option and there are compressed images in the log records, the **onlog** utility uses the compression dictionary to expand all expandable images in the log records. A compressed image is expandable only if there is a valid compression dictionary for that log record in the compression dictionary file. If **-c** is not specified or the compression dictionary file does not contain a valid compression dictionary for the compressed image, the **onlog** utility will display the row image in its compressed format.

If you do not have a compression dictionary file, you can use an UNLOAD statement to unload the compression dictionary, which is contained in the **syscompdicts_full** table in the **sysmaster** database, to a compression dictionary file, as follows:

```
UNLOAD TO 'compression_dictionary_file'
   SELECT * FROM sysmaster:syscompdicts_full;
```

If you do not specify any options, **onlog** displays a short listing of all the records in the log. You can combine options with any other options to produce more selective filters. For example, if you use both the **-u** and **-x** options, **onlog** displays only the activities that the specified user initiated during the specified transaction. If you use both the **-u** and **-t** options, **onlog** displays only the activities initiated by the specified user and associated with the specified tblspace.

Related reference:
alter logmode argument: Change the database logging mode (SQL administration API)

The onmode utility

Use the **onmode** utility to change the database server operating mode and perform various other operations on shared memory, sessions, transactions, parameters, and segments.

These topics show how to use the **onmode** options. If you do not use any options, the database server returns a usage statement.

On UNIX, you must be user **root** or user **informix** to run the **onmode** utility.

On Windows, you must be a member of the **Informix-Admin** group or the Administrators group to run the **onmode** utility.

For information on the **onmode -b** command, which is only used if you upgraded to a new version of Informix® and need to revert your databases to the previous version of the server, see Syntax of the onmode -b command in the *IBM® Informix Migration Guide*.

All **onmode** command options have equivalent SQL administration API *command* strings, except **onmode -b**, **onmode -BC**, and **onmode -R**.

- onmode command syntax
  Use **onmode** utility commands to perform various database server operations.
- onmode -a: Add a shared-memory segment
- onmode -BC: Allow large chunk mode
- onmode -c: Force a checkpoint
- onmode -C: Control the B-tree scanner
  Use the **onmode -C** command to control the B-tree scanner and specify information about B-tree scanner threads.
- onmode -cache surrogates: Cache the allowed.surrogates file
- onmode -d: Set data-replication types
- onmode -d: Set High Availability server characteristics
- onmode -d command: Replicate an index with data-replication
- onmode -D, -M, -Q, -S: Change decision-support parameters
- onmode -e: Change usage of the SQL statement cache
- onmode -F: Free unused memory segments
  Use the **onmode -F** command to free shared-memory segments that are unavailable or no longer needed for a process
- onmode -h: Update sqlhosts caches
- onmode -I: Control diagnostics collection
  Use the **onmode -I** option to start and stop diagnostics collection.
- onmode -k, -m, -s, -u, -j: Change database server mode
- onmode -l: Switch the logical-log file
- onmode -n, -r: Change shared-memory residency
- onmode -O: Override ONDBSPACEDOWN WAIT mode
- onmode -p: Add or drop virtual processors
  Use the **onmode -p** command to dynamically add or drop virtual processors for the database server instance. The **onmode -p** command does not update the onconfig file.
- onmode -P: Start, stop, or restart a listen thread dynamically
  Use the **onmode -P** command to start, stop, or restart an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.
- onmode -R: Regenerate .infos.dbservername File
- onmode -W: Change settings for the SQL statement cache
- onmode -we: Export a file that contains current configuration parameters
  Use the **onmode -we** command to create and export a configuration file that is a snapshot of your current configuration parameters.
- onmode -wf, -wm: Dynamically change certain configuration parameters
  Use the **onmode -wf** or **onmode -wm** command to dynamically change specific configuration parameters.
- onmode -wi: Import a configuration parameter file
  Use the **onmode -wi** command to import a file that contains new values for multiple configuration parameters. If the parameters are tunable, which means they can be updated individually with an **onmode -wm** command, the database server applies the new values.
- onmode -Y: Dynamically change SET EXPLAIN
- onmode -z: Kill a database server session
- onmode -Z: Kill a distributed transaction

onmode command syntax

Use **onmode** utility commands to perform various database server operations.

The following syntax diagram shows all of the options that you can use with the **onmode** command. The syntax diagram does not show all of the elements that you use with each command option. For the complete syntax of each command, see the topic on that command.

Syntax

```
>>-onmode--+------------------------+-------------------------->
           |                   (1) |
           '-|  -FILE option   |-----'

>--+- -a----------------+--+-----------+--+-----+-------------><
   |     (2)            |  +- -V-------+  '- -y-'
   +- -b----------------+  '- -version-'
   +- -BC---------------+
   +- -C----------------+
   +- -c----------------+
   +- -cache surrogates-+
   +- -e----------------+
   +- -d----------------+
   +-+- -D-+------------+
   | +- -M-+            |
   | +- -Q-+            |
   | '- -S-'            |
   +- -F----------------+
   +- -h-+---------+----+
   |     '- force -'    |
   +- -I----------------+
   +-+- -j-+------------+
   | +- -k-+            |
   | +- -m-+            |
   | +- -s-+            |
   | '- -u-'            |
   +- -l----------------+
   +-+- -n-+------------+
   | '- -r-'            |
   +- -O----------------+
   +- -P----------------+
   +- -p----------------+
   +- -R----------------+
   +- -W----------------+
   +-+- -wf-+-----------+
   | '- -wm-'           |
   +-+- -we-+-----------+
   | '- -wi-'           |
   +- -Y----------------+
   +- -Z----------------+
   '- -z----------------'
```

Element	Purpose	Key Considerations
-y	Causes the database server to automatically respond yes to all prompts	None.
-V	Displays the software version number and the serial number	See Obtaining utility version information.
-version	Displays the build version, host, OS, number and date, as well as the GLS version	See Obtaining utility version information.

onmode -a: Add a shared-memory segment

Syntax

```
>>-onmode-- -a--seg_size---------------------------------------><
```

Element	Purpose	Key considerations
-a *seg_size*	Allows you to add a new virtual shared-memory segment. Size is specified in kilobytes	Restriction: The value of ***seg_size*** must be a positive integer. It must not exceed the operating system limit on the size of shared-memory segments.

Ordinarily, you do not need to add segments to the virtual portion of shared memory because the database server automatically adds segments as they are needed. However, as segments are added, the database server might reach the operating-system limit for the maximum number of segments before it acquires the memory that it needs. This situation typically occurs when the SHMADD configuration parameter is set so small that the database server exhausts the number of available segments before it acquires the memory that it needs for some operation.

You can use this command to add a segment that is larger than the size specified by the SHMADD configuration parameter. By using this command to add a segment, you can adhere to the operating system limit for segments while meeting the need that the database server has for additional memory.

This command has an equivalent SQL administration API function.

Related reference:
add memory argument: Increase shared memory (SQL administration API)
onmode and a arguments: Add a shared-memory segment (SQL administration API)

onmode -BC: Allow large chunk mode

Syntax:

```
>>-onmode--+- -BC 1-+-------------------------------------------><
           '- -BC 2-'
```

Element	Purpose	Key Considerations
-BC 1	Enables support of large chunks, large offsets that are greater than 2 GB, and allows up to 32,768 chunks per instance.	This option allows large chunks to be created. Reversion without dropping the dbspace is possible if no chunks are larger than 2 GB. Dbspaces and blobspaces without chunks greater than 2 GB remain in the old format. After a chunk larger than 2 GB is added to a dbspace or blobspace then all chunks added or altered in that dbspace or blobspace are in the new format. See your *IBM® Informix Administrator's Guide*.
-BC 2	Allows large-chunk-only mode for all dbspaces.	Reversion is not possible. Enables the 9.4 large chunk feature for all dbspaces and blobspaces, Any chunk or offset added or modified has the new format. Existing chunks that you do not alter remain in the old format. See your *IBM Informix Administrator's Guide*

The **onmode -BC** (backward-compatible) commands are useful if you have converted from Informix® 9.40 (small chunk mode) to Informix 10.0 or later. When Informix 10.0 or later is first initialized (with the **oninit -iyv** command), by default it comes online with large chunk mode already fully enabled. Reversion is not possible. In the case of a newly initialized instance of Informix 10.0 or later, the **onmode -BC** commands will return an error.

Note: After executing the **onmode -BC** command, perform a complete system level-0 backup.

onmode -c: Force a checkpoint

Syntax:

```
>>-onmode-- -c--+--------------------+-------------------------><
               |        .- 15 ----. |
               +-block--+---------+-+
               |        '-timeout-' |
               '-unblock------------'
```

Element	Purpose	Key considerations
-c	Forces a checkpoint that flushes the buffers to disk.	You can use the **-c** option to force a sync checkpoint if the most recent checkpoint record in the logical log was preventing the logical-log file from being freed (status U-B-L).
block	Blocks the database server from any transactions.	While the database server is blocked, users can access it in read-only mode. Use this option to perform an external backup on IBM® Informix®. For more information, see the *IBM Informix Backup and Restore Guide*.
timeout	Specifies the number of seconds to wait for checkpoints to clear before returning to the command prompt.	The *timeout* option applies only if the DELAY_APPLY configuration parameter is configured (see DELAY_APPLY Configuration Parameter. If the DELAY_APPLY configuration parameter is enabled, the checkpoint requested by the primary server might not arrive at the secondary server for an extended period of time. It is also possible that no other checkpoints are staged in the staging directory. The default timeout value is 15 seconds and the maximum timeout allowed is 10 minutes (600 seconds). See *IBM Informix Backup and Restore Guide*.
unblock	Unblocks the database server.	When the database server is unblocked, data transactions and normal database server operations can resume. Use this option after you complete an external backup on IBM Informix. For more information, see the *IBM Informix Backup and Restore Guide*.

This command has an equivalent SQL administration API function.

Related reference:
checkpoint argument: Force a checkpoint (SQL administration API)
onmode and c arguments: Force a checkpoint (SQL administration API)

onmode -C: Control the B-tree scanner

Use the **onmode -C** command to control the B-tree scanner and specify information about B-tree scanner threads.

Syntax:

```
>>-onmode-- -C--+-(yielding syntax)--+------------------------><
                +-start--------------+
                +-count--------------+
                +-stop--count--------+
                +-kill    -count-----+
                +-threshold--size----+
                +-duration--num------+
                +-rangesize--size----+
                +-alice--mode--------+
                '-compression--value-'
```

Element	Purpose	Key considerations
-C	Controls the B-tree scanner for cleaning indexes of deleted items	There is no limit to the number of threads that can run at one time. However, there is a limit of 128 threads that can be started at one time. If, for example, you wanted 150 threads to run, you could execute two commands: **onmode -C 100** and **onmode -C 50.**
start *count*	Starts additional B-tree scanner threads.	If *count* is not specified, a *count* of 1 is assumed. There is no limit on the number of scanner threads that can be specified.
stop *count* **kill** *count*	Stops B-tree scanner threads.	If *count* is not specified, a *count* of 1 is assumed. Stopping all index scanners prevents all index cleaning. Either of these commands stop the B-tree scanner.
threshold size*count*	Sets the minimum number of deleted items an index must encounter before an index is placed on the hot list.	Once all indexes above the threshold have been cleaned and there is no other work for the B-tree scanner to do, the indexes below the threshold are added to the hot list.
duration *num*	The number of seconds that the hot list is valid.	After this number of seconds expires, the hot list will be rebuilt by the next available B-tree scanner thread, even if unprocessed items are on the list. Scanners currently processing requests are not interrupted.
rangesize *size*	Determines the size of an index before index range cleaning is enabled.	A size of `-1` can be used to disable range scanning.
alice *num*	Sets the system's **alice** mode.	Valid *num* values range from `0` (`OFF`) to `12`.
compression *value*	For a database server instance, modifies the level at which two partially used index pages are merged. The pages are merged if the data on those pages totals a set level.	Valid values for the level are `low`, `med` (medium), `high`, and `default`. The system default value is `med`.

The B-tree scanner has statistical information which tracks index efficiency and how much extra work the index currently places on the server. Based on the amount of extra work the index has accomplished because of committed deleted index items, the B-tree scanner develops an ordered list of indexes that have caused the server to do extra work. This list is called the hot list. The index causing the highest amount of extra work is cleaned first and the rest of the indexes are cleaned in descending order. The DBA can allocate cleaning threads dynamically, thus allowing for configurable workloads.

This command has an equivalent SQL administration API function.

Related reference:
onmode and C arguments: Control the B-tree scanner (SQL administration API)
set index compression argument: Change index page compression (SQL administration API)
BTSCANNER Configuration Parameter

onmode -cache surrogates: Cache the allowed.surrogates file

Syntax:

```
>>-onmode-- -cache surrogates-----------------------------------><
```

Element	Purpose	Key considerations
-cache surrogates	Reads the /etc/informix/allowed.surrogates file and stores the user IDs and group IDs values in shared memory cache. The user names and group names specified in allowed.surrogates file have to be valid operating system users and groups. The names are converted to corresponding UIDs and GIDs.	You can use **onmode -cache surrogates** during a session to load the allowed.surrogates file. The allowed.surrogates file is used specify users and groups who can act as surrogates for mapped users. The allowed.surrogates file will be automatically checked before a new connection is made to the database server or when users are created or altered. If the cache-refresh fails, the existing surrogate cache is cleared, effectively disabling mapped users. Existing connections on the server will be unaffected by changes in shared-memory cache. Changes in shared memory cache affect new sessions.

Related information:

Mapped user surrogates in the allowed.surrogates file (UNIX, Linux)
Specifying surrogates for mapped users (UNIX, Linux)
Surrogate user properties (UNIX, Linux)

onmode -d: Set data-replication types

Syntax:

```
>>-onmode-- -d--+-standard----------------+--------------------><
               '-+-primary---+--ha_alias-'
                 '-secondary-'
```

Element	Purpose	Key Considerations
-d	Used to set a server's data-replication type.	You can use the **-d standard** option when the database server is in quiescent, online, or read-only mode.
ha_alias	Identifies the high-availability alias of the primary or secondary database server.	The high-availability alias is the server's HA_ALIAS configuration parameter value. The *ha_alias* argument of the other database server in the data-replication pair and the database server's type (standard, primary, or secondary) is preserved after reinitialization of shared memory.

Using the -d standard option

The **-d standard** option drops the connection between database servers in a data replication pair (if one exists) and sets the database server type of the current database server to standard. This option does not change the mode or type of the other database server in the pair.

Use the **onmode -d standard** command only to disconnect a primary server from an HDR secondary server. Running the command converts the HDR secondary server to a standalone server. You should not run the **onmode -d standard** command to disconnect a primary server from an RS secondary server. To disconnect a primary server from an RS secondary server run the following commands:

On the RS secondary server:

```
onmode -d standard
```

On the primary server:

```
onmode -d delete RSS rss_ha_alias
```

Using the -d primary option

The **-d primary** option sets the database server type to primary and attempts to connect with the database server that *dbservername* specifies. If the connection is successful, data replication is turned on. The primary database server goes into online mode, and the secondary database server goes into read-only mode. If the connection is not successful, the database server is in online mode, but data replication is not turned on.

Using the -d secondary option

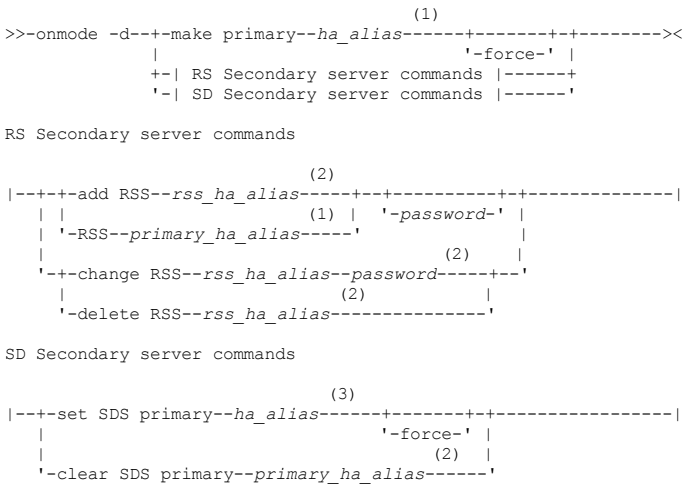
The **-d secondary** option sets the database server type to secondary and attempts to connect with the database server that *ha_alias* specifies. If the connection is successful, data replication is turned on. If the primary database server goes online, and the secondary database server goes into read-only mode. If the connection is not successful, the database server is in read-only mode, but data replication is not turned on.

This command has an equivalent SQL administration API function.

Related reference:
ha set primary argument: Define an HDR primary server (SQL administration API)
ha set secondary argument: Define an HDR secondary server (SQL administration API)
ha set standard argument: Convert an HDR server into a standard server (SQL administration API)
onmode and d arguments: Set data-replication types (SQL administration API)
HA_ALIAS configuration parameter
DBSERVERALIASES configuration parameter
DBSERVERNAME configuration parameter
onmode -d: Set High Availability server characteristics
onmode -d command: Replicate an index with data-replication
Related information:
Connection information set in the HA_ALIAS configuration parameter

onmode -d: Set High Availability server characteristics

Syntax:



Notes:

1. Run on secondary server only.
2. Run on primary server only.
3. Run on primary server or secondary server.

Element	Purpose	Key Considerations
-d	Used to create, modify, or delete secondary servers in high-availability configurations	
add RSS	Adds an RS secondary server	This command should be run on the primary database server.
rss_ha_alias	Identifies the RS secondary database server's high-availability alias.	The value can be an HA_ALIAS value or an ER group name.
password	Specifies the secondary server password	The password is used only during the first connection attempt. After the primary and secondary server have connected, the password cannot be changed.
RSS	Sets an RS secondary server type	This command should be run on the secondary database server.
pri_ha_alias	Identifies the name of the primary server	
change RSS	Change an RS secondary server	This command should be run on the primary database server.
delete RSS	Removes an RS secondary server definition	This command should be run on the primary database server.
set SDS primary	Defines the server as a shared disk primary server	
ha_alias	The high-availability alias of the database server	When used with **set SDS** or **make primary**, this is the name of the server whose role is changing.
force	Used to force a change	If the **force** option is specified, the operation is performed without requiring that the secondary server be connected to the current primary server. If the **force** option is not specified, the operation must be coordinated with the current primary server. The **force** option should be used only when the DBA is certain that the current primary server is not active; otherwise, the shared disk subsystem can become corrupted.
clear SDS primary	Disables the shared disk environment. The server name specified no longer acts as an SD primary server	
make primary	Creates a primary server	The **make primary** command can be issued on any type of secondary server, including HDR secondary, RS secondary, and SD secondary servers. If **make primary** is run on: • HDR Secondary: The current primary server is shut down and the secondary is made the primary. • RS secondary: The server is changed to a standard server. • SD secondary: The server is made the new primary server.

You can also set data replication characteristics can with SQL administration API *command* equivalents. For more information see SQL Administration API Overview and the *IBM® Informix® Administrator's Guide*.

For other **onmode -d** information, see onmode -d: Set data-replication types and onmode -d command: Replicate an index with data-replication.

Related reference:
ha make primary argument: Change the mode of a secondary server (SQL administration API)
ha rss argument: Create an RS secondary server (SQL administration API)
ha rss add argument: Add an RS secondary server to a primary server (SQL administration API)
ha rss change argument: Change the password of an RS secondary server (SQL administration API)
ha rss delete argument: Delete an RS secondary server (SQL administration API)
ha sds clear argument: Stop shared-disk replication (SQL administration API)
ha sds set argument: Create a shared-disk primary server (SQL administration API)
ha sds primary argument: Convert an SD secondary server to a primary server (SQL administration API)
onmode -d: Set data-replication types

DBSERVERALIASES configuration parameter
DBSERVERNAME configuration parameter
HA_ALIAS configuration parameter
onmode -d command: Replicate an index with data-replication

onmode -d command: Replicate an index with data-replication

Syntax:

```
>>-onmode-- -d--+-idxauto--+-on--+------------------------+----><
               |            '-off-'                       |
               '-index--database:-+--------+-table#index-'
                                  '-owner.-'
```

Element	Purpose	Key considerations
-d	Specifies how indexes are replicated to a High-Availability Data-Replication (HDR) secondary server when an index on the secondary server becomes corrupt	You can use the **onmode -d idxauto** and **onmode -d index** commands while the server is in online mode.
idxauto	Enables automatic index replication when an index on a secondary server becomes corrupt	Use the **onmode -d idxauto** command to overwrite the value of the DRIDXAUTO configuration parameter within a session.
index	Replicates an index from a primary to a secondary server	If you detect a corrupt index on a secondary server, use the **onmode -d index** command to start replication of the index from the primary to the secondary server.
database	Specifies the database containing the index to replicate	Syntax must conform to the Identifier segment; see the *IBM® Informix® Administrator's Guide*.
index	Specifies the name of the index to replicate	Index must exist on table and in database specified. Syntax must conform to the Identifier segment; see the *IBM Informix Administrator's Guide*.
owner	Specifies the owner of a table	You must specify the current owner of the table. Syntax must conform to the Table Name segment; see the *IBM Informix Administrator's Guide*.
table	Specifies the name of the table on which the index is based	Syntax must conform to the Table Name segment; see the *IBM Informix Administrator's Guide*.

The **onmode -d idxauto** and the **onmode -d index** commands provide methods to replicate an index to a secondary server containing a corrupted index. The base table will be locked during the transfer of an index. The alternative to using these options is to drop and rebuild the corrupt index on the primary server.

In the case of a fragmented index with one corrupt fragment, the **onmode -d idxauto** command only transfers the single affected fragment, whereas the **onmode -d index** command transfers the whole index.

Related reference:
DRIDXAUTO configuration parameter
ha set idxauto argument: Replicate indexes to secondary servers (SQL administration API)
onmode -d: Set data-replication types
onmode -d: Set High Availability server characteristics
DBSERVERALIASES configuration parameter
DBSERVERNAME configuration parameter
HA_ALIAS configuration parameter

onmode -D, -M, -Q, -S: Change decision-support parameters

Syntax:

```
>>-onmode--+- -D--max_priority-+-------------------------------><
           +- -M--kilobytes----+
           +- -Q--queries------+
           '- -S--scans--------'
```

Element	Purpose	Key considerations
-D *max_priority*	Changes the value of MAX_PDQPRIORITY	This value must be an unsigned integer between 0 and 100. Specify *max_priority* as a factor to temper user requests for PDQ resources. For information on parameters used for controlling PDQ, see MAX_PDQPRIORITY configuration parameter and the *IBM® Informix Performance Guide*.

Element	Purpose	Key considerations
-M *kilobytes*	Changes the value of DS_TOTAL_MEMORY	This value has a platform-dependent upper limit. If you enter a very large value and that value is too large for your platform, you will receive a message that gives you the range of values for your platform. Specify *kilobytes* for the maximum amount of memory available for parallel queries. For more information, see DS_TOTAL_MEMORY configuration parameter and the *IBM Informix Performance Guide*.
-Q *queries*	Changes the value of DS_MAX_QUERIES	This value must be an unsigned integer between 1 and 8,388,608. Specify *queries* for the maximum number of concurrently executing parallel queries. For information on parameters used for controlling PDQ, see DS_MAX_QUERIES configuration parameter and the *IBM Informix Performance Guide*.
-S *scans*	Changes the value of DS_MAX_SCANS	This value must be an unsigned integer between 10 and 1,048,576. Specify *scans* for the maximum number of concurrently executing parallel scans. For information on parameters used for controlling PDQ, see DS_MAX_SCANS configuration parameter and the *IBM Informix Performance Guide*.

These options allow you to change configuration parameters while the database server is online. The new values affect only the current instance of the database server; the values are not recorded in the ONCONFIG file. If you shut down and restart the database server, the values of the parameters revert to the values in the ONCONFIG file. For more information about these configuration parameters, see Database configuration parameters.

To check the current values for the MAX_PDQPRIORITY, DS_TOTAL_MEMORY, DS_MAX_SCANS, DS_MAX_QUERIES, and the DS_NONPDQ_QUERY_MEM configuration parameters, use **onstat -g mgm**. See onstat -g mgm command: Print MGM resource information.

This command has an equivalent SQL administration API function.

Related reference:
DS_MAX_QUERIES configuration parameter
DS_MAX_SCANS configuration parameter
MAX_PDQPRIORITY configuration parameter
DS_TOTAL_MEMORY configuration parameter
onmode and D arguments: Set PDQ priority (SQL administration API)
onmode and M arguments: Temporarily change decision-support memory (SQL administration API)
onmode and Q arguments: Set maximum number for decision-support queries (SQL administration API)
onmode and S arguments: Set maximum number of decision-support scans (SQL administration API)

onmode -e: Change usage of the SQL statement cache

Syntax:

```
>>-onmode-- -e--mode-------------------------------------------><
```

Element	Purpose	Key considerations
onmode -e `ENABLE`	Enables the SQL statement cache. For more information, see the material on improving query performance in the *IBM® Informix® Performance Guide*.	User sessions use the cache only when they perform either of the following actions: • Set the environment variable STMT_CACHE to `1` • Execute the SQL statement SET STATEMENT CACHE ON
onmode -e `FLUSH`	Flushes the statements that are not in use from the SQL statement cache	The **onstat -g ssc ref_cnt** field shows 0.
onmode -e `OFF`	Turns off the SQL statement cache	No statements are cached.
onmode -e `ON`	Turns on the SQL statement cache	All statements are cached unless the user turns it off with one of the following actions: • Set the environment variable STMT_CACHE to `0` • Execute the SQL statement SET STATEMENT CACHE OFF

The **onmode -e** changes are in effect for the current database server session only. When you restart the database server, it uses the default STMT_CACHE parameter value in the ONCONFIG file.

This command has an equivalent SQL administration API function.

Related reference:
STMT_CACHE configuration parameter
onmode and e arguments: Change usage of the SQL statement cache (SQL administration API)

onmode -F: Free unused memory segments

Use the **onmode -F** command to free shared-memory segments that are unavailable or no longer needed for a process

Syntax:

```
>>-onmode-- -F-------------------------------------------------><
```

Element	Purpose	Key considerations
-F	Frees unused memory segments	None.

When you execute **onmode -F**, the memory manager examines each memory pool for unused memory. When the memory manager locates blocks of unused memory, it immediately frees the memory. After the memory manager checks each memory pool, it begins checking memory segments and frees any that the database server no longer needs.

It is recommended that you run **onmode -F** from an operating-system scheduling facility regularly and after the database server performs any function that creates additional memory segments, including large index builds, sorts, or backups.

Running **onmode -F** causes a significant degradation of performance for any users that are active when you execute the utility. Although the execution time is brief (1 to 2 seconds), degradation for a single-user database server can reach 100 percent. Systems with multiple CPU virtual processors experience proportionately less degradation.

To confirm that **onmode** freed unused memory, check your message log. If the memory manager frees one or more segments, it displays a message that indicates how many segments and bytes of memory were freed.

This command has an equivalent SQL administration API function.

Related reference:
onmode and F arguments: Free unused memory segments (SQL administration API)

onmode -h: Update sqlhosts caches

Syntax

```
>>-onmode-- -h--+-------+--><
               +-force-+
```

Element	Purpose	Key considerations
-h	Allows you to update sqlhosts caches.	The database server maintains a hierarchy of sqlhosts caches. There is a cache of sqlhosts entries for each session. In addition, there exists a global cache of the sqlhosts entries which can be enabled and disabled by the use of the sqlhosts argument of the NS_CACHE configuration parameter. The global sqlhosts cache entries will be reloaded when this cache is enabled and the last modification time of the sqlhosts file is newer than the cache read time of the entry.
-force	This optional argument allows you to trigger the sqlhosts caches unconditionally.	The sqlhosts caches will be reloaded unconditionally.

This command has an equivalent SQL administration API function. For more information, see onmode and h arguments: Update sqlhosts caches (SQL administration API)

onmode -I: Control diagnostics collection

Use the **onmode -I** option to start and stop diagnostics collection.

When you encounter an error, you can specify the **onmode -I** *iserrno* option to start collecting diagnostics information. You can also specify the session ID to collect information for only specific session.

To stop the diagnostics collection, use the onmode **-I** option without any other parameters.

```
>>-onmode-- -I--+----------------------+-----------------------><
                |          .-,-------. |
                |          V         | |
                '-iserrno----+-----+-+-'
                             '-sid-'
```

Element	Purpose	Key Considerations
iserrno	Message number of the error that you want to collect diagnostic information for.	None.
sid	Session ID of the session that you want to collect diagnostic information for.	None.

The diagnostics collection procedures are described in the *IBM® Informix® Administrator's Guide*.

onmode -k, -m, -s, -u, -j: Change database server mode

Syntax:

```
>>-onmode--+- -k-+-----------------------------------------><
           +- -m-+
           +- -s-+
           +- -u-+
           '- -j-'
```

Element	Purpose	Key Considerations
-k	Takes the database server to offline mode and removes shared memory	To reinitialize shared memory, shut down and restart the database server. Taking the Database Server to Offline Mode with the -k Option.
-m	Takes the database server from quiescent or administration mode to online mode	See Bringing the Database Server Online with the -m Option.
-s	Shuts down the database server gracefully	Users who are using the database server are allowed to finish before the database server comes to quiescent mode, but no new connections are allowed. When all processing is finished, **-s** takes the database server to quiescent mode. The **-s** option leaves shared memory intact. See Shutting Down the Database Server Gracefully with the -s Option.
-u	Shuts down the database server immediately	This option brings the database server to quiescent mode without waiting for users to finish their sessions. Their current transactions are rolled back, and their sessions are terminated. See Shutting Down the Database Server Immediately with the -u Option.
-j	Puts the database server into administration mode	This option brings the database server to administration mode, allowing the **informix** user all functions including the issuance of SQL and DDL commands. The **-j -U** option enables the DBSA to designate specific users (in addition to the **informix** user) to access the database server. See your *IBM® Informix® Administrator's Guide*.

The following sections describe the options that take the database server from one mode to another.

- Taking the Database Server to Offline Mode with the -k Option
  The **onmode -k** option takes the database server to offline mode and removes database server shared memory.
- Bringing the Database Server Online with the -m Option
- Shutting Down the Database Server Gracefully with the -s Option
- Shutting Down the Database Server Immediately with the -u Option
- Changing the Database Server to Administration Mode with the -j Option

Related reference:
onmode and j arguments: Switch the database server to administration mode (SQL administration API)
onmode and m arguments: Switch to multi-user mode (SQL administration API)

Taking the Database Server to Offline Mode with the -k Option

The **onmode -k** option takes the database server to offline mode and removes database server shared memory.

A prompt asks for confirmation. Another prompt asks for confirmation to kill user threads before the database server comes offline. If you want to eliminate these prompts, execute the **-y** option with the **-s** option.

This option does not kill all client sessions. Use the **-u** option to avoid hanging client sessions or virtual server processes.
Important: When you use the **onmode -k** command to shut down the database server, utilities that are waiting for a user response might not terminate. For example, **ontape** might be waiting for another tape, **onstat -i** might be waiting for a user response, or **onspaces** might be waiting for **y** or **n** to continue. If this problem occurs, use **onmode -uk** or **-uky** instead to roll back work before removing shared memory. For more information, see the descriptions of other options on this page.
Related reference:
The onclean utility
The onshutdown script

Bringing the Database Server Online with the -m Option

The **-m** option brings the database server online from quiescent mode.

Shutting Down the Database Server Gracefully with the -s Option

The **-s** option causes a graceful shutdown. Users who are using the database server are allowed to finish before the database server comes to quiescent mode, but no new connections are allowed. When all processing is finished, **-s** takes the database server to quiescent mode. The **-s** option leaves shared memory intact.

A prompt asks for confirmation. If you want to eliminate this prompt, execute the **-y** option with the **-s** option.

Shutting Down the Database Server Immediately with the -u Option

The **-u** option causes immediate shutdown. This option brings the database server to quiescent mode without waiting for users to finish their sessions. Their current transactions are rolled back, and their sessions are terminated.

A prompt asks for confirmation. Another prompt asks for confirmation to kill user threads before the database server comes to quiescent mode. If you want to eliminate these prompts, execute the **-y** option with the **-s** option.

Changing the Database Server to Administration Mode with the -j Option

The **-j** option puts the database server into the administration mode and allows only the DBSA group and the user **informix** to connect to the server. The **-j** option allows a DBSA to have the server in a fully functional mode to perform maintenance.

The **-j -U** option enables the DBSA to grant individual users access to the database server in administration mode. Once connected, these individual users can execute any SQL or DDL command. When the server is changed to administration mode, all sessions for users other than user **informix**, the DBSA group users, and those identified in the **onmode -j -U** command lose their database server connection.

The following example enables three individual users to connect to the database server and have database server access until the database server mode changes to offline, quiescent or online mode:

```
onmode -j -U karin,sarah,andrew
```

Access for individual users can also be removed by executing **onmode -j -U** and removing their name from the new list of names in the command. For example, in the following commands, the first command grants only Karin access, the second command grants Karin and Sarah access, and the third command grants only Sarah access (and removes access from Karin).

```
onmode -j -U karin
onmode -j -U karin,sarah
onmode -j -U sarah
```

To allow user **informix** and the DBSA group user to retain their database server access in administration mode and remove all single users from accessing the database server, use the following command:

```
onmode -j -U ' '
```

For information on designating single users in administration mode using a configuration parameter, see ADMIN_MODE_USERS configuration parameter

Related reference:
ADMIN_MODE_USERS configuration parameter

onmode -l: Switch the logical-log file

Syntax:

```
>>-onmode-- -l-----------------------------------------------><
```

Element	Purpose	Key considerations
-l	Switches the current logical-log file to the next logical-log file	You must use **onmode** to switch to the next logical-log file. For information on switching to the next logical-log file, see the chapter on managing logical-log files in the *IBM® Informix® Administrator's Guide*.

This command has an equivalent SQL administration API function.

Related reference:
onmode and l arguments: Switch to the next logical log (SQL administration API)

onmode -n, -r: Change shared-memory residency

Syntax:

```
>>-onmode--+- -n-+-----------------------------------------><
           '- -r-'
```

Element	Purpose	Key considerations
-n	Ends forced residency of the resident portion of shared memory	This command does not affect the value of RESIDENT, the forced-residency parameter in the ONCONFIG file.
-r	Starts forced residency of the resident portion of shared memory	This command does not affect the value of RESIDENT, the forced-memory parameter in the ONCONFIG file.

Important: Set the RESIDENT parameter to 1 before you use the **onmode -r** or **-n** options.
For information on using the forced-residency parameter to turn residency on or off for the next time that you restart the database server, see the chapter on managing shared memory in the *IBM® Informix® Administrator's Guide*.

This command has an equivalent SQL administration API function.

Related reference:
RESIDENT configuration parameter
onmode and n arguments: Unlock resident memory (SQL administration API)
onmode and r arguments: Force residency of shared memory (SQL administration API)

onmode -O: Override ONDBSPACEDOWN WAIT mode

Syntax:

```
>>-onmode-- -O-------------------------------------------------><
```

Element	Purpose	Key considerations
-O	Overrides the WAIT mode of the ONDBSPACEDOWN configuration parameter	None.

Use the **onmode -O** option only in the following circumstances:

- ONDBSPACEDOWN is set to WAIT.
- A disabling I/O error occurs that causes the database server to block all updating threads.
- You cannot or do not want to correct the problem that caused the disabling I/O error.
- You want the database server to mark the disabled dbspace as down and continue processing.

When you execute this option, the database server marks the dbspace responsible for the disabling I/O error as down, completes a checkpoint, and releases blocked threads. Then **onmode** prompts you with the following message:

```
This will render any dbspaces which have incurred disabling I/O errors unusable
and require them to be restored from an archive.
Do you wish to continue?(y/n)
```

If **onmode** does not find any disabling I/O errors on noncritical dbspaces when you run the -O option, it notifies you with the following message:

```
There have been no disabling I/O errors on any noncritical dbspaces.
```

This command has an equivalent SQL administration API function.

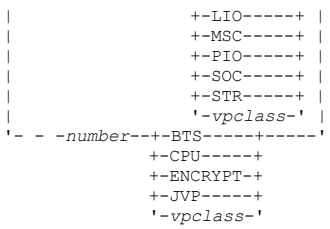
Related reference:
ONDBSPACEDOWN configuration parameter
onmode and O arguments: Mark a disabled dbspace as down (SQL administration API)

onmode -p: Add or drop virtual processors

Use the **onmode -p** command to dynamically add or drop virtual processors for the database server instance. The **onmode -p** command does not update the onconfig file.

Syntax:

```
>>-onmode-- -p--+-+----+--number--+-ADT-----+-+----------------><
               | '- +-'           +-AIO-----+ |
               |                  +-BTS-----+ |
               |                  +-CPU-----+ |
               |                  +-DWAVP---+ |
               |                  +-ENCRYPT-+ |
               |                  +-JVP-----+ |
```



Element	Purpose	Key Considerations
-p *number*	Adds or drops virtual processors. The *number* argument indicates the number of virtual processors to add or drop If this value is a negative integer, processors are dropped. If this value is a positive integer, processors are added.	You can use the **-p** option only when the database server is in online mode, and you can add to only one class of virtual processors at a time. For more details, see Rules for adding and dropping virtual processors. If you are dropping virtual processors, the maximum cannot exceed the actual number of processors of the specified type. If you are adding virtual processors, the maximum number depends on the operating system. For more information, see the chapter on using virtual processors in the *IBM® Informix Administrator's Guide*.
ADT	Runs auditing processes	The database server starts one virtual processor in the audit class when you turn on audit mode by setting the ADTMODE parameter in the ONCONFIG file.
AIO	Performs nonlogging disk I/O to cooked disk spaces	Also performs nonlogging I/O to raw disk spaces if kernel asynchronous I/O (KAIO) is not used.
BTS	Run basic text search index operations and queries.	BTS virtual processors are non-yielding. Specify more BTS virtual processors if you want multiple basic text search queries to be run simultaneously. Use the VPCLASS parameter in the onconfig file to create at least one permanent BTS virtual processor. For more information on basic text search queries, see the *IBM Informix Database Extensions User's Guide*.
CPU	Runs all session threads and some system threads	It is recommended that the number of CPU VPs not be greater than the number of physical processors. If KAIO is used, performs I/O to raw disk spaces, including I/O to physical and logical logs. Runs thread for KAIO where available or a single poll thread. The database server uses the number of CPU VPs to allocate resources for parallel database queries (PDQ). If you drop CPU VPs, your queries will run significantly slower. The **Reinit** field of the **onstat -g mgm** output displays information on the number of queries that are waiting for running queries to complete after an **onmode -p** command. Also see the *IBM Informix Performance Guide*.
DWAVP	Runs the administrative functions and procedures for Informix® Warehouse Accelerator on a database server that is connected to Informix Warehouse Accelerator	A single DWAVP can process most Informix Warehouse Accelerator operations without delay. However, in systems with significant activity, you can define a maximum of two DWAVP virtual processors to avoid the delay of other administrative commands while data marts are loading.
ENCRYPT	Executes column-level encryption and decryption routines	Specify more ENCRYPT virtual processors if you have multiple encrypted columns.
JVP	Executes Java™ user-defined routines in the Java Virtual Machine (JVM)	Specify more JVPs if you are running many Java UDRs.
LIO	Writes to the logical-log files if they are in cooked disk space	Use two LIO virtual processors only if the logical logs are in mirrored dbspaces. The database server allows a maximum of two LIO virtual processors.
MSC	Manages requests for system calls that require a large stack	Used for miscellaneous internal tasks.
PIO	Writes to the physical log if it is in cooked disk space	Use two PIO virtual processors only if the physical log is in a mirrored dbspace. The database server allows a maximum of two PIO virtual processors.
SOC	Uses sockets to perform network communications	You can use the SOC virtual processor only if the database server is configured for network connections through sockets.
STR	Performs stream pipe connections	
vpclass	Names a user-defined virtual processor class	Use the VPCLASS parameter in the onconfig to define the user-defined virtual-processor class. Specify more user-defined virtual processors if you are running many UDRs. On Windows, you can have only one user-defined virtual processor class at a time. Omit the *number* parameter in the **onmode -p** *vpclass* command. For more information on extension classes, see VPCLASS configuration parameter.

- Rules for adding and dropping virtual processors
  You can add or drop virtual processors.
- Monitoring poll threads with the onstat utility

Related reference:
onmode and p arguments: Add or remove virtual processors (SQL administration API)
VPCLASS configuration parameter

Rules for adding and dropping virtual processors

You can add or drop virtual processors.

The following rules apply:

- You cannot drop the final virtual processor. At least one virtual processor must remain.
- You cannot add or drop ADM or OPT.
- **Windows Only:** You can add a supported virtual processor of any class, but you cannot drop virtual processors.

These are the virtual processors that you can add or drop:

Virtual processor name	Add	Drop
ADT	Yes	No
AIO	Yes	No
BTS	Yes	Yes
CPU	Yes	Yes
ENCRYPT	Yes	Yes
JVP	Yes	Yes
LIO	Yes[1]	No
MSC	Yes	No
PIO	Yes[1]	No
SOC	Yes	No
STR	Yes	No
vpclass	Yes	Yes

Table note:

1. You can add one more virtual processor.

Monitoring poll threads with the onstat utility

While the database server is online, you cannot drop a CPU virtual processor that is running a poll thread. To identify poll threads that run on CPU virtual processors, use the following command:

```
onstat -g ath | grep 'cpu.*poll'
```

The following **onstat -g ath** output shows two CPU virtual processors with poll threads. In this situation, you cannot drop to fewer than two CPU virtual processors.

```
tid  tcb      rstcb  prty  status    vp-class  name

8     a362b90  0     2     running   1cpu      tlitcppoll
9     a36e8e0  0     2     cond wait arrived   3cpu
```

The `status` field contains information, such as `running`, `cond wait`, `IO Idle`, `IO Idle`, `sleeping secs: number_of_seconds`, or `sleeping forever`. To improve performance, you can remove or reduce the number of threads that are identified as `sleeping forever`.

For more information on the types of virtual processors, see the chapter on virtual processors and threads in the *IBM® Informix® Administrator's Guide*.

This command has an equivalent SQL administration API function.

onmode -P: Start, stop, or restart a listen thread dynamically

Use the **onmode -P** command to start, stop, or restart an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.

Syntax:

```
>>-onmode-- -P--+-start---+--server_name-----------------------><
               +-stop----+
               '-restart-'
```

Element	Purpose	Key Considerations
start	Start a new listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.	The definition of the listen thread must exist in the **sqlhosts** file for the server. If the definition of the listen thread does not exist in the **sqlhosts** file, you must add it before you can start the listen thread dynamically.

Element	Purpose	Key Considerations
stop	Stop an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.	The definition of the listen thread must exist in the **sqlhosts** file for the server.
restart	Stop and start an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.	The definition of the listen thread must exist in the **sqlhosts** file for the server.
server_name	The name of the database server on which you want to start, stop, or restart a listen thread.	

These commands do not update the **sqlhosts** file.

These commands are equivalent to the SQL administration API functions that have start listen, stop listen, or restart listen arguments.

Example

The following command stops and then starts a listen thread for a server named **ids_serv1**:

```
onmode -P restart ids_serv1
```

Related reference:
start listen argument: Start a listen thread dynamically (SQL administration API)
stop listen argument: Stop a listen thread dynamically (SQL administration API)
restart listen argument: Stop and start a listen thread dynamically (SQL administration API)

onmode -R: Regenerate .infos.dbservername File

The database server creates the .infos.*dbservername* file when you initialize shared memory and removes the file when you take the database server offline. This file is in the $INFORMIXDIR/etc or %INFORMIXDIR%\etc directory. The name of this file is derived from the DBSERVERNAME parameter in the ONCONFIG configuration file.

The database server uses information from the .infos.*dbservername* file when it accesses utilities. If the file is accidentally deleted, you must either re-create the file or shut down and restart the database server.

Syntax:

```
>>-onmode-- -R-------------------------------------------------><
```

Element	Purpose	Key Considerations
-R	Re-creates the .infos.*dbservername* file	Before you use the **-R** option, set the INFORMIXSERVER environment variable to match the DBSERVERNAME parameter from the ONCONFIG file. Do not use the **-R** option if the INFORMIXSERVER environment variable is set to one of the DBSERVERALIASES names.

onmode -W: Change settings for the SQL statement cache

Syntax:

```
>>-onmode-- -W--+-STMT_CACHE_HITS--hits-----+------------------><
               '-STMT_CACHE_NOLIMIT--value-'
```

Element	Purpose	Key Considerations
STMT_CACHE_HITS *hits*	Specifies the number of hits (references) to a statement before it is fully inserted in the SQL statement cache Set ***hits*** to 1 or more to exclude ad hoc queries from entering the cache.	You can only increase or reset the value of STMT_CACHE_HITS. The new value displays in the **#hits** field of the **onstat -g ssc** output. If ***hits*** = 0, the database server inserts all qualified statements and its memory structures in the cache. If ***hits*** > 0 and the number of times the SQL statement has been executed is less than STMT_CACHE_HITS, the database server inserts *key-only* entries in the cache. It inserts qualified statements in the cache after the specified number of hits have been made to the statement. **ONCONFIG** Parameter: STMT_CACHE_HITS
STMT_CACHE_NOLIMIT *value*	Controls whether statements are inserted in the SQL statement cache.	If *value* = 0, the database server inserts no statements in the cache. If *value* = 1, the database server always inserts statements in the cache. If none of the queries are shared, turn off STMT_CACHE_NOLIMIT to prevent the database server from allocating a large amount of memory for the cache.**ONCONFIG** Parameter: STMT_CACHE_NOLIMIT

- SQL statement cache examples

Related reference:
STMT_CACHE_HITS configuration parameter
STMT_CACHE_NOLIMIT configuration parameter
onmode and W arguments: Reset statement cache attributes (SQL administration API)

SQL statement cache examples

The following are examples of **onmode -W** commands for changing SQL statement cache (SSC) settings. The changes are in effect for the current database server session only and do not change the ONCONFIG values. When you restart the database server, it uses the default SSC settings, if not specified in the ONCONFIG file, or the ONCONFIG settings. To make the changes permanent, set the appropriate configuration parameter.

```
onmode -W STMT_CACHE_HITS 2  # number of hits before statement is
     # inserted into SSC
onmode -W STMT_CACHE_NOLIMIT 1  # always insert statements into
     # the cache
```

This command has an equivalent SQL administration API function.

onmode -we: Export a file that contains current configuration parameters

Use the **onmode -we** command to create and export a configuration file that is a snapshot of your current configuration parameters.

Syntax:

```
>>-onmode-- -we--path_name-------------------------------------><
```

Element	Description	Key Considerations
path_name	The full or relative path name of the configuration file.	Do not add an extension.

Usage

The **onmode -we** command automatically creates an ASCII file, assigning it the name that you specified in the command. The format of the file is the same as the format of the onconfig.std file.

If you changed any values dynamically during the current session, the exported file contains the changed values instead of the values that are permanently saved in the onconfig file.

After you export the configuration file, you can import it and use it as your configuration file.

If run the **onmode -we** command and specify a file that was previously exported, the command exports the new version of the file, overwriting the previous exported file.

The **onmode -we** command is equivalent to the SQL administration API function that has the onmode and export arguments.

Examples

The following command exports all configuration parameters and their current values to the onconfig3 file in the `/tmp` directory:

```
onmode -we /tmp/onconfig3
```

Related tasks:
Modifying the onconfig file
Related reference:
onmode -wi: Import a configuration parameter file
export config argument: Export configuration parameter values (SQL administration API)

onmode -wf, -wm: Dynamically change certain configuration parameters

Use the **onmode -wf** or **onmode -wm** command to dynamically change specific configuration parameters.

Syntax:

```
>>-onmode--+- -wf--config_param=value-+------------------------><
```

```
'- -wm--config_param=value-'
```

Element	Purpose	Key considerations
-wf	Updates the value of the specified configuration parameter in the onconfig file.	The DBA user must have write permission for the directory that contains the onconfig file.
-wm	Dynamically sets the value of the specified configuration parameter in memory.	The specified *value* is not preserved when the server is restarted.
config_param=*value*	Specifies the configuration parameter and its new value.	See Database configuration parameters.

To see a list of configuration parameters that you can tune dynamically with an **onmode -wm** or **-wf command**, run the **onstat -g cfg tunable** command.

The **onmode -wf** and **onmode -wm** commands have equivalent SQL administration API functions.

- onmode -wm: Change LRU tuning status
  You can use the **onmode -wm** option to change the LRU tuning status without updating the onconfig file.

Related tasks:
Modifying the onconfig file
Related reference:
onstat -g cfg command: Print the current values of configuration parameters
onmode and wf arguments: Permanently update a configuration parameter (SQL administration API)
onmode and wm arguments: Temporarily update a configuration parameter (SQL administration API)
set onconfig memory argument: Temporarily change a configuration parameter (SQL administration API)
set onconfig permanent argument: Permanently change a configuration parameter (SQL administration API)

onmode -wm: Change LRU tuning status

You can use the **onmode -wm** option to change the LRU tuning status without updating the onconfig file.

Syntax:

```
>>-onmode-- -wm--AUTO_LRU_TUNING----+-0-+----------------------><
                                    '-1-'
```

Element	Purpose	Key considerations
-wm	Dynamically sets the value of the specified configuration parameter for the current session.	None.
0	Turns off automatic LRU tuning for the current session.	None.
1	Turns on automatic LRU tuning for the current session.	None.

This command has an equivalent SQL administration API function.

Related reference:
onmode, wm, and AUTO_LRU_TUNING arguments: Change LRU tuning status (SQL administration API)

onmode -wi: Import a configuration parameter file

Use the **onmode -wi** command to import a file that contains new values for multiple configuration parameters. If the parameters are tunable, which means they can be updated individually with an **onmode -wm** command, the database server applies the new values.

Syntax:

```
>>-onmode-- -wi--path_name--------------------------------------><
```

Element	Purpose	Key Considerations
path_name	The full or relative path name of the previously exported configuration file.	

Usage

Importing a configuration file with **onmode -wi** is often faster and more convenient than running individual **onmode -wm** commands on multiple tunable configuration parameters.

The import operation ignores the configuration parameters in the file that are not tunable. The operation also ignores new parameter values that match the values that are currently used by the instance.

After you import the file, you can modify the values of the imported configuration parameters.

An import operation changes only the values of configuration parameters that are in memory. The operation does not affect the values in the $INFORMIXDIR/etc/$ONCONFIG file.

The **onmode -wi** command is equivalent to the SQL administration API functions that have onmode and wi arguments or the import argument.

Example

The following command imports the configuration parameters that are in a file named onconfig3 in the /tmp directory:

```
onmode -wi /tmp/onconfig3
```

Related tasks:
Modifying the onconfig file
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
onmode -we: Export a file that contains current configuration parameters
import config argument: Import configuration parameter values (SQL administration API)

onmode -Y: Dynamically change SET EXPLAIN

Syntax:

```
>>-onmode-- -Y--session_id--+-0--------------------+----------><
                           '-+-2-+--+-----------+-'
                             '-1-'  '-file_name-'
```

Element	Purpose	Key considerations
file_name	The explain output file name.	If the file's absolute path is not included, the example output file is created in the default example output file location. If the file exists, explain output is appended to it. If a file exists from the SET EXPLAIN statement, that file is not used until dynamic explain is turned off.
session_id	Identifies the specific session.	None.
-Y	Dynamically change the value of the SET EXPLAIN statement.	None.

You can use the SET EXPLAIN statement to display the query plan of the optimizer, an estimate of the number of rows returned, and the relative cost of the query. When you use the **onmode -Y** command to turn on SET EXPLAIN, the output is displayed in the explain output file.

The **onmode -Y** command dynamically changes the value of the SET EXPLAIN statement for an individual session. The following invocations are valid with this command:

Invocation	Explanation
`onmode -Y session_id 2`	Turns SET EXPLAIN on for *session_id*
`onmode -Y session_id 1`	Turns SET EXPLAIN on for *session_id* and displays the query statistics section in the explain output file
`onmode -Y session_id 1 /tmp/myexplain.out`	Turns SET EXPLAIN on for *session_id* and writes explain output to /tmp/myexplain.out.
`onmode -Y session_id 0`	Turns SET EXPLAIN off for *session_id*

This command has an equivalent SQL administration API function.

Related reference:
EXPLAIN_STAT configuration parameter
onmode and Y arguments: Change query plan measurements for a session (SQL administration API)
Related information:
SET EXPLAIN statement
Using the FILE TO option
Default name and location of the explain output file on UNIX
Default name and location of the output file on Windows
Report that shows the query plan chosen by the optimizer
The explain output file
Query statistics section provides performance debugging information

onmode -z: Kill a database server session

Syntax:

```
>>-onmode-- -z--sid-------------------------------------------><
```

Element	Purpose	Key considerations
-z *sid*	Kills the session that you specify in *sid*	This value must be an unsigned integer greater than 0 and must be the session identification number of a currently running session.

To use the **-z** option, first obtain the session identification (*sessid*) with **onstat -u**, then execute **onmode -z**, substituting the session identification number for *sid*.

When you use **onmode -z**, the database server attempts to kill the specified session. If the database server is successful, it frees any resources that the session holds. If the database server cannot free the resources, it does not kill the session.

If the session does not exit the section or release the latch, the database server administrator can take the database server offline, as described in Taking the Database Server to Offline Mode with the -k Option, to close all sessions.

This command has an equivalent SQL administration API function.

Related reference:
onmode and z arguments: Terminate a user session (SQL administration API)

onmode -Z: Kill a distributed transaction

Syntax:

```
>>-onmode-- -Z--address-----------------------------------------><
```

Element	Purpose	Key considerations
-Z *address*	Kills a distributed transaction associated with the shared-memory address *address*	This argument must be the address of an ongoing distributed transaction that has exceeded the amount of time that TXTIMEOUT specifies. The address must conform to the operating-system-specific rules for addressing shared-memory. (The address is available from **onstat -x** output.) This option is not valid until the amount of time that the ONCONFIG parameter TXTIMEOUT specifies has been exceeded. The **-Z** option should rarely be used and only by an administrator of a database server involved in distributed transactions. For information on initiating independent actions in a two-phase commit protocol, see the chapter on multiphase commit protocols in the *IBM® Informix® Administrator's Guide*.

Distributed transactions provide the ability to query data on different database servers.

Attention: If applications are performing distributed transactions, killing one of the distributed transactions can leave your client/server database system in an inconsistent state. Try to avoid this situation.
This command has an equivalent SQL administration API function.

Related reference:
onmode and Z arguments: Terminate a distributed transaction (SQL administration API)

The onparams Utility

Use the **onparams** utility to add or drop a logical-log file, change physical-log parameters, and add a new buffer pool.

In This Chapter

This chapter shows you how to use the following **onparams** options:

- onparams -a -d dbspace: Add a logical-log file
- onparams -d -l lognum: Drop a logical-log file
- onparams -p: Change physical-log parameters
- onparams -b: Add a buffer pool

Any **onparams** command fails if a storage-space backup is in progress. If you do not use any options, **onparams** returns a usage statement.

You cannot use the **onparams** utility on High-Availability Data Replication (HDR) secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

You can also use SQL administration API commands that are equivalent to **onparams** commands to add or drop a logical-log file, change physical-log parameters, and add a new buffer pool.

On UNIX, you must be logged in as user **root** or user **informix** to execute **onparams**. Only user **informix** is allowed to execute the SQL administration API *command* strings.

On Windows, you must be a member of the **Informix-Admin** group to execute **onparams**.

- onparams syntax
  Use the **onparams** utility to modify the configuration of logical logs or physical logs.
- onparams -a -d dbspace: Add a logical-log file
- onparams -d -l lognum: Drop a logical-log file
- onparams -p: Change physical-log parameters
- onparams -b: Add a buffer pool
  Use the **onparams -b** command to create a buffer pool that corresponds to the page size of the dbspace.
- Examples of onparams Commands

Related reference:
LOGFILES configuration parameter
RTO_SERVER_RESTART configuration parameter

onparams syntax

Use the **onparams** utility to modify the configuration of logical logs or physical logs.

```
>>-onparams--+-------------------------+--+-----------------+-->
             |                    (1) |  +- -a -d--dbspace-+
             '-|  -FILE option   |-----'  +- -d -l--lognum--+
                                         +- -p-------------+
                                         +- -b-------------+
                                         +- -V-------------+
                                         '- -version-------'
```

Notes:

1. See The -FILE option.

Element	Purpose	Key Considerations
-V	Displays the software version number and the serial number	See Obtaining utility version information.
-version	Displays the build version, host, OS, number and date, as well as the GLS version	See Obtaining utility version information.

onparams -a -d *dbspace*: Add a logical-log file

Syntax:

```
>>-onparams-- -a-- -d--dbspace--+-----------+--+-----+---------><
                               '- -s--size-'  '- -i-'
```

Element	Purpose	Key considerations
-a -d *dbspace*	Adds a logical-log file to the end of the log-file list to the specified *dbspace*	You can add a log file to a dbspace only if the database server has adequate contiguous space. The newly added log files have a status of **A** and are immediately available for use. You can add a log file during a backup. You can have a maximum of 32,767 logical-log files. Use **onstat -l** to view the status of your logical-log files. It is recommended that you take a level-0 backup of the root dbspace and the dbspace that contains the log file as soon as possible. You cannot add a log file to a blobspace or sbspace. Syntax must conform to the Identifier segment; see *IBM® Informix® Guide to SQL: Syntax*.
-i	Inserts the logical-log file after the current log file	Use this option when the Log File Required alarm prompts you to add a logical-log file.
-s *size*	Specifies a size in kilobytes for the new logical-log file	This value must be an unsigned integer greater than or equal to 200 kilobytes If you do not specify a size with the **-s** option, the size of the log file is taken from the value of the LOGSIZE parameter in the ONCONFIG file when database server disk space was initialized. For information on changing LOGSIZE, see the chapter on managing logical-log files in the *IBM Informix Administrator's Guide*.

This command has an equivalent SQL administration API function.

Related reference:
DYNAMIC_LOGS configuration parameter
add log argument: Add a new logical log (SQL administration API)

onparams -d -l *lognum*: Drop a logical-log file

Syntax:

```
>>-onparams-- -d-- -l--lognum--+-----+------------------------><
                               '- -y-'
```

Element	Purpose	Key considerations
-d -l *lognum*	Allows you to drop a logical-log file specified by the log file number	**Restrictions**: The *lognum* value must be an unsigned integer greater than or equal to 0. You can obtain the ***lognum*** from the **number** field of **onstat -l**. The sequence of ***lognum*** might be out of order.
-y	Causes the database server to automatically respond yes to all prompts	None.

Usage

You can only drop one log files at a time.

The database server requires a minimum of three logical-log files at all times. You cannot drop a log if your logical log is composed of only three log files.

Important: Before you can drop any of the first three logical-log files, you must add new logical-log files and run a backup of the logical-log files. The backup must be run using either the **ontape -a** command or the **ontape -c** command. After you add the new logical-log files and run a backup, you can then use **onparams -d -l***lognum* to delete the first three logical-log files.
The status of the log file determines if the log file can be dropped, and the actions taken by the database server when the log file is dropped:

- If you drop a log file that has never been written to, status is newly Added (**A**), the database server deletes the log file and frees the space immediately.
- If you drop a used log file that has a status of User (**U**) or Free (**F**), the database server marks the log file as Deleted (**D**). After you take a level-0 backup of the dbspaces that contain the log files and the root dbspace, the database server deletes the log file and frees the space.
- You cannot drop a log file that is currently in use (**C**) or contains the last checkpoint record (**L**).

This command has an equivalent SQL administration API function.

When you move logical-log files to another dbspace, use the **onparams** commands to add and drop logical-log files. See moving a logical-log file, in the section on managing logical-log files in the *IBM® Informix® Administrator's Guide*.

Related reference:
drop log argument: Drop a logical log (SQL administration API)

onparams -p: Change physical-log parameters

Syntax:

```
>>-onparams-- -p-- -s--size--+---------------+--+-----+---------><
                            '- -d--dbspace-'  '- -y-'
```

Element	Purpose	Key Considerations
-p	Changes the physical log	Whenever you use the **onparams -p** command, you must include the **-s** parameter. Additionally, you can specify the **-d** and **-y** parameters. The database server must be in either administration, online, or quiescent mode to specify the **-p** parameter. The database server does not need to be restarted for the changes take effect.
-s *size*	Changes the size (in kilobytes) of the physical log	This value must be an unsigned integer greater than or equal to 200 kilobytes. Attention: If you move the log to a dbspace without adequate contiguous space or increase the log size beyond the available contiguous space, the operation will fail and the physical log will not change.
-d *dbspace*	Changes the location of the physical log to the specified *dbspace*	The space allocated for the physical log must be contiguous. Syntax must conform to the Identifier segment; see the *IBM® Informix® Guide to SQL: Syntax*.
-y	Causes the database server to automatically respond yes to all prompts	None.

- Backing Up After You Change the Physical-Log Size or Location
- Changing the Size of the Physical Log and Using Non-Default Page Sizes

Related reference:
PHYSFILE configuration parameter
alter plog argument: Change the physical log (SQL administration API)
LOGSIZE configuration parameter

Backing Up After You Change the Physical-Log Size or Location

The database server must be in either the online or quiescent mode when you change the physical log. The database server does not need to be restarted for the changes to take effect.

Create a level-0 backup of the root dbspace immediately after you change the physical-log size or location. This backup is critical for proper recovery of the database server.

Changing the Size of the Physical Log and Using Non-Default Page Sizes

If you use non-default page sizes, you might need to increase the size of your physical log. If you perform many updates to non-default pages you might need a 150 to 200 percent increase of the physical log size. Some experimentation might be needed to tune the physical log. You can adjust the size of the physical log as necessary according to how frequently the filling of the physical log triggers checkpoints.

onparams -b: Add a buffer pool

Use the **onparams -b** command to create a buffer pool that corresponds to the page size of the dbspace.

Syntax

```
>>-onparams-- -b-- -g--size-----------------------------------><
```

Element	Purpose	Key considerations
-b	Creates a buffer pool	You can add a buffer pool while the database server is running.
-g *size*	Specifies the size in KB of the buffer pages to create	The size of the buffer pages must be 2 - 16 KB and a multiple of the default page size.

All other characteristics of the buffer pool that you create are set to the values of the fields in the default line of the BUFFERPOOL configuration parameter.

Each dbspace that you create with a non-default page size must have a corresponding buffer pool with the corresponding page size. If you create a dbspace with a page size that has no buffer pool, the system automatically creates a buffer pool based the fields in the default line of the BUFFERPOOL parameter.

When you add a buffer pool, a new entry for the BUFFERPOOL configuration parameter is added in the onconfig file.

This command has an equivalent SQL administration API function.

Related reference:
add bufferpool argument: Add a buffer pool (SQL administration API)
BUFFERPOOL configuration parameter

Examples of onparams Commands

The following are examples of **onparams** commands:

```
onparams -a -d rootdbs -s 1000  # adds a 1000-KB log file to rootdbs

onparams -a -d rootdbs -i       # inserts the log file after the current log

onparams -d -l 7                # drops log 7

onparams -p -d dbspace1 -s 3000 # resizes and moves physical-log to dbspace1

onparams -b -g 6 -n 3000 -r 2 -x 2.0 -m 1.0 # adds 3000 buffers of size
6K bytes each with 2 LRUS with maximum dirty of 2% and minimum dirty of 1%
```


The onpassword utility

Use the **onpassword** utility to encrypt and decrypt a password file. Connection Manager and Enterprise Replication utilities require a password file to connect to database servers over an untrusted network.

Syntax

```
>>-onpassword-- -k --encryption_key----------------------------->

>--+- -e --text_file--------+---------------------------------><
   '- -d --output_file_name-'
```

Element	Purpose	Key Considerations
-k	Specifies the password key.	
-e	Encrypts an ASCII text file	The password information is encrypted to $INFORMIXDIR/etc/passwd_file
-d	Decrypts the specified encrypted password file.	The passwd_file is decrypted to *$INFORMIXDIR/etc/output_file_name*.
output_file_name	The name of the file that is output by the decryption process.	An encrypted password file that is created on one type of platform is not supported on a different type of platform. You must run the **onpassword** utility on each type of platform, and use the same text file and encryption key.
encryption_key	The encryption key used to encrypt and decrypt password information.	The encryption key can be any sequence of numbers or letters up to 24 bytes in length. To use an encryption key that includes spaces, enclose the encryption key in quotation marks. For example: `"my secret encryption key"`
text_file	The ASCII text file that contains user password information.	The **onpassword** utility uses the following default location: • UNIX: $INFORMIXDIR/tmp • Windows: %INFORMIXDIR%\etc

Usage

Only users logged in as user **informix** have permission to run the **onpassword** utility.

Example 1: Encrypting a password file

To encrypt tmp/my_passwords.txt with **my_secret_encryption_key**, run the following command:

```
onpassword -k my_secret_encryption_key -e my_passwords.txt
```

The password information is encrypted into $INFORMIXDIR/etc/passwd_file.

Example 2: Decrypting an encrypted password file

To decrypt $INFORMIXDIR/etc/passwd_file with **my_secret_encryption_key**, run the following command:

```
onpassword -k my_secret_encryption_key -d my_passwords.txt
```

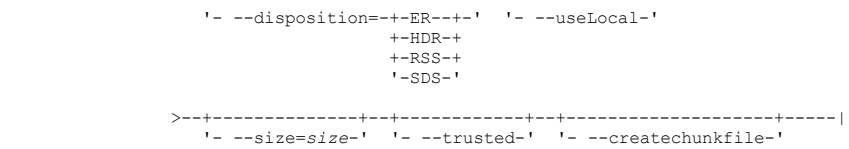
The password information is decrypted to $INFORMIXDIR/etc/my_passwords.txt.
Related information:
Creating a password file for connecting to database servers on untrusted networks
Modifying encrypted password information

The ifxclone utility

You use the **ifxclone** utility to create a server clone from a snapshot of an existing database server.

Syntax

```
>>-ifxclone--+-| Mandatory parameters |--+-------------------------+-+-><
             |                           '-| Optional parameters |-' |
             '- help ------------------------------------------------'

Mandatory parameters

|-- --source=source_name --sourceIP=source_IP --sourcePort=source_port-->

>-- --target=target_name --targetIP=target_IP --targetPort=target_port--|

Optional parameters

   .---------------------------------.
   V                                 |
|----+-------------------------------+-+--+-------------+-------->
     '- --configparmparameter=value-'    '- --autoconf-'

>--+-------------------------+--+-------------+----------------->
```



Element	Purpose	Key Considerations
disposition	Specifies the final disposition of the new server instance.	If the **--disposition** (**-d**) parameter is not specified, a standard server is created.
ER	Specifies that the new server instance is created as a replication server.	
HDR	Specifies that the new server instance is created as an HDR secondary server.	
parameter=value	Specifies an optional configuration parameter and value to set on the target server.	Certain configuration parameters on the source server must match those on the target server. See Prerequisites for cloning an RS secondary server.
RSS	Specifies that the new server instance is created as an remote stand-alone secondary server.	
SDS	Specifies that the new server instance is created as a shared-disk secondary server.	The **ifxclone** utility sets the target server's SDS_PAGING, and SDS_TEMPDBS configuration parameters, but full configuration is outside the scope of the **ifxclone** utility. If **--disposition=SDS** is specified in the command, but **--useLocal** is not, you must set the SD secondary server's ROOTPATH configuration parameter to the same value as the ROOTPATH configuration parameter on the primary server.
size	Specifies the size of the target server. Valid values are `tiny`, `small`, `medium`, and `large`.	If the size parameter is not specified, the size parameters from the source instance are used.
source_name	Specifies the name of the source instance.	The source server must be a primary server and cannot be a secondary server.
source_IP	Specifies the source server instance TCP/IP address.	
source_port	Specifies the TCP/IP port address of the source server instance, or the name of a service associated with the port.	
target_name	Specifies the name of the target server instance.	
target_IP	Specifies the target server instance TCP/IP address.	
target_port	Specifies the TCP/IP port address of the target server instance, or the name of a service associated with the port.	

The following table describes the options of the **ifxclone** utility.

Long Form	Short Form	Meaning
--autoconf	**-a**	Autoconfigures connectivity information between a newly cloned server and the other servers of a high-availability cluster or Enterprise Replication domain. If this option is used to create a replication server, the **--autoconf** option can autoconfigure replication. The **--autoconf** option has the following requirements: • The CDR_AUTO_DISCOVER configuration parameter must be set to `1` on the target server, the source server, and all other cluster or replication servers. • REMOTE_SERVER_CFG must be set on all cluster or replication servers. • The target server's host information must be in the source server's trusted-host file. • If used with the **--disposition=ER** option, and the primary server is part of an Enterprise Replication, all other replication servers in the domain must be active.
--configParm	**-c**	Specifies the name and value of a configuration parameter to set on the target server.
--createchunkfile	**-k**	Automatically creates the same cooked chunk files on the target server as exist on the source server. This option does not create raw chunks. However, if you have raw chunks on the source server and you do not create matching raw chunks on the target server before using this option, the **ifxclone** utility creates cooked chunks on the target server that match the raw chunks on the source server.
--disposition	**-d**	Specifies the disposition of the new server instance.
--help	**-h**	Displays usage information.
--size	**-s**	Specifies the size of the target instance.
--source	**-S**	Specifies the name of the source server instance.
--sourceIP	**-I**	Specifies the TCP/IP address of the source server instance.
--sourcePort	**-P**	Specifies the TCP/IP port address of the source server instance, or the name of a service associated with the port.

Long Form	Short Form	Meaning
--target	**-t**	Specifies the name of the target server instance.
--targetIP	**-i**	Specifies the TCP/IP address of the target server instance.
--targetPort	**-p**	Specifies the TCP/IP port address of the target server instance, or the name of a service associated with the port.
--trusted	**-T**	Specifies that the server is trusted and that it is not necessary to obtain a userid and password to access the server.
--useLocal	**-L**	Specifies that the configuration information contained in the source server onconfig file should be merged with the target server onconfig file. Certain configuration parameters on the source server must match those on the target server. See Prerequisites for all servers.

Usage

Use the **ifxclone** utility to clone a server with minimum setup or configuration, or to quickly add a new node to an existing ER replication domain. When the **ifxclone** utility is run, the majority of the setup information is obtained from the server node that is being cloned. Successfully cloning a server might still require some post-configuration steps to achieve a better running system.

The source server is the server that contains the data you wish to clone. The target server is the server that is to be loaded with data from the source server. You must run the **ifxclone** utility from the target server.

To run the **ifxclone** utility on a UNIX computer, you must run the command on the target server as user **root**, user **informix**, or as a member of the **informix** group. You must also be a DBSA on the source server.

To run the **ifxclone** utility on a Windows computer, you must run the command on the target server as a member of the local administrators group. You must also be a DBSA on the source server and you must belong to the **Informix-Admin** group on the source server.

The **ifxclone** utility uses the onconfig and sqlhosts configuration files from the source server to configure the target server. The **ifxclone** utility also configures some additional configuration settings, but only those required to configure the clone server. The **--autoconf** option provides the additional ability to configure sqlhosts file records, and then propagate sqlhosts and trusted-host file information to the servers of a high-availability cluster or Enterprise Replication domain. The **--createchunkfile** options creates the same cooked chunks and cooked mirror chunks on the target system that are on the source server. The **ifxclone** utility is not meant to configure all of the possible configuration options, but rather to provide enough configuration to clone the source server.

The number of CPU VPs and buffers on the target server can be configured using the size parameter. Table 1 lists the number of CPU VPs and buffer pools created on the target server for each size option. Additional refinement of the generated configuration should be performed after the target system is created. If the size configuration is omitted, the parameter configured on the source server is used.

Table 1. List of size parameter values

Size	Number of CPU VPs	Number of buffers
tiny	1	50,000
small	2	100,000
medium	4	250,000
large	8	500,000

You can use the **-c** option to specify a configuration parameter and its value on the target server. You can also use an existing configuration file. If the target server contains a configuration file that is different than the source server configuration file, the **ifxclone** utility does not overwrite the file but modifies those parameters that must match the source server during the cloning process.

The **--useLocal** parameter is required if the target server is located on the same host machine as the source server.

If the **--useLocal** parameter is specified, the **ifxclone** utility merges the source server onconfig file with the target server onconfig file. The configuration parameters listed in Prerequisites for all servers are overwritten by the **ifxclone** utility and the rest of the parameters are not affected.

If the **--useLocal** parameter is not specified as an input parameter, the **ifxclone** utility uses the source server's onconfig file as the target's onconfig file and uses the server name from the input parameters of the **ifxclone** utility.

If the **--useLocal** parameter is not specified, the **ifxclone** utility updates the sqlhosts file on the host server with the target server entry and copies both entries to the target's sqlhosts file.

The order of precedence of options for the **ifxclone** parameters is as follows:

- The **--configParm** parameter takes precedence over the configuration file on the source server.
- The **--size** parameter takes precedence over merged configuration parameters or the settings in the local configuration file.
- The **--configParm** parameter takes precedence over the **--size** parameter.
- Parameters that must be the same on each server take precedence over all other options.

Prerequisites for all servers

Perform the following prerequisites before cloning a server:

- Hardware and software requirements for the servers are generally the same as those for HDR secondary servers (refer to the machine notes for specific supported platforms).
- Both the source and target servers must be part of a trusted network environment. See Network security files for information about configuring a trusted environment.
- If the disposition of the target server is specified as ER or RSS then you must provide users with connection permission to the **sysadmin** database on the source server. By default, connection permission to the sysadmin database is limited to user **informix**.
- Only one server clone process can occur at a time. Do not start cloning a second server until the first clone process has completed running.

- The source server must have the ENABLE_SNAPSHOT_COPY configuration parameter set to 1 in the onconfig file.
- The target server must not have any old ROOTPATH pages. If the target server has old ROOTPATH pages, create a zero-length ROOTPATH file or set the FULL_DISK_INIT configuration parameter to 1 in the target server's onconfig file.
- The target server must not have any existing storage space encryption keystore or stash files. If you receive an error message that the command failed because of existing keystore and stash files, follow the instructions in the message and rerun the **ifxclone** command.

Archive operations, such as **ontape** and **ON-Bar** commands, are not allowed while cloning a server. Perform your data archive activities before starting to clone a server.

The following environment variables must be set on the target server before cloning a server:

- INFORMIXDIR
- INFORMIXSERVER
- INFORMIXSQLHOSTS
- ONCONFIG

The following configuration parameter values must be identical on both the source and target servers:

- DISK_ENCRYPTION (if the target server is an SD secondary server)
- DRAUTO
- DRINTERVAL
- DRTIMEOUT
- LOGBUFF
- LOGFILES
- LOGSIZE
- LTAPEBLK
- LTAPESIZE
- ROOTNAME
- ROOTSIZE
- PHYSBUFF
- PHYSFILE
- STACKSIZE
- TAPEBLK
- TAPESIZE

If the MIRROR configuration parameter is enabled on the target server, the following configuration parameters also must match between the source and target servers:

- MIRRORPATH
- MIRROROFFSET

The following table shows the valid combinations of the MIRROR configuration parameter on the source and target servers.

Table 2. Valid settings of the MIRROR configuration parameter on source and target servers

MIRROR configuration parameter set on the source server	MIRROR configuration parameter set on the target server	Permitted or not permitted
No	No	Permitted
Yes	Yes	Permitted
Yes	No	Permitted
No	Yes	Not permitted. If this setting is configured, the server issues a warning and disables the MIRROR parameter in the target server onconfig file.

Prerequisites for cloning an RS secondary server

1. Set the following environment variables on the target server:
   - INFORMIXDIR
   - INFORMIXSERVER
   - ONCONFIG
   - INFORMIXSQLHOSTS
2. On the target server, create all of the chunks and mirror chunks that exist on the source server. If the target server is using mirroring, the mirror chunk paths must match those of the source server and the chunks must exist. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server. Follow these steps to create the chunks and (if necessary) mirror chunks for the target server:
   a. On the source server, run the **onstat -d** command to display a list of chunks and mirror chunks:

      ```
      onstat -d
      ```

   b. On the target server, log in as user informix and use the **touch**, **chown**, and **chmod** commands to create the set of chunks and mirror chunks reported by the **onstat -d** command. For example, to create a chunk named /usr/informix/chunks/rootdbs.chunk, follow these steps:

      ```
      $ su informix
      Password:
      $ touch /usr/informix/chunks/rootdbs.chunk
      $ chown informix:informix /usr/informix/chunks/rootdbs.chunk
      $ chmod 660 /usr/informix/chunks/rootdbs.chunk
      ```

   c. Repeat all of the commands in the previous step for each chunk reported by the **onstat -d** command.
3. Run the **ifxclone** utility with the appropriate parameters on the target system.
4. Optionally, create onconfig and sqlhosts files on the target server.

Example 1, Cloning an RS secondary server using the source server configuration

This example shows how to clone a server by using the onconfig and sqlhosts configuration files from the source server.

In this example, omitting the **-L** option causes the **ifxclone** utility to retrieve the necessary configuration information from the source server. The configuration files are used as a template to create the target server configuration. Having the **ifxclone** utility create the configuration files for you saves time and reduces the chance of introducing errors into the configuration files.

The **-k** option creates the necessary cooked chunks on the target server.

For this example, assume that the source server (Amsterdam) has an sqlhosts file configured as follows:

```
#Server   Protocol HostName         Service Group
Amsterdam onsoctcp 192.168.0.1 123     -
```

You also need the name, IP address, and port number of the target server. The following values are used for this example:

- Source server name: Amsterdam
- Source IP address: 192.168.0.1
- Source port: 123
- Target server name: Berlin
- Target IP address: 192.168.0.2
- Target port: 456

1. On the target server, create all of the chunks that exist on the source server. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server. Log-in as user **informix** and use the commands **touch**, **chown**, and **chmod** to create the chunks.
2. On the target server, run the **ifxclone** utility:

   ```
   ifxclone -T -S Amsterdam -I 192.168.0.1 -P 123 -t Berlin
            -i 192.168.0.2 -p 456 -d RSS -k
   ```

   The **ifxclone** utility modifies the sqlhosts file on the source server and creates a copy of the file on the new target server. The sqlhosts file on the target server is the same as the source server:

   ```
   #Server   Protocol HostName         Service Group
   Amsterdam onsoctcp 192.168.0.1 123     -
   Berlin    onsoctcp 192.168.0.2 456
   ```

Example 2, Cloning an RS secondary server by merging the source server configuration

Use the **--useLocal** option to create a clone of a server on a remote host computer: The **--useLocal** option is used to merge the source onconfig file configuration information with the target onconfig file. This option also copies the source sqlhosts file to the target server. The following values are used for this example:

- Source server name: Amsterdam
- Source IP address: 192.168.0.1
- Source port: 123
- Target server name: Berlin
- Target IP address: 192.168.0.2
- Target port: 456

1. Create the onconfig and sqlhosts files and set the environment variables on the target computer.
2. On the target server, create all of the chunks that exist on the source server. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server.Log-in as user **informix** and use the commands **touch**, **chown**, and **chmod** to create the chunks.
3. On the target server, run the **ifxclone** utility:

   ```
   ifxclone -T -L -S Amsterdam -I 192.168.0.1 -P 123 -t Berlin
            -i 192.168.0.2 -p 456 -d RSS -k
   ```

Example 3, Adding an RS secondary server to a cluster

This example shows how to add an RS secondary server to the existing Informix® high-availability cluster. The following values are used for this example:

- Source server name: Amsterdam
- Source IP address: 192.168.0.1
- Source port: 123
- Target server name: Berlin
- Target IP address: 192.168.0.2
- Target port: 456

1. Create the onconfig and sqlhosts files and set the environment variables on the target computer.
2. On the target server, create all of the chunks that exist on the source server. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server.Log-in as user **informix** and use the commands **touch**, **chown**, and **chmod** to create the chunks.
3. On the target server, run the **ifxclone** utility:

   ```
   ifxclone -T -L -S Amsterdam -I 192.168.0.1 -P 123 -t Berlin
            -i 192.168.0.2 -p 456 -s medium -d RSS -k
   ```

Prerequisites for cloning an ER server

Complete the following prerequisites before attempting to clone an ER server.

1. The source server (that is, the server that is being cloned) must have ER configured and active.
2. For configuration parameters that specify directory names, the directory names must exist on the target server. For example, if the CDR_LOG_STAGING_DIR configuration parameter is set to a directory name on the source server then the directory must also exist on the target server.
3. If ATS or RIS is enabled on the source server then the appropriate ATS or RIS directories must exist on the target server. See Enabling ATS and RIS File Generation and Creating ATS and RIS directories. If the directories do not exist then ATS/RIS spooling will fail.
4. If the source server has the CDR_SERIAL configuration parameter set then you must set the value for CDR_SERIAL to a different value on the server to be cloned. The value of CDR_SERIAL must be different on all replication servers. You can specify a unique value for the CDR_SERIAL configuration parameter by using the **--**

configParm (**-c**) parameter in the **ifxclone** command line.
5. The clock on the new ER clone must be appropriately synchronized. See Time synchronization.
6. The source server (that is, the server being cloned) must not have any stopped or suspended replicates, nor can it have any shadow replicates defined.

Avoid performing ER administrative tasks that change the set of replicates on which the target server participates while the **ifxclone** utility is running.

Example: Creating a clone of an ER server

Suppose you have five ER servers named S1, S2, S3, S4, and S5 currently configured as root servers in an ER domain. You would like to add a new server, S6, on a new computer named machine6, and you want it to have the same data as server S3.

1. Install and configure Informix database software on machine6. You can use the deployment utility to deploy a pre-configured database server instance.
2. Copy the sqlhosts file from server S3 to server S6 and modify it to add entries for the new server. For example, assuming the ER group name for the new server is g_S6 and the ID is 60, the sqlhosts file lines would look like the following.

```
g_S6    group     -           -         i=60
S6      onsoctcp  machine6    service6  g=g_S6
```

3. Add the two lines from the previous step in the sqlhosts files on all of the other five servers (S1 through S5).
4. Copy the onconfig file from server S3 to server S6 and change the DBSERVERNAME configuration parameter to S6. Do not modify any storage or chunk parameters except for path information.
5. On server S6 (machine6) provision chunk paths and other storage to the same sizes as server S3. Ensure that S6 has adequate memory and disk space resources. You can use the **--createchunkfile** option (**-k**) to automatically create cooked chunks on the target server.
6. Run the following command as user informix:

```
ifxclone -L -S S3 -I machine3 -P service3 -t S6 -i machine6 -p service6 -d ER -k
```

When prompted, enter the user name **informix** and then enter the password for user **informix**.

7. Monitor the server logs of servers S6 and S3. When the cloning process is complete you can check the status of servers by running the following command on servers S3 and S6:

```
cdr list server
```

You should see the new ER server g_S6 connected to all of the other five servers. In addition, ER node g_S6 will now participate in all replicates in which ER node g_S3 participates.

Related reference:
ENABLE_SNAPSHOT_COPY configuration parameter
DISK_ENCRYPTION configuration parameter
Related information:
CDR_AUTO_DISCOVER configuration parameter

The onspaces utility

Use the **onspaces** utility to manage the storage spaces in your database.

- onspaces syntax
  Run **onspaces** utility commands to manage your storage spaces.
- onspaces -a: Add a chunk to a dbspace or blobspace
- onspaces -a: Add a chunk to an sbspace
- onspaces -c -b: Create a blobspace
- onspaces -c -d: Create a dbspace
  Use the **onspaces -c -d** command to create a dbspace or a temporary dbspace.
- onspaces -c -P: Create a plogspace
  Use the **onspaces -c -P** command to create a plogspace in which to store the physical log.
- onspaces -c -S: Create an sbspace
  Use the **onspaces -c -S** option to create a sbspace or a temporary sbspace.
- onspaces -c -x: Create an extspace
  Use the **onspaces -c -x** option to create an extspace.
- onspaces -ch: Change sbspace default specifications
  Use the **onspaces -ch** option to change the default specifications of a sbspace.
- onspaces -cl: Clean up stray smart large objects in sbspaces
  Use the **onspaces -cl** option to clean up stray smart large objects in sbspaces.
- onspaces -d: Drop a chunk in a dbspace, blobspace, or sbspace
  Use the **onspaces -d** option to drop a chunk in a dbspace, blobspace, or sbspace.
- onspaces -d: Drop a space
  Use the **onspaces -d** option to drop a dbspace, blobspace, plogspace, sbspace, or extspace.
- onspaces -f: Specify DATASKIP parameter
  Use the **onspaces -f** option to specify the value of the DATASKIP configuration parameter on a dbspace level or across all dbspaces.
- onspaces -m: Start mirroring
  Use the **onspaces -m** option to start mirroring for a dbspace, blobspace, or sbspace.
- onspaces -r: Stop mirroring
  Use the **onspaces -r** option to end mirroring for a dbspace, blobspace, or sbspace.
- onspaces -ren: Rename a dbspace, blobspace, sbspace, or extspace
  Use the **onspaces -ren** option to rename a dbspace, blobspace, sbspace, or extspace.
- onspaces -s: Change status of a mirrored chunk
  Use the **onspaces -s** option to change the status of a mirrored chunk in a dbspace, a non-primary chunk within a noncritical dbspace, a blobspace, or an sbspace.
- Avoid overwriting a chunk

Related reference:
TBLTBLFIRST configuration parameter

onspaces syntax

Run **onspaces** utility commands to manage your storage spaces.

```
>>-onspaces--+-------------------------+--+- -a-------+--------><
             |                   (1) |  +- -c -b----+
             '-|  -FILE option   |-----'  +- -c -d----+
                                        +- -c -P----+
                                        +- -c -S----+
                                        +- -c -x----+
                                        +- -ch------+
                                        +- -cl------+
                                        +- -d-------+
                                        +- -f-------+
                                        +- -m-------+
                                        +- -r-------+
                                        +- -ren-----+
                                        +- -s-------+
                                        +- -V-------+
                                        '- -version-'
```

Notes:

1. See The -FILE option.

Element	Purpose	Key Considerations
-V	Shows the software version number and the serial number	See Obtaining utility version information
-version	Shows the build version, host, OS, build number, date, and the GLS version	See Obtaining utility version information

onspaces -a: Add a chunk to a dbspace or blobspace

Syntax:

```
>>-onspaces -a--+-dbspace---+--+- -p--pathname--------------+--->
                '-blobspace-'  |                            (1) |
                               '- -p--\--\--.--\--drive-----'

>-- -o--offset-- -s--size------------------------------------------>

>--+- -m--pathname offset--------------+----------------------><
   |                              (1) |
   '- -m--\--\--.--\--drive offset-----'
```

Notes:

1. Windows only

Use **onspaces -a** to add a chunk to a dbspace or blobspace.

Element	Purpose	Key considerations
-a	Indicates that a chunk is to be added	You can have up to 32766 chunks in an instance. You can put all those chunks in one storage space, or spread them among multiple storage spaces.
drive	Specifies the Windows drive to allocate as unbuffered disk space The format can be either **\\.\<*drive*>**, where ***drive*** is the drive letter assigned to a disk partition, or **\\.*PhysicalDrive*<*number*>**, where ***PhysicalDrive*** is a constant value and ***number*** is the physical drive number.	For more information, see Allocating raw disk space on Windows. Example: \\.\F: For path name syntax, see your operating-system documentation.
-m *pathname offset*	Specifies an optional path name and offset to the chunk that mirrors the new chunk Also see the entries for *pathname* and *offset* in this table.	For more information, see Adding a chunk to a dbspace or blobspace.
-o *offset*	After the **-a** option, *offset* indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the new blobspace or dbspace	Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 terabytes. For more information, see Allocating raw disk space on UNIX.

Element	Purpose	Key considerations
-p *pathname*	Indicates the disk partition or unbuffered device of the initial chunk of the blobspace or dbspace that you are adding The chunk must be an existing unbuffered device or buffered file.	The chunk path name can be up to 256 bytes. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. UNIX example (unbuffered device): /dev/rdsk/c0t3d0s4 UNIX example (buffered device): /ix/ids9.2/db1chunk Windows example: c:\Ifmxdata\ol_icecream\mychunk1.dat For path name syntax, see your operating-system documentation.
-s *size*	Indicates, in kilobytes, the size of the new blobspace or dbspace chunk	Unsigned integer. The size must be equal to or greater than 1000 kilobytes and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 terabytes.
blobspace	Names the blobspace to which you are adding a chunk	See Adding a chunk to a dbspace or blobspace. Syntax must conform to the Identifier.
dbspace	Names the dbspace to which you are adding a chunk	See Adding a chunk to a dbspace or blobspace. Syntax must conform to the Identifier.

This command has an equivalent SQL administration API function.

Related reference:
Avoid overwriting a chunk
add chunk argument: Add a new chunk (SQL administration API)
create chunk argument: Create a chunk (SQL administration API)

onspaces -a: Add a chunk to an sbspace

```
Syntax:

>>-onspaces -a--sbspace-- -p--pathname-- -o--offset------------>

>-- -s--size--+----------------------+--+--------------+-------->
              '- -m--pathname offset-'  '- -Ms--mdsize-'

>--+----------------+--+-----+---------------------------------><
   '- -Mo--mdoffset-'  '- -U-'
```

Use **onspaces -a** to add a chunk to an sbspace.

Element	Purpose	Key considerations
-a	Indicates that a chunk is to be added	You can have up to 32766 chunks in an instance. You can put all those chunks in one storage space, or spread them among multiple storage spaces.
-m *pathname offset*	Specifies an optional path name and offset to the chunk that mirrors the new chunk Also see the entries for ***pathname*** and ***offset*** in this table.	For background information, see adding a chunk to an sbspace, in the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.
-Mo *mdoffset*	Indicates, in kilobytes, the offset into the disk partition or into the device where metadata should be stored	Value can be an integer between 0 and the chunk size. You cannot specify an offset that causes the end of the metadata space to be past the end of the chunk. For background information, see sizing sbspace metadata, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-Ms *mdsize*	Specifies the size, in kilobytes, of the metadata area allocated in the initial chunk. The remainder is user-data space	Value can be an integer between 0 and the chunk size. For background information, see sizing sbspace metadata, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-o *offset*	After the **-a** option, *offset* indicates, in kilobytes, the offset into the disk partition or into the unbuffered device to reach the initial chunk of the new blobspace or dbspace.	Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 2 or 4 terabytes, depending on the platform. For more information, see allocating raw disk space on UNIX, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-p *pathname*	Indicates the disk partition or unbuffered device of the initial chunk of the sbspace that you are creating The chunk must be an existing unbuffered device or buffered file.	The chunk path name can be up to 256 bytes. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. For path name syntax, see your operating-system documentation.
-U	Specifies that the entire chunk should be used to store user data	The **-M** and **-U** options are mutually exclusive. For background information, see adding a chunk to an sbspace, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-s *size*	Indicates, in kilobytes, the size of the new sbspace chunk	Unsigned integer. The size must be equal to or greater than 1000 kilobytes and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 terabytes

Element	Purpose	Key considerations
sbspace	Names the sbspace to which you are adding a chunk	See adding a chunk to an sbspace in the chapter on managing disk space in the *IBM Informix Administrator's Guide*. Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*.

This command has an equivalent SQL administration API function.

Related reference:
Avoid overwriting a chunk
add chunk argument: Add a new chunk (SQL administration API)
create chunk argument: Create a chunk (SQL administration API)

onspaces -c -b: Create a blobspace

```
Syntax:

>>-onspaces -c---- -b--blobspace-- -g--pageunit----------------->

>--+- -p--pathname--------------+-- -o--offset-- -s--size------->
   |                      (1) |
   '- -p--\--\--.--\--drive-----'

>--+-------------------------------------+--+-----+-------------><
   +- -m--pathname offset--------------+  '- -u-'
   |                                   (1) |
   '- -m--\--\--.--\--drive offset-----'
```

Notes:

1. Windows Only

Use **onspaces -c -b** to create a blobspace.

Element	Purpose	Key considerations
-b *blobspace*	Names the blobspace to be created	The blobspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character. For more information, see creating a blobspace, in the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*. The syntax must conform to the Identifier segment. For more information, see the *IBM Informix Guide to SQL: Syntax*.
-c	Creates a dbspace, blobspace, sbspace, or extspace You can create up to 2047 storage spaces of any type.	After you create a storage space, you must back up both this storage space and the root dbspace. If you create a storage space with the same name as a deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see creating a dbspace, blobspace, or extspace, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
drive	Specifies the Windows drive to allocate as unbuffered disk space The format can be either \\.\<*drive*>, where *drive* is the drive letter assigned to a disk partition, or \\.*PhysicalDrive*<*number*>, where *PhysicalDrive* is a constant value and *number* is the physical drive number.	For information on allocating unbuffered disk space, see allocating unbuffered disk space on Windows in the chapter on managing disk space in the *IBM Informix Administrator's Guide*. Examples: `\\.\F:` `\\.\PhysicalDrive2` For path name syntax, see your operating-system documentation.
-g *pageunit*	Specifies the blobspace blobpage size in terms of *page_unit*, the number of the base page size of the instance (either 2K or 4K)	Unsigned integer. Value must be greater than 0. The maximum number of pages that a blobspace can contain is 2147483647. Therefore, the size of the blobspace is limited to the blobpage size x 2147483647. This includes blobpages in all chunks that make up the blobspace. For more information, see blobpage size considerations, in the chapter on I/O Activity in the *IBM Informix Performance Guide*.
-m *pathname offset*	Specifies an optional path name and offset to the chunk that mirrors the initial chunk of the new blobspace or dbspace Also see the entries for **-p** *pathname* and **-o** *offset* in this table.	For more information, see creating a dbspace or a blobspace in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the new blobspace, dbspace, or sbspace	Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 2 or 4 terabytes, depending on the platform. For more information, see allocating raw disk space, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.

Element	Purpose	Key considerations
-p *pathname*	Indicates the disk partition or device of the initial chunk of the blobspace or dbspace that you are creating	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. UNIX example (unbuffered device): /dev/rdsk/c0t3d0s4 UNIX example (buffered device): /ix/ids9.2/db1chunk Windows example:c:\Ifmxdata\ol_icecream\mychunk1.dat For path name syntax, see your operating-system documentation.
-s *size*	Indicates, in kilobytes, the size of the initial chunk of the new blobspace or dbspace	Unsigned integer. The size must be equal to or greater than 1000 kilobytes and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 terabytes, depending on the platform.
-u	Specifies to create an unencrypted space	Use this option to create an unencrypted storage space when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This command has an equivalent SQL administration API function.

Related reference:
Avoid overwriting a chunk
create blobspace argument: Create a blobspace (SQL administration API)
DISK_ENCRYPTION configuration parameter

onspaces -c -d: Create a dbspace

Use the **onspaces -c -d** command to create a dbspace or a temporary dbspace.

Syntax

```
>>-onspaces -c---- -d--dbspace----+- -p--pathname----------+---->
                                  '- -p--\--\--.--\--drive-'

>-- -o--offset-- -s--size----------------------------------------->

>--+-------------------------------------+---------------------->
   +- -ef--extentsize-- -en--extentsize-+
   '- -t--------------------------------'

>--+-------------------------------------+--+---------------+----->
   +- -m--pathname offset--------------+  '- -k--pagesize-'
   |                                   (1) |
   '- -m--\--\--.--\--drive offset-----'

>--+-----+-------------------------------------------------------><
   '- -u-'
```

Notes:

1. Windows Only

Element	Purpose	Key considerations
-c	Creates a dbspace You can create up to 2047 storage spaces of any type.	After you create a storage space, you must back up both this storage space and the root dbspace. If you create a storage space with the same name as a deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see Manage dbspaces.
drive	Specifies the Windows drive to allocate as unbuffered disk space The format can be either \\.*drive*, where *drive* is the drive letter that is assigned to a disk partition, or \\.*PhysicalDrivenumber*, where *PhysicalDrive* is a constant value and *number* is the physical drive number.	For information on allocating unbuffered disk space, see Allocating raw disk space on Windows. Examples: `\\.\F:` `\\.\PhysicalDrive2` For path name syntax, see your operating-system documentation.
-d *dbspace*	Names the dbspace to be created	The dbspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character. For more information, see Manage dbspaces. The syntax must conform to the Identifier segment. For more information, see Identifier.

Element	Purpose	Key considerations
-ef *extentsize*	Indicates, in KB, the size of the first extent for the tblspace **tblspace**	The minimum, and default, size of the first extent for the tblspace **tblspace** of a non-root dbspace is equivalent to 50 dbspace pages, which are specified in KB. For example: 100 KB for a 2 KB page size dbspace, 200 KB for a 4 KB page size dbspace, 400 KB for an 8 KB page size dbspace. The maximum size of a tblspace **tblspace** extent is 1048575 pages minus the space that is needed for any system objects. On a 2 KB page size system, the maximum size is approximately 2 GB. For more information, see Specifying the first and next extent sizes for the tblspace tblspace.
-en *extentsize*	Indicates, in KB, the size of the next extents in the tblspace **tblspace**	The minimum size of the next extents for the tblspace **tblspace** of a non-root dbspace is equivalent to 4 dbspace pages, which are specified in KB. For example: 8 KB for a 2 KB page size dbspace, 16 KB for a 4 KB page size dbspace, 32 KB for an 8 KB page size dbspace. The default size for a next extent is 50 dbspace pages. The maximum size of a tblspace **tblspace** extent is 1048572 pages. On a 2 KB page size system, the maximum size is approximately 2 GB. If there is not enough space for a next extent in the primary chunk, the extent is allocated from another chunk. If the specified space is not available, the closest available space is allocated. For more information, see Specifying the first and next extent sizes for the tblspace tblspace.
-k *pagesize*	Indicates in KB, the non-default page size for the new dbspace. For systems with sufficient storage, performance advantages of a larger page size can include the following: • Reduced depth of B-tree indexes, even for smaller index keys • You can group on the same page long rows that currently span multiple pages of the default page size • Checkpoint time is typically reduced with larger pages • You can define a different page size for temporary tables so that they have a separate buffer pool.	The page size must be between 2 KB and 16 KB and must be a multiple of the default page size. For example, if the default page size is 2 KB, then *pagesize* can be 2, 4, 6, 8, 10, 12, 14, or 16. If the default page size is 4 KB (Windows), then *pagesize* can be 4, 8, 12, or 16. For more information, see Creating a dbspace with a non-default page size.
-m *pathname offset*	Specifies an optional path name and offset to the chunk that mirrors the initial chunk of the new dbspace Also see the entries for **-p** *pathname* and **-o** *offset* in this table.	For more information, see Manage dbspaces.
-o *offset*	Indicates, in KB, the offset into the disk partition or into the device to reach the initial chunk of the new dbspace	Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The offset must be a multiple of the page size. The maximum offset is 2 or 4 TB, depending on the platform. For more information, see Allocating raw disk space on Windows.
-p *pathname*	Indicates the disk partition or device of the initial chunk of the dbspace that you are creating	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. UNIX example (unbuffered device): /dev/rdsk/c0t3d0s4 UNIX example (buffered device): /ix/ids9.2/db1chunk Windows example:c:\Ifmxdata\ol_icecream\mychunk1.dat For path name syntax, see your operating-system documentation.
-s *size*	Indicates, in KB, the size of the initial chunk of the new dbspace	Unsigned integer. The size must be equal to or greater than 1000 KB and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 TB, depending on the platform.
-t	Creates a temporary dbspace for storage of temporary tables	You cannot mirror a temporary dbspace. You cannot specify the first and next extent sizes for the tblspace **tblspace** of a temporary dbspace. For more information, see Temporary dbspaces.
-u	Specifies to create an unencrypted space	Use this option to create an unencrypted storage space when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

The maximum size of a dbspace is equal to the maximum number of chunks multiplied by the maximum size of a chunk. (The maximum number of chunks is 32766 per instance. The maximum size of a chunk is equal to 2147483647 pages multiplied by the page size.)

This command has an equivalent SQL administration API function.

You cannot change the page size of a dbspace after you create it.

You cannot store logical or physical logs in a dbspace that is not the default platform page size.

If a dbspace is created when a buffer pool with that page size does not exist, Informix® creates a buffer pool using the values of the fields of the **default** line of the BUFFERPOOL parameter.

Temporary dbspaces

When you create a temporary dbspace with **onspaces**, the database server uses the newly created temporary dbspace, after you add the name of the new temporary dbspace to your list of temporary dbspaces in the DBSPACETEMP configuration parameter, the DBSPACETEMP environment variable, or both and restart the server.

You cannot specify the first and next extent of a temporary dbspace. The extent size for temporary dbspaces is 100 KB for a 2 KB page system or 200 KB for a 4 KB page system.

You can specify the first and next space of the tblspace **tblspace** in the root dbspace if you do not want the database server to automatically manage the size. To specify the first and next extent sizes of a root tblspace **tblspace**, use the TBLTBLFIRST and TBLTBLNEXT configuration parameters before you create the root dbspace the first time that you start the database server.

Related reference:
DBSPACETEMP configuration parameter
Avoid overwriting a chunk
TBLTBLFIRST configuration parameter
TBLTBLNEXT configuration parameter
create dbspace argument: Create a dbspace (SQL administration API)
create tempdbspace argument: Create a temporary dbspace (SQL administration API)
DISK_ENCRYPTION configuration parameter
Related information:
Specifying the first and next extent sizes for the tblspace tblspace

onspaces -c -P: Create a plogspace

Use the **onspaces -c -P** command to create a plogspace in which to store the physical log.

Syntax

```
>>-onspaces -c---- -P--plogspace--------------------------------->

>--+- -p--pathname----------+-- -o--offset-- -s--size----------->
   '- -p--\--\--.--\--drive-'

>--+-------------------------------------+--+-----+-------------><
   +- -m--pathname --offset------------+  '- -u-'
   |                                   (1) |
   '- -m--\--\--.--\--drive offset-----'
```

Notes:

1. Windows Only

Element	Purpose	Key considerations
-c	Creates a plogspace.	An instance can have only one plogspace. If a plogspace exists, creating a new one moves the physical log to the new space and drops the old plogspace.
-m *pathname offset*	Specifies an optional path name and offset to the chunk that mirrors the chunk of the new plogspace. See **-p** *pathname* and **-o** *offset* in this table.	If you mirror the plogspace, the plogspace chunk cannot be extendable.
-m \\.*drive*	Specifies the Windows drive for the chunk that mirrors the chunk of the new plogspace. The *drive* is the drive letter that is assigned to a disk partition or a constant value and the physical drive number.	Examples: **\\.\F:** **\\.\PhysicalDrive2** For drive name syntax, see your operating-system documentation.
-o *offset*	Indicates, in KB, the offset into the disk partition or into the device to reach the chunk of the new plogspace.	Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The offset must be a multiple of the page size. The maximum offset is 2 or 4 TB, depending on the platform.
-P *plogspace*	Names the plogspace to be created.	The plogspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character. The syntax must conform to the Identifier segment. For more information, see Identifier.

Element	Purpose	Key considerations
-p *pathname*	Indicates the disk partition or device of the chunk of the plogspace that you are creating.	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. UNIX example (unbuffered device): /dev/rdsk/c0t3d0s4 UNIX example (buffered device): /ix/ifmx/db1chunk Windows example: c:\Ifmxdata\ol_icecream\mychunk1.dat
-p \\.*drive*	Specifies the Windows drive to allocate as unbuffered disk space for the plogspace. The *drive* is the drive letter that is assigned to a disk partition or a constant value and the physical drive number.	Examples: `\\.\F:` `\\.\PhysicalDrive2` For drive name syntax, see your operating-system documentation.
-s *size*	Indicates, in KB, the size of the chunk of the new plogspace.	Unsigned integer. The size must be equal to or greater than 1000 KB and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 TB, depending on the platform.
-u	Specifies to create an unencrypted space	Use this option to create an unencrypted storage space when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

The physical log must be stored on a single chunk. By default the chunk for the plogspace is extendable and the database server expands the plogspace as needed to improve performance.

Examples

The following example creates a plogspace that is called **plogdbs** that has a size of 40000 KB and an offset of 0:

```
onspaces -c -P plogdbs -p /dev/chk1 -o 0 -s 40000
```

The following example creates a mirrored plogspace that is called **pdbs1** that has a size of 60000 KB and an offset of 500 KB:

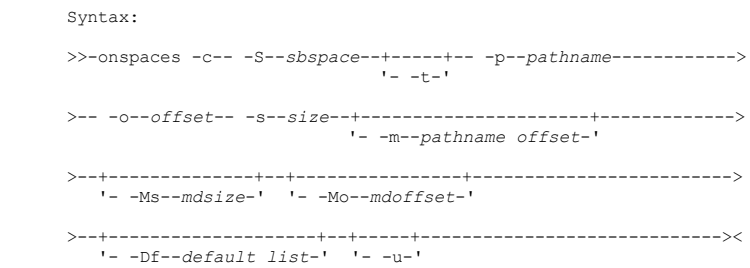
```
onspaces -c -P pdbs1 -p /dev/pchk1 -o 500 -s 60000 -m /dev/mchk1 0
```

Related reference:
create plogspace: Create a plogspace (SQL administration API)
DISK_ENCRYPTION configuration parameter
Related information:
Plogspace
Manage the plogspace

onspaces -c -S: Create an sbspace

Use the **onspaces -c -S** option to create a sbspace or a temporary sbspace.

Syntax:



Element	Purpose	Key Considerations
-S*sbspace*	Names the sbspace to be created	The sbspace name must be unique and must not exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character. Syntax must conform to the Identifier segment; see the *IBM® Informix® Guide to SQL: Syntax*.
-c	Creates an sbspace You can create up to 32767 storage spaces of any type.	None.

Element	Purpose	Key Considerations
-Df *default list*	Lists default specifications for smart large objects stored in the sbspace	Restrictions: Tags are separated by commas. If a tag is not present, system defaults take precedence. The list must be enclosed in double quotation marks (“) on the command line. References: For a list of tags and their parameters, see Table 1.
-m *pathname offset*	Specifies an optional pathname and offset to the chunk that mirrors the initial chunk of the new sbspace Also see the entries for **-p** *pathname* and **-o** *offset* in this table.	For more information, see sbspaces in the chapter on data storage, and creating an sbspace, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-Mo *mdoffset*	Indicates, in kilobytes, the offset into the disk partition or into the device where metadata will be stored.	Restrictions: Value can be an integer between 0 and the chunk size. You cannot specify an offset that causes the end of the metadata space to be past the end of the chunk. References: For more information, see sizing sbspace metadata, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-Ms *mdsize*	Specifies the size, in kilobytes, of the metadata area allocated in the initial chunk The remainder is user-data space.	Restrictions: Value can be an integer between 0 and the chunk size.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the sbspace	Restrictions: Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 4 terabytes for systems with a two-kilobyte page size and 8 terabytes for systems with a four-kilobyte page size. References: For more information, see allocating raw disk space on UNIX, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-p *pathname*	Indicates the disk partition or unbuffered device of the initial chunk of the sbspace	The chunk must be an existing unbuffered device or buffered file. When you specify a pathname, you can use either a full pathname or a relative pathname. However, if you use a relative pathname, it must be relative to the directory that was the current directory when you initialized the database server. References: For pathname syntax, see your operating-system documentation.
-s *size*	Indicates, in kilobytes, the size of the initial chunk of the new sbspace	Restrictions: Unsigned integer. The size must be equal to or greater than 1000 kilobytes and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 terabytes, depending on the platform.
-t	Creates a temporary sbspace for storage of temporary smart large objects. You can specify the size and offset of the metadata area	Restrictions: You cannot mirror a temporary sbspace. You can specify any **-Df** option, except the LOGGING=ON option, which has no effect. **References**: For more information, see Creating a Temporary Sbspace with the -t Option.
-u	Specifies to create an unencrypted space	Use this option to create an unencrypted storage space when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

- Creating a Temporary Sbspace with the -t Option
- Creating an Sbspace with the -Df option
- Changing the -Df Settings
- Using the onspaces -g option

Related reference:
SBSPACENAME configuration parameter
SBSPACETEMP configuration parameter
SYSSBSPACENAME configuration parameter
Avoid overwriting a chunk
create sbspace argument: Create an sbspace (SQL administration API)
create tempsbspace argument: Create a temporary sbspace (SQL administration API)
create sbspace with log argument: Create an sbspace with transaction logging (SQL administration API)
DISK_ENCRYPTION configuration parameter

Creating a Temporary Sbspace with the -t Option

This example creates a temporary sbspace of 1000 kilobytes:

```
onspaces -c -S tempsbsp -t -p ./tempsbsp -o 0 -s 1000
```

You can optionally specify the name of the temporary sbspace in the SBSPACETEMP configuration parameter. Restart the database server so that it can use the temporary sbspace.

Creating an Sbspace with the -Df option

When you create an sbspace with the optional **-Df** option, you can specify several default specifications that affect the behavior of the smart large objects stored in the sbspace. The default specifications must be expressed as a list separated by commas. The list need not contain all of the tags. The list of tags must be enclosed in double quotation marks (“). The table in Table 1 describes the tags and their default values.

The four levels of inheritance for sbspace characteristics are system, sbspace, column, and smart large objects. For more information, see smart large objects in the chapter on where data is stored in the *IBM® Informix Administrator's Guide*.

Table 1. -Df Default Specifications

Tag	Values	Default	Description
ACCESSTIME	ON or OFF	OFF	When set to ON, the database server tracks the time of access to all smart large objects stored in the sbspace. For information about altering storage characteristics of smart large objects, see the *IBM Informix DataBlade API Programmer's Guide*.
AVG_LO_SIZE	Windows: 4 to 2**31 UNIX: 2 to 2**31	8	Specifies the average size, in kilobytes, of the smart large object stored in the sbspace The database server uses this value to calculate the size of the metadata area. Do not specify AVG_LO_SIZE and **-Ms** together. You can specify AVG_LO_SIZE and the metadata offset (**-Mo**) together. If the size of the smart large object exceeds 2**31, specify 2**31. If the size of the smart large object is less than 2 on UNIX or less than 4 in Windows, specify 2 or 4. Error 131 is returned if you run out of space in the metadata and reserved areas in the sbspace. To allocate additional chunks to the sbspace that consist of metadata area only, use the **-Ms** option instead. For more information, see creating smart large objects, in the chapter on managing data on disk in the *IBM Informix Administrator's Guide*.
BUFFERING	ON or OFF	ON	Specifies the buffering mode of smart large objects stored in the sbspace If set to ON, the database server uses the buffer pool in the resident portion of shared memory for smart-large-object I/O operations. If set to OFF, the database server uses light I/O buffers in the virtual portion of shared memory (lightweight I/O operations). BUFFERING = OFF is incompatible with LOCK_MODE = RANGE and creates a conflict For more information, see lightweight I/O, in the chapter on configuration effects on memory in the *IBM Informix Performance Guide*.
LOCK_MODE	RANGE or BLOB	BLOB	Specifies the locking mode of smart large objects stored in the sbspace If set to RANGE, only a range of bytes in the smart large object is locked. If set to BLOB, the entire smart large object is locked. LOCK_MODE = RANGE is incompatible with BUFFERING = OFF and creates a conflict. For more information, see smart large objects, in the chapter on locking in the *IBM Informix Performance Guide*.
LOGGING	ON or OFF	OFF	Specifies the logging status of smart large objects stored in the sbspace If set to ON, the database server logs changes to the user data area of the sbspace. When you turn on logging for an sbspace, take a level-0 backup of the sbspace. When you turn off logging, the following message displays: You are turning off smart large object logging. For more information, see smart large objects, in the chapters on data storage and logging in the *IBM Informix Administrator's Guide*. For information about **onspaces -ch** messages, see Messages in the database server log.
EXTENT_SIZE	4 to 2**31	None	Specifies the size, in kilobytes, of the first allocation of disk space for smart large objects stored in the sbspace when you create the table Let the system select the EXTENT_SIZE value. To reduce the number of extents in a smart large object, use **mi_lo_specset_estbytes** (DataBlade API) or **ifx_lo_specset_estbytes** (Informix® ESQL/C) to hint to the system the total size of the smart large object. The system attempts to allocate a single extent for the smart large object. For more information, see smart large objects, in the chapter on where data is stored in the *IBM Informix Administrator's Guide*. For information about altering storage characteristics of smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* or the *IBM Informix ESQL/C Programmer's Manual*.
MIN_EXT_SIZE	2 to 2**31	Windows: 4UNIX: 2	Specifies the minimum amount of space, in kilobytes, to allocate for each smart large object The following message displays: Changing the sbspace minimum extent size: old value *value1* new value *value2*. For information about tuning this value, see smart large objects, in the chapter on configuration effects on I/O utilization in the *IBM Informix Performance Guide*. For information about **onspaces -ch** messages, see Messages in the database server log.

Tag	Values	Default	Description
NEXT_SIZE	4 to 2**31	None	Specifies the extent size, in kilobytes, of the next allocation of disk space for smart large objects when the initial extent in the sbspace becomes full. Let the system select the NEXT_SIZE value. To reduce the number of extents in a smart large object, use **mi_lo_specset_estbytes** or **ifx_lo_specset_estbytes** to hint to the system the total size of the smart large object. The system attempts to allocate a single extent for the smart large object. For more information, see smart large objects, in the chapter on where data is stored in the *IBM Informix Administrator's Guide*. For information about obtaining the size of smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* or the *IBM Informix ESQL/C Programmer's Manual*.

This example creates a 20-megabyte mirrored sbspace, **eg_sbsp,** with the following specifications:

- An offset of 500 kilobytes for the primary and mirror chunks
- An offset of 200 kilobytes for the metadata area
- An average expected smart-large-object size of 32 kilobytes
- Log changes to the smart large objects in the user-data area of the sbspace

UNIX Only:

```
% onspaces -c -S eg_sbsp -p /dev/raw_dev1 -o 500 -s 20000
   -m /dev/raw_dev2 500 -Mo 200 -Df "AVG_LO_SIZE=32,LOGGING=ON"
```


Changing the -Df Settings

As the database server administrator, you can override or change the **-Df** default settings in one of the following ways:

- To change the default settings for an sbspace, use the **onspaces -ch** option. For more information, refer to onspaces -ch: Change sbspace default specifications.
- To override the following **-Df** default settings for a specific table, use the SQL statements CREATE TABLE or ALTER TABLE:
  - LOGGING
  - ACCESSTIME
  - EXTENT_SIZE
  - NEXT_SIZE

  For more information on the ALTER TABLE and CREATE TABLE statements, see the *IBM® Informix Guide to SQL: Syntax*.

The programmer can override these **-Df** default settings with DataBlade API and Informix® ESQL/C functions. For information about altering storage characteristics of smart large objects, see the *IBM Informix DataBlade API Programmer's Guide* and the *IBM Informix ESQL/C Programmer's Manual*.

Using the onspaces -g option

The **onspaces -g** option is not used for sbspaces. The database server uses a different method to determine the number of pages to transfer in an I/O operation for sbspaces than for blobspaces. The database server can automatically determine the block size to transfer in an I/O operation for smart large objects. For more information, see sbspace extent sizes in the chapter on I/O activity in your *IBM® Informix® Performance Guide*.

This command has an equivalent SQL administration API function.

onspaces -c -x: Create an extspace

Use the **onspaces -c -x** option to create an extspace.

Syntax:

```
>>-onspaces -c-- -x--extspace-- -l--location-- -o--offset------->

>-- -s--size-------------------------------------------------><
```

Element	Purpose	Key Considerations
-c	Creates a dbspace, blobspace, sbspace, or extspace You can create up to 2047 storage spaces of any type.	After you create a storage space, you must back up both this storage space and the root dbspace. If you create a storage space with the same name as a deleted storage space, perform another level-0 backup to ensure that future restores do not confuse the new storage space with the old one. For more information, see creating a dbspace, blobspace, or extspace, in the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.

Element	Purpose	Key Considerations
-l *location*	Specifies the location of the extspace The access method determines the format of this string.	**Restrictions**: String. Value must not be longer than 255 bytes. For more information, see creating an extspace, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the device to reach the initial chunk of the new blobspace, dbspace, or sbspace	**Restrictions**: Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 2 or 4 terabytes, depending on the platform. For more information, see allocating raw disk space, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-s*size*	Indicates, in kilobytes, the size of the initial chunk of the new blobspace or dbspace	**Restrictions**: Unsigned integer. The size must be equal to or greater than 1000 kilobytes and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 terabytes, depending on the platform.
-x *extspace*	Names the extspace to be created	**Restrictions**: Extspace names can be up to 128 bytes. They must be unique, begin with a letter or underscore, and contain only letters, digits, underscores, or **$** characters. For more information, see extspaces, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.

Related reference:
DISK_ENCRYPTION configuration parameter

onspaces -ch: Change sbspace default specifications

Use the **onspaces -ch** option to change the default specifications of a sbspace.

Syntax:

```
>>-onspaces -ch--sbspace-- -Df--default list------------------><
```

Element	Purpose	Key Considerations
-ch	Indicates that one or more sbspace default specifications are to be changed	None.
sbspace	Names the sbspace for which to change the default specifications	Syntax must conform to the Identifier segment; see the *IBM® Informix® Guide to SQL: Syntax*. For background information, see changing default specifications of an sbspace with **onspaces** in the *IBM Informix Performance Guide*.
-Df *default list*	Lists new default specifications for smart large objects stored in the sbspace	Tags are separated by commas. If a tag is not present, system defaults take precedence. The list must be enclosed in double quotation marks (“) on the command line. For a list of tags and their parameters, see Table 1.

You can change any of the **-Df** tags with the **onspaces -ch** option. The database server applies the change to each smart large object that was created prior to changing the default specification.

For example, to turn off logging for the sbspace that you created in Creating an Sbspace with the -Df option, use the following command:

```
onspaces -ch eg_sbsp -Df "LOGGING=OFF"
```

Note: After you turn on logging for an sbspace, take a level-0 backup of the sbspace to create a point from which to recover.
Related reference:
set sbspace accesstime argument: Control access time tracking (SQL administration API)
set sbspace avg_lo_size argument: Set the average size of smart large objects (SQL administration API)
set sbspace logging argument: Change the logging of an sbspace (SQL administration API)

onspaces -cl: Clean up stray smart large objects in sbspaces

Use the **onspaces -cl** option to clean up stray smart large objects in sbspaces.

Syntax:

```
>>-onspaces -cl--sbspace----------------------------------------><
```

Element	Purpose	Key Considerations
-cl	Cleans up stray smart large objects in an sbspace	To find any stray smart large objects, use the **oncheck -pS** command when no users are connected to the database server. The smart large objects with a reference count of 0 are stray objects.

Element	Purpose	Key Considerations
sbspace	Names the sbspace to be cleaned up	Syntax must conform to the Identifier segment; see the *IBM® Informix® Guide to SQL: Syntax*.

During normal operation, no unreferenced (stray) smart large objects should exist. When you delete a smart large object, the space is released. If the database server fails or runs out of system memory while you are deleting a smart large object, the smart large object might remain as a stray object.

The following is an example of the **onspaces -cl** command:

```
onspaces -cl myspace
```

The best way to find the reference count for a smart large object is to call the **mi_lo_stat** or **ifx_lo_stat** functions from a C program. Although the **mi_lo_increfcount** and **mi_lo_decrefcount** functions return the reference count, they increment or decrement the reference count. For more information on these functions, see the *IBM Informix DataBlade API Function Reference*.

This command has an equivalent SQL administration API function.

Related reference:
clean sbspace argument: Release unreferenced smart large objects (SQL administration API)

onspaces -d: Drop a chunk in a dbspace, blobspace, or sbspace

Use the **onspaces -d** option to drop a chunk in a dbspace, blobspace, or sbspace.

Syntax:

```
>>-onspaces -d--+-dbspace----------+-- -p--pathname------------>
                +-blobspace--------+
                '-+-----+--sbspace-'
                  '- -f-'

>-- -o--offset--+-----+---------------------------------------><
                '- -y-'
```

This command has an equivalent SQL administration API function.

Element	Purpose	Key considerations
-d	Drops a chunk	You can drop a chunk from a dbspace, temporary dbspace, or sbspace when the database server is online or quiescent. For more information, see the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*. You can drop a chunk from a blobspace only when the database server is in quiescent mode.
-f	Drops an sbspace chunk that contains user data but no metadata If the chunk contains metadata for the sbspace, you must drop the entire sbspace.	Use the **-f** option with sbspaces only. If you omit the **-f** option, you cannot drop an sbspace that contains data. For more information, see dropping a chunk from an sbspace with **onspaces**, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or into the unbuffered device to reach the initial chunk of the dbspace, blobspace, or sbspace that you are dropping	**Restrictions**: Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 terabytes. For more information, see allocating raw disk space on UNIX, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
-p *pathname*	Indicates the disk partition or unbuffered device of the initial chunk of the dbspace, blobspace, or sbspace that you are dropping	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. For path name syntax, see your operating-system documentation.
-y	Causes the database server to automatically respond yes to all prompts	None.
blobspace	Names the blobspace from which the chunk is dropped	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see dropping a chunk from a blobspace, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
dbspace	Names the dbspace from which the chunk is dropped	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see dropping a chunk from a dbspace with **onspaces**, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
sbspace	Names the sbspace from which the chunk is dropped	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For background information, see dropping a chunk from a dbspace with **onspaces**, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.

Important: You must specify a path name to indicate to the database server that you are dropping a chunk.
Related reference:
drop chunk argument: Drop a chunk (SQL administration API)
drop plogspace: Drop the plogspace (SQL administration API)

onspaces -d: Drop a space

Use the **onspaces -d** option to drop a dbspace, blobspace, plogspace, sbspace, or extspace.

Syntax:

```
>>-onspaces -d--+-dbspace----------+--+-----+-----------------><
               +-blobspace--------+  '- -y-'
               +-plogspace--------+
               +-+-----+--sbspace-+
               | '- -f-'          |
               '-extspace---------'
```

Element	Purpose	Key considerations
-d	Indicates that a storage space is to be dropped	You can drop a dbspace, blobspace, plogspace, sbspace, or extspace while the database server is online or in quiescent mode. After you drop a storage space, you must back it up to ensure that the **sysutils** database and the reserved pages are up-to-date. Run **oncheck -pe** to verify that no table is storing data in the dbspace, blobspace, or sbspace.
-y	Causes the database server to automatically respond yes to all prompts	None.
-f	Drops an sbspace that contains user data and metadata	You must use the **-f** (force) option to drop an sbspace that contains data. Restriction: Use the **-f** option with sbspaces only. Warning: If you use the **-f** option, the tables in the database server might have dead pointers to the smart large objects that were deleted with this option.
blobspace	Names the blobspace to be dropped	Before you drop a blobspace, drop all tables that include a TEXT or BYTE column that references the blobspace.
dbspace	Names the dbspace to be dropped	Before you drop a dbspace, drop all databases and tables that you previously created in the dbspace.
extspace	Names the extspace to be dropped	You cannot drop an extspace if it is associated with an existing table or index.
plogspace	Names the plogspace to be dropped	The plogspace must be empty to be dropped.
sbspace	Names the sbspace to be dropped	Before you drop an sbspace, drop all tables that include a BLOB or CLOB column that references the sbspace.

Important: Do not specify a path name when you drop these storage spaces.
This command has an equivalent SQL administration API function.

Related reference:
drop blobspace argument: Drop a blobspace (SQL administration API)
drop dbspace argument: Drop a dbspace (SQL administration API)
drop sbspace argument: Drop an sbspace (SQL administration API)
drop tempdbspace argument: Drop a temporary dbspace (SQL administration API)

onspaces -f: Specify DATASKIP parameter

Use the **onspaces -f** option to specify the value of the DATASKIP configuration parameter on a dbspace level or across all dbspaces.

Syntax:

```
>>-onspaces -f--+-OFF-+--+--------------+--+-----+-------------><
               '-ON--'  '-dbspace-list-'  '- -y-'
```

This command has an equivalent SQL administration API function.

Element	Purpose	Key considerations
-f	Indicates to the database server that you want to change the DATASKIP default for specified dbspaces or all dbspaces	All changes in the DATASKIP status are recorded in the message log.
-y	Causes the database server to automatically respond yes to all prompts	None.
dbspace-list	Specifies the name of one or more dbspaces for which DATASKIP will be turned ON or OFF	Syntax must conform to the Identifier segment; see the *IBM® Informix® Guide to SQL: Syntax*. For more information, see DATASKIP Configuration Parameter and the *IBM Informix Performance Guide*.
OFF	Turns off DATASKIP	If you use OFF without *dbspace-list*, DATASKIP is turned off for all fragments. If you use OFF with *dbspace-list*, only the specified fragments are set with DATASKIP off.

Element	Purpose	Key considerations
ON	Turns on DATASKIP	If you use ON without *dbspace-list*, DATASKIP is turned on for all fragments. If you use ON with dbspace-list, only the specified fragments are set with DATASKIP on.

Related reference:
set dataskip argument: Start or stop skipping a dbspace (SQL administration API)
DATASKIP Configuration Parameter

onspaces -m: Start mirroring

Use the **onspaces -m** option to start mirroring for a dbspace, blobspace, or sbspace.

Syntax:

```
>>-onspaces -m--+-dbspace---+----------------------------------->
                +-blobspace-+
                '-sbspace---'

   .-,-------------------------------------------------.
   V                                                   |
>----+- -p--pathname-- -o--offset-- -m--pathname--offset-+-+---->
     '- -f--filename-------------------------------------'

>--+-----+-------------------------------------------------------><
   '- -y-'
```

This command has an equivalent SQL administration API function.

Element	Purpose	Key considerations
-f *filename*	Indicates that chunk-location information is in a file named *filename*	The file must be a buffered file that already exists. The path name must conform to the operating-system-specific rules for path names. For more information, see Using a File to Specify Chunk-Location Information with the -f Option.
-m	Adds mirroring for an existing dbspace, blobspace, or sbspace	User-data chunks in a mirrored sbspace need not be mirrored. The mirrored chunks should be on a different disk. You must mirror all the chunks at the same time.
-m *pathname offset*	The second time that *pathname* occurs in the syntax diagram, it indicates the disk partition or unbuffered device of the initial chunk of the dbspace, blobspace, or sbspace that performs the mirroring. The second time *offset* appears in the syntax diagram, it indicates the offset to reach the mirrored chunk of the newly mirrored dbspace, blobspace, or sbspace. Also see the entries for *pathname* and *offset* in this table.	None.
-o *offset*	The first time that *offset* occurs in the syntax diagram, it indicates, in kilobytes, the offset into the disk partition or into the unbuffered device to reach the initial chunk of the newly mirrored dbspace, blobspace, or sbspace.	**Restrictions**: Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 terabytes. For more information, see allocating raw disk space on UNIX, in the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.
-p *pathname*	The first time *pathname* occurs in the syntax diagram, it indicates the disk partition or unbuffered device of the initial chunk of the dbspace, blobspace, or sbspace that you want to mirror.	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. For path name syntax, see your operating-system documentation.
-y	Causes the database server to automatically respond yes to all prompts	None.
blobspace	Names the blobspace that you want to mirror	Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*. For more information, see the chapter on using mirroring in the *IBM Informix Administrator's Guide*.
dbspace	Names the dbspace that you want to mirror	Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*. For background information, see the chapter on using mirroring in the *IBM Informix Administrator's Guide*.
sbspace	Names the sbspace that you want to mirror	Syntax must conform to the Identifier segment; see *IBM Informix Guide to SQL: Syntax*. For background information, see the chapter on using mirroring in the *IBM Informix Administrator's Guide*.

- Using a File to Specify Chunk-Location Information with the -f Option

Related reference:
add mirror argument: Add a mirror chunk (SQL administration API)
start mirroring argument: Starts storage space mirroring (SQL administration API)

Using a File to Specify Chunk-Location Information with the -f Option

You can create a file that contains the chunk-location information. Then, when you execute **onspaces**, use the **-f** option to indicate to the database server that this information is in a file whose name you specify in ***filename***.

The contents of the file should conform to the following format, with options separated by spaces and each set of primary and mirror chunks on separate lines:

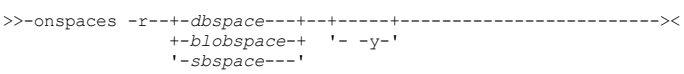
```
primary_chunk_path offset mirror_chunk_path offset
```

If the dbspace that you are mirroring contains multiple chunks, you must specify a mirror chunk for each of the primary chunks in the dbspace that you want to mirror. For an example that enables mirroring for a multichunk dbspace, see starting mirroring for unmirrored dbspaces with **onspaces** in the chapter on using mirroring in the *IBM® Informix® Administrator's Guide*.

onspaces -r: Stop mirroring

Use the **onspaces -r** option to end mirroring for a dbspace, blobspace, or sbspace.

Syntax:



This command has an equivalent SQL administration API function.

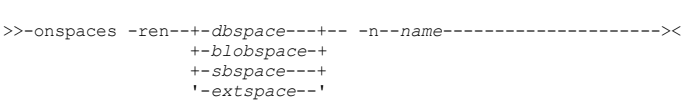
Element	Purpose	Key considerations
-r	Indicates to the database server that mirroring should be ended for an existing dbspace, blobspace, or sbspace	For background information, see the chapter on using mirroring in the *IBM® Informix® Administrator's Guide*.
-y	Causes the database server to respond yes to all prompts automatically	None.
blobspace	Names the blobspace for which you want to end mirroring.	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see the chapter on using mirroring in the *IBM Informix Administrator's Guide*.
dbspace	Names the dbspace for which you want to end mirroring.	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see the chapter on using mirroring in the *IBM Informix Administrator's Guide*.
sbspace	Names the sbspace for which you want to end mirroring	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For background information, see the chapter on using mirroring in the *IBM Informix Administrator's Guide*.

Related reference:
stop mirroring argument: Stops storage space mirroring (SQL administration API)

onspaces -ren: Rename a dbspace, blobspace, sbspace, or extspace

Use the **onspaces -ren** option to rename a dbspace, blobspace, sbspace, or extspace.

Syntax:



This command has an equivalent SQL administration API function.

Element	Purpose	Key considerations

Element	Purpose	Key considerations
-ren	Causes the database server to rename the specified blobspace, dbspace, extspace, or sbspace	**Restrictions**: You can rename a blobspace, dbspace, extspace, or sbspace when the database server is in quiescent mode. For more information, see the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.
-n *name*	Specifies the new name for the blobspace, dbspace, extspace, or sbspace	**Restrictions**: The blobspace, dbspace, external space, or sbspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the **$** character. For more information, see the chapter on managing disk space in the *IBM Informix Administrator's Guide*. The syntax must conform to the Identifier segment. For more information, see the *IBM Informix Guide to SQL: Syntax*.
blobspace	Names the blobspace to be renamed	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see renaming spaces, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
dbspace	Names the dbspace to be renamed	**Restrictions:** You cannot rename a critical dbspace, such as the root dbspace or a dbspace that contains physical logs. **Additional Information:** If you rename dbspaces that are included in the DATASKIP list, update the DATASKIP configuration parameter with the new names using the **onspaces -f** command. Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see renaming spaces, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
extspace	Names the extspace to be renamed	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see renaming spaces, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.
sbspace	Names the sbspace to be renamed	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see renaming spaces, in the chapter on managing disk space in the *IBM Informix Administrator's Guide*.

- Renaming a dbspace, blobspace, sbspace, or extspace when Enterprise Replication is active
  You can rename a space (dbspace, blobspace, sbspace, or extspace) when Enterprise Replication is active.
- Performing an Archive after Renaming a Space

Related reference:
rename space argument: Rename a storage space (SQL administration API)

Renaming a dbspace, blobspace, sbspace, or extspace when Enterprise Replication is active

You can rename a space (dbspace, blobspace, sbspace, or extspace) when Enterprise Replication is active.

When you put the database server into quiescent mode to rename the space, Enterprise Replication will be disconnected. You can then rename the space. The servers will resynchronize after you put the database server into online mode.

If you want to rename the same space on another server, you must put that server into quiescent mode and rename the space separately. No enforced relationship is propagated between renamed spaces on different ER servers; the same tables can be in different spaces.

If the Enterprise Replication server also participates in High-Availability Data Replication (HDR), you can rename the dbspace on the primary server and it will be automatically propagate to the secondary server. (The secondary server cannot participate in Enterprise Replication.)

Performing an Archive after Renaming a Space

After renaming any space (except extspaces or temporary spaces), perform a level-0 archive of the renamed space and the root dbspace. This will ensure that you can restore the spaces to a state including or following the rename dbspace operation. It is also necessary prior to performing any other type of archive.

onspaces -s: Change status of a mirrored chunk

Use the **onspaces -s** option to change the status of a mirrored chunk in a dbspace, a non-primary chunk within a noncritical dbspace, a blobspace, or an sbspace.

Syntax:

```
>>-onspaces -s--+-dbspace---+-- -p--pathname-- -o--offset------->
```

```
                +-blobspace-+
                '-sbspace---'

>--+- -D-+--+-----+-----------------------------------------><
   '- -O-'  '- -y-'
```

This command has an equivalent SQL administration API function.

Element	Purpose	Key considerations
-D	Indicates that you want to take the chunk down	None.
-o *offset*	Indicates, in kilobytes, the offset into the disk partition or unbuffered device to reach the chunk	**Restrictions**: Unsigned integer. The starting offset must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size.The offset must be a multiple of the page size. The maximum offset is 4 terabytes. For more information, see allocating raw disk space on UNIX, in the chapter on managing disk space in the *IBM® Informix® Administrator's Guide*.
-O	Indicates that you want to restore the chunk and bring it online	None.
-p *pathname*	Indicates the disk partition or unbuffered device of the chunk	The chunk can be an unbuffered device or a buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. For path name syntax, see your operating-system documentation.
-s	Indicates that you want to change the status of a chunk	**Restrictions**: You can only change the status of a chunk in a mirrored pair or a non-primary chunk within a noncritical dbspace. For more information, see changing the mirror status in the *IBM Informix Administrator's Guide*.
-y	Causes the database server to respond yes to all prompts automatically	None.
blobspace	Names the blobspace whose status you want to change	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see changing the mirror status in the *IBM Informix Administrator's Guide*.
dbspace	Names the dbspace whose status you want to change	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For more information, see changing the mirror status in the *IBM Informix Administrator's Guide*.
sbspace	Names the sbspace whose status you want to change	Syntax must conform to the Identifier segment; see the *IBM Informix Guide to SQL: Syntax*. For background information, see changing the mirror status in the *IBM Informix Administrator's Guide*.

Related reference:
alter chunk argument: Change chunk status to online or offline (SQL administration API)
set chunk argument: Change the status of a chunk (SQL administration API)

Avoid overwriting a chunk

The chunks associated with each Informix® instance are not known to other Informix instances. It is possible to inadvertently create a chunk on a file or device that is allocated as a chunk to another Informix instance, which results in data corruption.

If you attempt to initialize an instance, where the ROOTPATH configuration parameter specifies a file or device that is the root chunk of another instance, the command fails with the following message in the online.log:

```
DISK INITIALIZATION ABORTED: potential instance overwrite detected.
```

To disable this initialization check, set the FULL_DISK_INIT configuration parameter to 1 in your configuration file and try to initialize the instance again. However, this initialization check is restricted to the root chunk. Adding dbspaces or chunks succeeds even when the file or device is allocated to another instance.

Related reference:
onspaces -a: Add a chunk to a dbspace or blobspace
onspaces -a: Add a chunk to an sbspace
onspaces -c -b: Create a blobspace
onspaces -c -d: Create a dbspace
onspaces -c -S: Create an sbspace
create blobspace argument: Create a blobspace (SQL administration API)
create chunk argument: Create a chunk (SQL administration API)
create dbspace argument: Create a dbspace (SQL administration API)
create sbspace argument: Create an sbspace (SQL administration API)

The onstat utility

The **onstat** utility reads shared-memory structures and provides statistics about the database server at the time that the command runs.

You can combine multiple **onstat** option flags in a single command. The contents of shared memory might change as the **onstat** output displays. The **onstat** utility does not place any locks on shared memory, so running the utility does not affect performance.

You use SQL administration API commands that are equivalent to **onstat** commands.

- onstat Portal: onstat Utility Commands Sorted by Functional Category
  The information in this topic lists **onstat** commands that are sorted by functional category.
- Monitor the database server status
  To monitor the database server status, view the heading of the **onstat** command.
- onstat command syntax
  The complete syntax for the **onstat** command, including information about the interactive mode and how to have options to execute repeatedly.
- onstat command: Equivalent to the onstat -pu command
  If you invoke **onstat** without any options, the command is interpreted as **onstat -pu** (the **-p** option and the **-u** option).
- onstat - command: Print output header
  All **onstat** output includes a header. The **onstat -** command displays only the output header and the value that is returned from this command indicates the database server mode.
- onstat -- command: Print onstat options and functions
  Use the **onstat --** command to display a listing of all of the **onstat** options and their functions. You cannot combine this option with any other flag.
- Running onstat Commands on a Shared Memory Dump File
  You can run **onstat** commands against a shared memory dump file. The shared memory dump file can be produced explicitly by using the **onstat -o** command. If the DUMPSHMEM configuration parameter is set to 1 or set to 2, the dump file is created automatically at the time of an assertion failure.
- onstat -a command: Print overall status of the database server
  Use the **onstat -a** command to display information about the status of the database server. This command does not display information about all of the onstat options, only about those onstat options used for initial troubleshooting.
- onstat -b command: Print buffer information for buffers in use
  Use the **onstat -b** option to display information about the buffers that are currently in use, including the total number of resident pages in the buffer pool.
- onstat -B command: Prints information about used buffers
  Use the **onstat -B** option to display information about buffers that are not on the free-list.
- onstat -c command: Print ONCONFIG file contents
  Use the **onstat -c** command to display the contents of the ONCONFIG file.
- onstat -C command: Print B-tree scanner information
  Use the **-C** command to display information about the B-tree scanner subsystem and each B-tree scanner thread.
- onstat -d command: Print chunk information
  Use the **onstat -d** command to show information about chunks in each storage space.
- onstat -D command: Print page-read and page-write information
  Use the **onstat -D** command to display page-read and page-write information for the first 50 chunks in each space.
- onstat -f command: Print dbspace information affected by dataskip
  Use the **-f** command to list the dbspaces that the dataskip feature currently affects.
- onstat -F command: Print counts
  Use the **onstat -F** command to display a count for each type of write that flushes pages to disk.
- onstat -g monitoring options
  The options that you can use with **onstat -g** command are used for support and debugging only. You can include only one of these options in the **onstat -g** command.
- onstat -G command: Print TP/XA transaction information
  Use the **onstat -G** command to display information about global transactions generated through the TP/XA library.
- onstat -h command: Print buffer header hash chain information
  Use the **onstat -h** command to display information about the buffer header hash chains (sometimes called "hash buckets") that are used to access pages in each buffer pool.
- onstat -i command: Initiate interactive mode
  Use the **onstat -i** command to put the **onstat** utility in the interactive mode.
- onstat -j command: Provide onpload status information
  Use the **onstat -j** command to provide information about the status of an **onpload** job.
- onstat -k command: Print active lock information
  Use the **onstat -k** command to print information about active locks, including the address of the lock in the lock table.
- onstat -l command: Print physical and logical log information
  Use the **onstat -l** command to display information about the physical logs, logical logs, and temporary logical logs.
- onstat -L command: Print the number of free locks
  Use the **onstat -L** command to print the number of free locks on a lock-free list.
- onstat -m command: Print recent system message log information
  Use the **onstat -m** command to display the 20 most recent lines of the system message log.
- onstat -o command: Output shared memory contents to a file
  Use the **onstat -o** command to write the contents of shared memory to a specified file for later analysis. If you do not specify an output file, a file named **onstat.out** is created in the current directory.
- onstat -p command: Print profile counts
  Use the **onstat -p** command to display information about profile counts either since you started the database server or since you ran the **onstat -z** command.
- onstat -P command: Print partition information
  Use the **onstat -P** command to display the partition number and the pages in the buffer pool for all of the partitions.
- onstat -r command: Repeatedly print selected statistics
  Use the **onstat -r** command to repeatedly print the statistics for other options specified in the command at specified intervals.
- onstat -R command: Print LRU, FLRU, and MLRU queue information
  Use the **onstat -R** command to display detailed information about the LRU queues, FLRU queues, and MLRU queues. For each queue, the **onstat -R** command displays the number of buffers in the queue and the number and percentage of buffers that have been modified.
- onstat -s command: Print latch information
  Use the **onstat -s** command to display general latch information, including the resource that the latch controls.
- onstat -t and onstat -T commands: Print tblspace information
  Use the **onstat -t** command to display tblspace information for active tblspaces. Use the **onstat -T** command to display tblspace information for all tblspaces.
- onstat -u command: Print user activity profile
  Use the **onstat -u** command to display a profile of user activity.

- onstat -x command: Print database server transaction information
  Use the **onstat -x** command to display transaction information on the database server.
- onstat -X command: Print thread information
  Use the **onstat -X** command to obtain precise information about the threads that are waiting for buffers.
- onstat -z command: Clear statistics
  Use the **onstat -z** command to clear database server statistics, including statistics that relate to Enterprise Replication, and set the profile counts to 0.
- Return codes on exiting the onstat utility
  The **onstat** utility displays a set of return codes when you exit the utility.

Related reference:
onstat argument: Monitor the database server (SQL administration API)

onstat Portal: onstat Utility Commands Sorted by Functional Category

The information in this topic lists **onstat** commands that are sorted by functional category.

Each category represents a different IBM® Informix® feature for which **onstat** commands are useful for providing troubleshooting and performance enhancement information. Commands that appear in **bold** typeface are especially useful for providing troubleshooting information. Certain **onstat** commands are specific to one category, while others provide more general information and are listed in more than one category.

Category List

Determine the appropriate category from the following list, then follow the link to the **onstat** options for that category.

onstat Utility Archive Information Options

Use the following **onstat** options to display information about archives and restores.

Table 1. onstat Utility Archive Information Options

Commands	Reference
onstat -D	Prints chunk I/O activity. Prints dbspace read/write activity for monitoring restore progress. onstat -D command: Print page-read and page-write information
onstat -g arc	Prints the last committed and any ongoing backups for each dbspace. onstat -g arc command: Print archive status

onstat Utility Cache Information Options

Use the following **onstat** options to display information about caches and cached data, including buffer pools.

Table 2. onstat Utility Cache Information Options

Commands	Reference
onstat -b	Prints buffer pages in use. onstat -b command: Print buffer information for buffers in use
onstat -B	Prints information about used buffers. onstat -B command: Prints information about used buffers

Commands	Reference
onstat -F	Prints state of buffer queue cleaners and I/O. onstat -F command: Print counts
onstat -g cac	Prints summary and detailed information about all memory caches or about the specified cache. onstat -g cac command: Print information about caches
onstat -g dic	Prints data dictionary cache, containing system catalog data for tables. Prints one line of information for each table that is cached in the shared-memory dictionary. For more information, see your *IBM Informix Performance Guide*. onstat -g dic command: Print table information
onstat -g dsc	Prints table distribution statistics for the optimizer. onstat -g dsc command: Print distribution cache information.
onstat -g prc	Prints the stored procedure (SPL) routine cache. Prints information about SPL routine cache. onstat -g prc command: Print sessions using UDR or SPL routines
onstat -g ssc	Prints the number of times that the database server reads the SQL statement in the cache. Displays the same output as **onstat -g cac**. For more information, see improving query performance in the *IBM Informix Performance Guide*. onstat -g ssc command: Print SQL statement occurrences
onstat -g vpcache	Prints CPU virtual processor memory cache. onstat -g vpcache command: Print CPU virtual processor and tenant virtual processor private memory cache statistics
onstat -h	Prints buffer hash chain information. onstat -h command: Print buffer header hash chain information
onstat -p	Prints global (server) information regarding the effectiveness of buffer pool caching. onstat -p command: Print profile counts
onstat -X	Prints threads that are waiting for buffers. onstat -X command: Print thread information

onstat Utility Compression Options

Use the following **onstat** options to print compression information.

Table 3. onstat Utility Compression Options

Commands	Reference
onstat -g dsk	Prints progress of currently running compression operations. onstat -g dsk command: Print the progress of the currently running compression operation
onstat -g ppd	Prints partition compression dictionary information. onstat -g ppd command: Print partition compression dictionary information

onstat Utility Debugging Options

Use the following **onstat** options to display information that is useful for debugging problems with the server.

Table 4. onstat Utility Debugging Options

Commands	Reference
onstat -g dmp	Prints raw memory at a specified address for a number of given bytes. onstat -g dmp command: Print raw memory
onstat -g src	Searches for patterns in shared memory. Note that memory is byte-swapped on Intel platforms. onstat -g src command: Patterns in shared memory
onstat -o	Prints shared memory contents to a file. onstat -o command: Output shared memory contents to a file.

onstat Utility Enterprise Replication Options

Use the following **onstat** options to track Enterprise Replication statistics and to provide troubleshooting information. For additional information about Enterprise Replication see the **cdr view** and **cdr view profile** commands that are described in the .

Table 5. onstat Utility Enterprise Replication Options

Commands	Reference
onstat -g cat	Prints information from the Enterprise Replication global catalog. The global catalog contains a summary of information about the defined servers, replicates, and replicate sets on each of the servers within the enterprise. onstat -g cat: Print ER global catalog information
onstat -g cdr	Prints the output for all of the Enterprise Replication statistics commands. onstat -g cdr: Print ER statistics
onstat -g cdr config	Prints Enterprise Replication configuration parameters and environment variables. onstat -g cdr config: Print ER settings
onstat -g ddr	Prints status of Enterprise Replication components that read and process log records. onstat -g ddr: Print status of ER log reader
onstat -g dss	Prints activity of individual data sync (transaction processing) threads. onstat -g dss: Print statistics for data sync threads
onstat -g dtc	Prints delete table cleaner activity. Deleted or updated rows that are placed in the delete table are purged at intervals. onstat -g dtc: Print statistics about delete table cleaner
onstat -g grp	Prints Enterprise Replication grouper statistics. The grouper evaluates the log records, rebuilds the individual log records into the original transaction, packages the transaction, and queues the transaction for transmission. onstat -g grp: Print grouper statistics
onstat -g nif	Prints network interface statistics. Shows the state of the network interface, servers, and data transfer among servers. onstat -g nif: Print statistics about the network interface
onstat -g que	Prints statistics for the high-level queue interface (which is common to all of the queues of the Enterprise Replication Queue Manager). onstat -g que: Print statistics for all ER queues
onstat -g rcv	Prints receive manager statistics. onstat -g rcv: Print statistics about the receive manager
onstat -g rep	Prints events that are in the queue for the schedule manager. onstat -g rep: Prints the schedule manager queue
onstat -g rqm	Prints statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM). onstat -g rqm: Prints statistics for RQM queues
onstat -g sync	Prints synchronization status. onstat -g sync: Print statistics about synchronization

onstat Utility High-Availability Replication Options

Use the following **onstat** options to monitor high-availability cluster environments and the Connection Manager.

Table 6. onstat Utility High-Availability Replication Options

Commands	Reference
onstat -g cluster	Prints high-availability cluster information. onstat -g cluster command: Print high-availability cluster information
onstat -g cmsm	Prints Connection Manager information. onstat -g cmsm command: Print Connection Manager information
onstat -g dri	Prints data-replication information. See *Monitoring High-Availability Data-Replication status* in the *IBM Informix Administrator's Guide*. onstat -g dri command: Print high-availability data replication information.
onstat -g ipl	Prints index page logging status. onstat -g ipl command: Print index page logging status information
onstat -g laq	Prints information about log recovery apply queues. onstat -g laq command: Print log apply queues
onstat -g proxy	Prints proxy distributors for high-availability. onstat -g proxy command: Print proxy distributor information

Commands	Reference
onstat -g rss	Prints remote stand-alone server (RSS) information. onstat -g rss command: Print RS secondary server information
onstat -g sds	Prints shared disk secondary (SDS) server information. onstat -g sds command: Print SD secondary server information
onstat -g smx	Prints Server Multiplexer Group (SMX) connections in high-availability environments. Prints data transfer statistics and encryption status. onstat -g smx command: Print multiplexer group information

onstat Utility Informix Warehouse Accelerator Options

Use the following **onstat** options to display information that is exchanged between the database server and the Informix Warehouse Accelerator.

Table 7. onstat Utility Informix Warehouse Accelerator options

Commands	Reference
onstat -g aqt	Prints information about the data marts and the associated accelerated query tables (AQTs). onstat -g aqt command: Print data mart and accelerated query table information.

onstat Utility I/O Options

Use the following **onstat** options to track input and output (read and write) activity.

Table 8. onstat Utility I/O Options

Commands	Reference
onstat -D	Prints chunk I/O activity. onstat -D command: Print page-read and page-write information
onstat -g cpu	Prints runtime statistics for each thread. onstat -g cpu: Print runtime statistics
onstat -g ioa	Prints combined information from **onstat -g ioq** (queues), **onstat -g iov** (virtual processors), and **onstat -g iob** (big buffer). onstat -g ioa command: Print combined onstat -g information
onstat -g iob	Prints the big buffer usage summary. onstat -g iob command: Print big buffer use summary
onstat -g iof	Prints I/O statistics by file or chunk. This option is similar to the **onstat -D** option, but also displays information about non-chunk, temporary, and sort-work files. onstat -g iof command: Print asynchronous I/O statistics
onstat -g iog	Prints AIO global information. onstat -g iog command: Print AIO global information
onstat -g ioq	Prints queue read/write statistics and queue length. onstat -g ioq command: Print I/O queue information. Also see the *IBM Informix Performance Guide*.
onstat -g iov	Prints asynchronous I/O statistics by virtual processor. onstat -g iov command: Print AIO VP statistics
onstat -p	Prints global disk activity, including sequential scans. onstat -p command: Print profile counts

onstat Utility Locks and Latches Options

Use the following **onstat** options to display information about locks.

Table 9. onstat Utility Locks and Latches Options

Commands	Reference
onstat -k	Prints information about active locks. onstat -k command: Print active lock information
onstat -L	Prints the number of locks on a lock free list. onstat -L command: Print the number of free locks

Commands	Reference
onstat -p	Prints global statistics on lock requests, lock waits, and latch waits. onstat -p command: Print profile counts
onstat -s	Prints latch (mutex) information. onstat -s command: Print latch information

onstat Utility Logs Options

Use the following **onstat** options to monitor logical and physical logs.

Table 10. onstat Utility Logs Options

Commands	Reference
onstat -g ipl	Prints index page logging information in high-availability environments. onstat -g ipl command: Print index page logging status information
onstat -l	Prints status of physical and logical logs, and log buffering. onstat -l command: Print physical and logical log information

onstat Utility Memory Options

Use the following **onstat** options to monitor the various aspects of server memory allocation and use.

Table 11. onstat Utility Memory Options

Commands	Reference
onstat -g afr	Prints allocated memory fragments for a specified session or shared-memory pool. To obtain the pool name, see the **onstat -g mem** option. onstat -g afr command: Print allocated memory fragments
onstat -g ffr (*pool name session ID*)	Prints free fragments for a session or shared memory pool. onstat -g ffr command: Print free fragments
onstat -g lmm	Prints information about automatic low memory management settings and recent activity: onstat -g lmm command: Print low memory management information
onstat -g mem	Prints session or pool virtual shared memory statistics. onstat -g mem command: Print pool memory statistics
onstat -g mgm	Prints Memory Grant Manager (parallel and sort operations) resource information. onstat -g mgm command: Print MGM resource information. Also see the *IBM Informix Performance Guide*.
onstat -g nbm	Prints block map for non-resident segments. onstat -g nbm command: Print a block bit map
onstat -g rbm	Prints block map for resident segment. onstat -g rbm command: Print a block map of shared memory
onstat -g seg	Prints memory segment statistics. onstat -g seg command: Print shared memory segment statistics. Also see the *IBM Informix Administrator's Guide*.
onstat -g ses	Prints session information, including memory breakdown. For detailed information, use: **onstat -g ses** *session_id* onstat -g ses command: Print session-related information Also see the *IBM Informix Performance Guide*
onstat -g stm	Prints SQL statement memory use. onstat -g stm command: Print SQL statement memory usage
onstat -g stq	Prints stream queue buffers. onstat -g stq command: Print queue information
onstat -g ufr	Prints memory pool fragments for a session or shared memory pool in use. onstat -g ufr command: Print memory pool fragments
onstat -R	Prints buffer pool queues and their status. onstat -R command: Print LRU, FLRU, and MLRU queue information

onstat Utility Network Options

Use the following **onstat** options to monitor shared memory and network connection services.

Table 12. onstat Utility Network Options

Commands	Reference
onstat -g imc	Prints information about Informix MaxConnect instances that are connected to the database server. If Informix MaxConnect is not connected to the database server, this command displays `No MaxConnect servers are connected`.
onstat -g nsc	Prints shared-memory status by *client id*. If *client id* is omitted, all client status areas are displayed. This command prints the same status data as the **nss** command. onstat -g nsc command: Print current shared memory connection information
onstat -g nsd	Prints network shared-memory data for poll threads. onstat -g nsd command: Print poll threads shared-memory data
onstat -g nss	Prints network shared-memory status by *session id*. If *session id* is omitted, all session status areas are displayed. This command prints the same status data as the **onstat -g nsc** command. onstat -g nss command: Print shared memory network connections status
onstat -g nta	Prints combined network statistics from **onstat -g ntd**, **onstat -g ntm**, **onstat -g ntt**, and **onstat -g ntu**. If Informix MaxConnect is installed, this command prints statistics that you can use to tune Informix MaxConnect performance.
onstat -g ntd	Prints network statistics by service. onstat -g ntd command: Print network statistics
onstat -g ntm	Prints network mail statistics. onstat -g ntm command: Print network mail statistics
onstat -g ntt	Prints network user times. onstat -g ntt command: Print network user times
onstat -g ntu	Prints network user statistics. onstat -g ntu command: Print network user statistics

onstat Utility Performance Checks (First Tier)

Use the following **onstat** options to monitor performance and to check for performance impediments. Use the second-tier **onstat** options (and other **onstat** commands) to further narrow the problem.

Table 13. onstat Utility Performance Checks (First Tier)

Commands	Reference
onstat -c	Prints server configuration. onstat -c command: Print ONCONFIG file contents
onstat -D	Prints chunk I/O. onstat -D command: Print page-read and page-write information
onstat -g ath	Prints status and statistics for all threads. The **sqlexec** thread is a client session thread. The **rstcb** value corresponds to the user field of the **onstat -u** command. onstat -g ath command: Print information about all threads. For information about using **onstat -g ath** to print Enterprise Replication threads, see the .
onstat -g ckp	Prints checkpoint history and display configuration recommendations. onstat -g ckp command: Print checkpoint history and configuration recommendations
onstat -g cpu	Prints runtime statistics for each thread. onstat -g cpu: Print runtime statistics
onstat -g ioq	Prints pending I/O operations for the *queue name*. onstat -g ioq command: Print I/O queue information
onstat -p	Prints global server performance profile. onstat -p command: Print profile counts
onstat -u	Prints status and statistics for user threads. If a thread is waiting for a resource, this command identifies the type (flags field) and address (wait field) of the resource. onstat -u command: Print user activity profile

onstat Utility Performance Checks (Second Tier)

Use the following **onstat** options to identify performance impediments.

Table 14. onstat Utility Performance Checks (Second Tier)

Commands	Reference

Commands	Reference
onstat -b	Prints active buffers. onstat -b command: Print buffer information for buffers in use
onstat -g act	Prints active threads. onstat -g act command: Print active threads
onstat -g glo	Prints virtual processors and their operating system processes (**oninit** processes). Prints virtual processor CPU use. On Windows, the virtual processors are operating system threads, and the values in the **pid** field are thread IDs. onstat -g glo command: Print global multithreading information
onstat -g mgm	Prints Memory Grant Manager resource information. onstat -g mgm command: Print MGM resource information
onstat -g rah	Prints read-ahead request information onstat -g rah command: Print read-ahead request statistics
onstat -g rea	Prints threads in the ready queue that are waiting for CPU resources. onstat -g rea command: Print ready threads
onstat -g seg	Prints shared-memory-segment statistics. This option shows the number and size of shared-memory segments that are allocated to the database server. onstat -g seg command: Print shared memory segment statistics.
onstat -g wai	Prints waiting threads; all threads that are waiting for mutex or condition, or yielding. onstat -g wai command: Print wait queue thread list
onstat -k	Prints active locks. onstat -k command: Print active lock information

onstat Utility Table Options

Use the following **onstat** options to display information about table status and table statistics.

Table 15. onstat Utility Table Options

Commands	Reference
onstat -g buf	Prints buffer pool profile information. onstat -g buf command: Print buffer pool profile information
onstat -g lap	Prints information about the status of currently active light appends (writes bypassing the buffer pool). onstat -g lap command: Print light appends status information
onstat -g opn	Prints open partitions (tables). onstat -g opn command: Print open partitions
onstat -g ppf	Prints partition profile (activity data) for the specified partition number or prints profiles for all partitions. onstat -g ppf command: Print partition profiles
onstat -g scn	Prints information about the progress of a scan, based on rows scanned on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data, and identifies whether a scan is a light or bufferpool scan. onstat -g scn command: Print scan information
onstat -P	Prints table and B-tree pages in the buffer pool, listed by partition (table). onstat -P command: Print partition information
onstat -t **onstat -T**	Prints basic tblspace (partition) information for active (t) or all (T) tblspaces. onstat -t and onstat -T commands: Print tblspace information

onstat Utility Thread Options

Use the following **onstat** options to display the status and activity of threads.

Table 16. onstat Utility Thread Options

Commands	Reference
onstat -g act	Prints active threads. This output is included in **onstat -g ath** output. onstat -g act command: Print active threads

Commands	Reference
onstat -g ath	Prints all threads. onstat -g ath command: Print information about all threads. For information about using **onstat -g ath** to print Enterprise Replication threads, see the .
onstat -g bth	Displays the dependencies between blocking and waiting threads. onstat -g bth and -g BTH: Print blocked and waiting threads
onstat -g BTH	Displays session and stack information for the blocking threads. onstat -g bth and -g BTH: Print blocked and waiting threads
onstat -g cpu	Prints runtime statistics for each thread. onstat -g cpu: Print runtime statistics
onstat -g rea	Prints ready threads (threads that are waiting for CPU resources). This output is included in the **onstat -g ath** output. onstat -g rea command: Print ready threads.
onstat -g sle	Prints information about threads that are sleeping for a specified time. Does not include threads that are sleeping forever. onstat -g sle command: Print all sleeping threads
onstat -g stk	Prints the stack of a specified thread or prints stacks for all threads. onstat -g stk command: Print thread stack
onstat -g sts	Prints maximum and current stack use per thread. onstat -g sts command: Print stack usage for each thread
onstat -g tpf	Prints thread activity statistics. onstat -g tpf command: Print thread profiles
onstat -g wai	Prints waiting (idle, sleeping, and waiting) threads. Included in **onstat -g ath** output. onstat -g wai command: Print wait queue thread list
onstat -g wst	Prints wait statistics for threads. onstat -g wst command: Print wait statistics for threads

onstat Utility User/Session Options

Use the following **onstat** options to display information about the user environment and active sessions.

Table 17. onstat Utility User/Session Options

Commands	Reference
onstat -g env	Prints the values of environment variables the database server is using. onstat -g env command: Print environment variable values
onstat -g his	Prints SQL tracing information. onstat -g his command: Print SQL trace information
onstat -g pqs	Prints operators that are used in currently running SQL queries. onstat -g pqs command: Print operators for all SQL queries
onstat -g ses	Prints summary information for all active sessions or detailed information for individual sessions. onstat -g ses command: Print session-related information
onstat -g spf	Prints prepared statement profiles for all active sessions. onstat -g spf: Print prepared statement profiles
onstat -g sql	Prints SQL information for all active sessions or detailed SQL information for individual sessions. onstat -g sql command: Print SQL-related session information
onstat -G	Prints global transactions. onstat -G command: Print TP/XA transaction information
onstat -u	Prints status of user threads and their global read/write statistics. onstat -u command: Print user activity profile
onstat -x	Prints information about transactions. onstat -x command: Print database server transaction information

onstat Utility Virtual Processor Options

Use the following **onstat** options to display information and statistics for virtual processors.

Table 18. onstat Utility Virtual Processor Options

Commands	Reference
onstat -g glo	Prints global multithreading information and global statistics for virtual processor classes and individual virtual processors. On Windows, the virtual processors are operating system threads, and the values in the **pid** field are thread IDs. onstat -g glo command: Print global multithreading information
onstat -g sch	Prints the number of semaphore operations, spins, and busy waits for each virtual processor. On Windows, the virtual processors are operating system threads, and the values in the **pid** field are thread IDs. onstat -g sch command: Print VP information

onstat Utility Waiting Options

Use the following **onstat** options to display information about wait conditions for threads.

Table 19. onstat Utility Waiting Options

Commands	Reference
onstat -g con	Prints IDs of threads that are waiting for conditions. **onstat -g ath** to print thread information. See onstat -g con command: Print condition and thread information
onstat -g lmx	Prints all locked mutexes. onstat -g lmx command: Print all locked mutexes
onstat -g qst	Prints queue-wait statistics for mutex and condition queues. onstat -g qst command: Print wait options for mutex and condition queues
onstat -g rwm	Prints read/write mutexes. onstat -g rwm command: Print read and write mutexes
onstat -g spi	Prints spin locks with long spins and spin lock statistics. onstat -g spi command: Print spin locks with long spins
onstat -g wai	Prints waiting threads; all threads that are waiting for mutex or condition, or yielding. onstat -g wai command: Print wait queue thread list
onstat -g wmx	Prints all mutexes with waiters. onstat -g wmx command: Print all mutexes with waiters

Other Useful onstat Utility Options

Table 20. Other Useful onstat Utility Options

Commands	Reference
onstat -	Prints **onstat** header; includes engine version, status (online, Quiescent, and so on), elapsed time since initialization, and memory footprint. onstat - command: Print output header
onstat –	Prints **onstat** usage options. onstat -- command: Print onstat options and functions
onstat *options infile*	Print **onstat** output using a shared memory dump (infile) as input. Running onstat Commands on a Shared Memory Dump File
onstat -a	Prints collective **onstat** outputs. onstat -a command: Print overall status of the database server
onstat -c	Prints the server configuration file. onstat -c command: Print ONCONFIG file contents
onstat -C	Prints B-tree index scanner information (shows statistics about index cleaning). onstat -C command: Print B-tree scanner information
onstat -d	Prints chunk information. onstat -d command: Print chunk information
onstat -f	Prints dbspaces configured for dataskip. onstat -f command: Print dbspace information affected by dataskip
onstat -g all	Prints diagnostic information. onstat -g all command: Print diagnostic information

Commands	Reference
onstat -g cfg	Prints a list of configuration parameters with their current values. onstat -g cfg command: Print the current values of configuration parameters
onstat -g dbc	Prints statistics about dbScheduler and dbWorker threads. onstat -g dbc command: Print dbScheduler and dbWorker thread statistics
onstat -g dis	Prints a list of database servers, their status, directory location, configuration information, and host name. onstat -g dis command: Print database server information
onstat -g dll	Prints a list of dynamic libraries that are loaded. onstat -g dll command: Print dynamic link library file list
onstat -g osi	Prints information about operating system resources and parameters. onstat -g osi: Print operating system information
onstat -g pos	Prints values from $INFORMIXDIR/etc/.infos.servernum file, which are used by clients such as **onmode** for shared memory connections to the server. **onmode -R** rebuilds the $INFORMIXDIR/etc/.infos.servernum file. onstat -g pos command: Print file values
onstat -g smb	Prints detailed information about sbspaces. onstat -g smb command: Print sbspaces information
onstat -g sym	Prints symbol table information for the **oninit** utility. onstat -g sym command: Print symbol table information for the oninit utility
onstat -i	Changes **onstat** mode to interactive. onstat -i command: Initiate interactive mode
onstat -j	Prints information about the status of an **onpload** job. onstat -j command: Provide onpload status information
onstat -m	Prints message log contents. onstat -m command: Print recent system message log information
onstat -r	Prints repetitive **onstat** execution. onstat -r command: Repeatedly print selected statistics
onstat -z	Resets the accumulated statistics to zero. onstat -z command: Clear statistics

Monitor the database server status

To monitor the database server status, view the heading of the **onstat** command.

Whenever the database server is blocked, **onstat** displays the following line after the banner line:

```
Blocked: reason
```

The variable *reason* can be one or more of the following values.

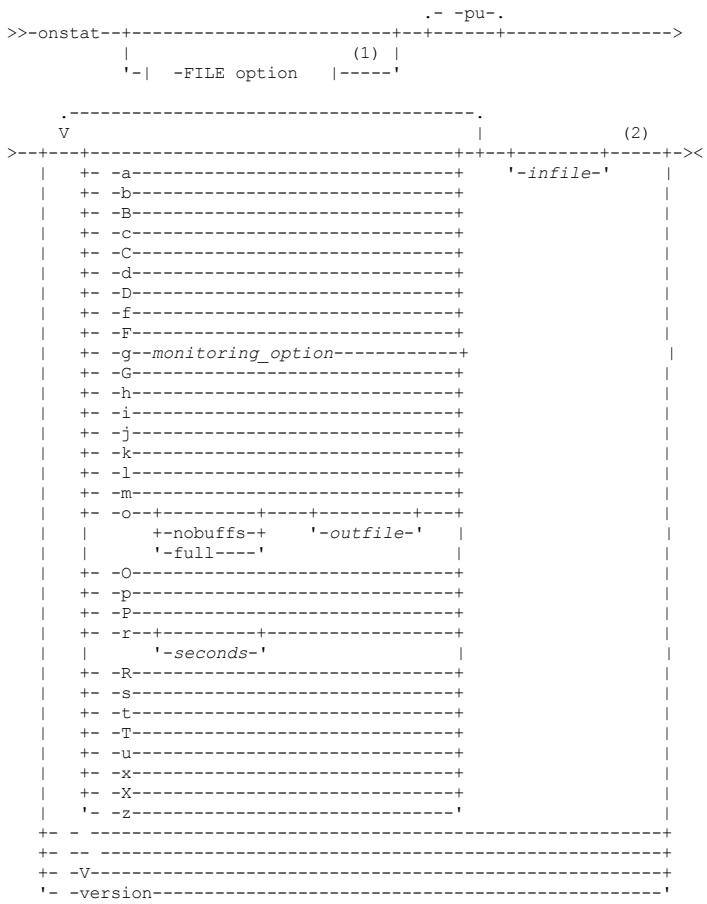
Reason	Description
ADMINISTRATION	Database is in administration mode
ARCHIVE	Ongoing storage-space backup
ARCHIVE_EBR	Blocked for External Backup and Recovery.
CHG_PLOG	Blocked while physical log is being changed.
CKPT	Checkpoint
CKPT INP	Interval checkpoint in progress
DBS_DROP	Dropping a dbspace
DDR	Discrete data replication
DYNAMIC_LOG	Log file is being added dynamically
DYNAMIC_LOG_FOR_ER	Log file is being added dynamically in ER setup
FREE_LOG	Log file is being freed
HA_CONV_STD	Blocked while High Availability server is being converted to standard server.
HA_FAILOVER	Blocked while High Availability server failover being processed.

HANG_SYSTEM	Database server failure
LAST_LOG_RESERVED4BACKUP	Waiting for last available log to be backed up
LBU	Logs full high-watermark
LOG_DROP	Log file is being dropped
LONGTX	Long transaction
MEDIA_FAILURE	Media failure
OVERRIDE_DOWN_SPACE	Waiting to override down dbspace setting because the ONDBSPACEDOWN onconfig parameter is set to WAIT

In this table, the value CHKP INP does not indicate that the database server is blocked, but that a nonblocking interval checkpoint is in progress while the buffer pool is being flushed. This CHKP INP value appears in the status line of **onstat** output until all pages in the shared-memory buffer pool have been written to disk. For information about setting interval checkpoints to flush the buffer pool, see the CKPTINTVL configuration parameter.

onstat command syntax

The complete syntax for the **onstat** command, including information about the interactive mode and how to have options to execute repeatedly.



Notes:

1. See The -FILE option.
2. Only one occurrence of each item is allowed. More than one option can be specified on a single **onstat** command invocation.

Element	Purpose	Key Considerations
-	Displays the output header only.	See onstat - command: Print output header.
--	Displays a listing of all **onstat** options and their functions	See onstat -- command: Print onstat options and functions. This option cannot be combined with any other **onstat** option.
-a	Interpreted as **onstat -cuskbtdlp**. Displays output in that order.	See onstat -a command: Print overall status of the database server.
-b	Displays information about buffers currently in use, including number of resident pages in the buffer pool	See onstat -b command: Print buffer information for buffers in use.

Element	Purpose	Key Considerations
-B	Obtains information about all database server buffers, not just buffers currently in use.	See onstat -B command: Prints information about used buffers.
-c	Displays the ONCONFIG file: • $INFORMIXDIR/etc/$ONCONFIG for UNIX • %INFORMIXDIR%\etc\ %ONCONFIG% for Windows	See onstat -c command: Print ONCONFIG file contents.
-C	Prints B-tree scanner information	See onstat -C command: Print B-tree scanner information.
-d	Displays information for chunks in each storage space	See onstat -d command: Print chunk information.
-D	Displays page-read and page-write information for the first 50 chunks in each dbspace	See onstat -D command: Print page-read and page-write information.
-f	Lists the dbspaces currently affected by the DATASKIP feature	See onstat -f command: Print dbspace information affected by dataskip.
-F	Displays a count for each type of write that flushes pages to disk	See onstat -F command: Print counts.
-g *option*	Prints monitoring option	See onstat -g monitoring options.
-G	Prints global transaction IDs	See onstat -G command: Print TP/XA transaction information.
-h	Provides information on the buffer header hash chains	See onstat -h command: Print buffer header hash chain information.
-i	Puts the **onstat** utility into interactive mode	See onstat -i command: Initiate interactive mode.
-j	Prints the interactive status of the active **onpload** process	See onstat -j command: Provide onpload status information.
-k	Displays information about active locks	See onstat -k command: Print active lock information.
-l	Displays information about physical and logical logs, including page addresses	See onstat -l command: Print physical and logical log information.
-m	Displays the 20 most recent lines of the database server message log	Output from this option lists the full pathname of the message-log file and the 20 file entries. A date-and-time header separates the entries for each day. A time stamp prefaces single entries within each day. The name of the message log is specified as MSGPATH in the ONCONFIG file. See onstat -m command: Print recent system message log information.
-o	Saves a copy of the shared-memory segments to *outfile*	See onstat -o command: Output shared memory contents to a file.
-p	Displays profile counts.	See onstat -p command: Print profile counts.
-P	Displays for all partitions the partition number and the break-up of the buffer-pool pages that belong to the partition	See onstat -P command: Print partition information.
-pu	If you invoke **onstat** without any options, the command is interpreted as **onstat -pu** (-p option and -u option). Displays profile counts and prints a profile of user activity	See onstat -p command: Print profile counts and onstat -u command: Print user activity profile.
-r *seconds*	Repeats the accompanying **onstat** options after a wait time specified in *seconds* between each execution	See onstat -r command: Repeatedly print selected statistics.
-R	Displays detailed information about the LRU queues, FLRU queues, and MLRU queues	See onstat -R command: Print LRU, FLRU, and MLRU queue information.
-s	Displays general latch information	See onstat -s command: Print latch information.
-t	Displays tblspace information, including residency state, for active tblspaces	See onstat -t and onstat -T commands: Print tblspace information.
-T	Displays tblspace information for all tblspaces	See onstat -t and onstat -T commands: Print tblspace information.
-u	Prints a profile of user activity	See onstat -u command: Print user activity profile.
-**V**	Displays the software version number and the serial number. This option cannot be combined with any other **onstat** option.	See Obtaining utility version information.
-**version**	Displays the build version, host, OS, number and date, as well as the GLS version. This option cannot be combined with any other **onstat** option.	See Obtaining utility version information.
-x	Displays information about transactions	See onstat -x command: Print database server transaction information.
-X	Obtains precise information about the threads that are sharing and waiting for buffers	See onstat -X command: Print thread information.
-z	Sets the profile counts to 0	See onstat -z command: Clear statistics.
infile	Specifies a source file for the **onstat** command	This file must include a previously stored shared-memory segment that you created with the **onstat -o** command. For instructions on how to create the *infile* with **onstat -o**, see onstat -o command: Output shared memory contents to a file. For information about running **onstat** on the source file, see Running onstat Commands on a Shared Memory Dump File.

Interactive execution

To put the **onstat** utility in interactive mode, use the -i option. Interactive mode allows you to enter multiple options, one after the other, without exiting the program. For information on using interactive mode, see onstat -i command: Initiate interactive mode.

Continuous onstat command execution

Use the **onstat -r** option combined with other **onstat** options to cause the other options to execute repeatedly at a specified interval. For information, see onstat -r command: Repeatedly print selected statistics.

onstat command: Equivalent to the onstat -pu command

If you invoke **onstat** without any options, the command is interpreted as **onstat -pu** (the **-p** option and the **-u** option).

Syntax:

```
>>-onstat-----------------------------------------------------><
```


onstat - command: Print output header

All **onstat** output includes a header. The **onstat -** command displays only the output header and the value that is returned from this command indicates the database server mode.

Syntax:

```
>>-onstat-- - ------------------------------------------------><
```

The header takes the following form:

```
Version--Mode (Type)--(Checkpnt)--Up Uptime--Sh_mem Kbytes
```

Version
: Is the product name and version number

Mode
: Is the current operating mode.

(Type)
: If the database server uses High-Availability Data Replication, indicates whether the type is primary or secondary
If the database server is not involved in data replication, this field does not appear. If the type is primary, the value `P` appears. If the type is secondary, the value `S` appears.

(Checkpnt)
: Is a checkpoint flag
If it is set, the header might display two other fields after the mode if the timing is appropriate:

(CKPT REQ)
: Indicates that a user thread has requested a checkpoint

(CKPT INP)
: Indicates that a checkpoint is in progress. During the checkpoint, access is limited to read only. The database server cannot write or update data until the checkpoint ends

Uptime
: Indicates how long the database server has been running
If the system time is manually changed to the past and the server startup time is later than the current system time, the uptime is not available. In this situation, the header displays the text `Uptime Unavailable`.

Sh_mem
: Is the size of database server shared memory, expressed in kilobytes

The following is a sample header for the database server:

```
Informix Version 14.10.UC1--On-Line--Up 15:11:41--9216 Kbytes
```

From xC5 onwards, the header also includes a datetime stamp in the format: **<year>-<month>-<day> <hour>:<minute>:<second>**

```
IBM Informix Dynamic Server Version 14.10.FC5 -- On-Line -- Up 00:00:22 -- 54180 Kbytes
2020-12-17 19:15:15
```

If the database server is blocked, the **onstat** header output includes an extra line. For information about status codes in that line, see Monitor the database server status.

Return codes

When you exit the **onstat** utility, there are several useful codes that are displayed. See Return codes on exiting the onstat utility.

onstat -- command: Print onstat options and functions

Use the **onstat --** command to display a listing of all of the **onstat** options and their functions. You cannot combine this option with any other flag.

Syntax:

```
>>-onstat-- ----------------------------------------------><
```


Running onstat Commands on a Shared Memory Dump File

You can run **onstat** commands against a shared memory dump file. The shared memory dump file can be produced explicitly by using the **onstat -o** command. If the DUMPSHMEM configuration parameter is set to 1 or set to 2, the dump file is created automatically at the time of an assertion failure.

Syntax:

```
>>-onstat--options--infile-------------------------------------><
```

When using the command line, enter the source file as the final argument. The following example prints information about all threads for the shared memory dump contained in the file named `onstat.out`, rather than attempting to attach to the shared memory of a running server.

```
onstat -g ath onstat.out
```

For instructions on how to create the memory dump file with **onstat -o**, see onstat -o command: Output shared memory contents to a file.

Running onstat Commands on a Shared Memory Dump File Interactively

Use **onstat -i** (interactive mode) to run more than one **onstat** command against a dump file. Interactive mode can save time because the file is read only once. In command-line mode, each command reads the file.

The following example reads the shared memory dump file and enters interactive mode. Other **onstat** commands can be executed against the dump file in the normal interactive fashion.

```
onstat -i source_file
```

For information about interactive mode, see onstat -i command: Initiate interactive mode.

Running onstat Commands on a Shared Memory Dump File Created Without a Buffer Pool

Certain **onstat** commands have different output when you run them on a dump file created without the buffer pool (created with **onstat -o nobuffs** or with the DUMPSHMEM configuration parameter set to 2):

- If you run **onstat -B** on a dump file created without the buffer pool, the output will display `0` in the `memaddr`, `nslots`, and `pgflgs` columns.
- If you run **onstat -g seg** on a dump file created without the buffer pool, the output will show both the original and nobuffs resident segment size.
- If you run **onstat -P** on a shared-memory dump file that does not have the buffer pool, the output is:

  ```
  Nobuffs dumpfile -- this information is not available
  ```

Related reference:
DUMPSHMEM configuration parameter (UNIX)
onstat -g seg command: Print shared memory segment statistics

onstat -a command: Print overall status of the database server

Use the **onstat -a** command to display information about the status of the database server. This command does not display information about all of the onstat options, only about those onstat options used for initial troubleshooting.

Syntax:

```
>>-onstat-- -a-------------------------------------------------><
```

onstat -b command: Print buffer information for buffers in use

Use the **onstat -b** option to display information about the buffers that are currently in use, including the total number of resident pages in the buffer pool.

Syntax:

```
>>-onstat-- -b-------------------------------------------------><
```

The maximum number of buffers available is specified in the **buffers** field in the BUFFERPOOL configuration parameter in the ONCONFIG file.

The **onstat -b** command also provides summary information about the number of modified buffers, the total number of resident pages in the buffer pool, the total number of buffers available, the number of hash buckets available, and the size of the buffer in bytes (the page size).

```
123 modified, 23 resident, 2000 total, 2048 hash buckets, 2048 buffer size.
```

For information about displaying information about all buffers, use onstat -B command: Prints information about used buffers.

Example output

Following is sample output from the **onstat -b** command. For a description of the output, see onstat -B command: Prints information about used buffers.

Figure 1. **onstat -b** command output

```
Buffer pool page size: 4096

address          userthread flgs pagenum memaddr          nslots pgflgs xflgs owner waitlist
70000001097e9e8 0            c07 1:47841 7000000118e0000 10     1      0     0     0
700000010982188 0            807 1:47827 700000011939000 225    90     10    0     0
2011 modified, 50000 total, 65536 hash buckets, 4096 buffer size
```


onstat -B command: Prints information about used buffers

Use the **onstat -B** option to display information about buffers that are not on the free-list.

Syntax:

```
>>-onstat-- -B-------------------------------------------------><
```

Both **onstat -B** and **onstat -b** display the similar information, except that the **onstat -b** command only displays buffers that are currently being accessed by a user thread. The **onstat -B** command displays information for all the buffers that are not on the free-list.

For information about running the **onstat -B** command on a dump file created without the buffer pool, see Running onstat Commands on a Shared Memory Dump File.

Example output

Output description

Buffer pool page size
: the size of the buffer pool pages in bytes

address
: the address of the buffer header in the buffer table

userthread
: the address of the most recent user thread to access the buffer table. Many user threads might be reading the same buffer concurrently.

flgs
: Uses the following flag bits to describe the buffer:

  0x01
  : Modified data

  0x02
  : Data

  0x04
  : LRU

  0x08
  : Error

pagenum
: the physical page number on the disk

memaddr
: the buffer memory address

nslots
: the number of slot-table entries in the page
  This field indicates the number of rows (or portions of a row) that are stored on the page.

pgflgs
Uses the following values, alone or in combination, to describe the page type:

1
Data page
2
Tblspace page
4
Free-list page
8
Chunk free-list page
9
Remainder data page
b
Partition resident blobpage
c
Blobspace resident blobpage
d
Blob chunk free-list bit page
e
Blob chunk blob map page
10
B-tree node page
20
B-tree root-node page
40
B-tree branch-node page
80
B-tree leaf-node page
100
Logical-log page
200
Last page of logical log
400
Sync page of logical log
800
Physical log
1000
Reserved root page
2000
No physical log required
8000
B-tree leaf with default flags

xflgs
Uses the following flag bits to describe buffer access:

0x10
share lock
0x80
exclusive lock

owner
the user thread that set the **xflgs** buffer flag
waitlist
the address of the first user thread that is waiting for access to this buffer
For a complete list of all threads waiting for the buffer, refer to onstat -X command: Print thread information.

onstat -c command: Print ONCONFIG file contents

Use the **onstat -c** command to display the contents of the ONCONFIG file.

```
Syntax:

>>-onstat-- -c-------------------------------------------------><
```

The database server first checks if you have assigned a value to the environment variable **ONCONFIG**. You can use the **onstat -c** option with the database server in any mode, including offline.

UNIX Only:
On UNIX, if you have set **ONCONFIG**, **onstat -c** displays the contents of the **$INFORMIXDIR/etc/$ONCONFIG** file. If not, by default, **onstat -c** displays the contents of **$INFORMIXDIR/etc/onconfig**.

Windows Only:
On Windows, if you have set **ONCONFIG**, **onstat -c** displays the contents of the **%INFORMIXDIR%\etc\%ONCONFIG%** file. If not, by default, **onstat -c** displays the contents of **%INFORMIXDIR%\etc\onconfig**.

Related tasks:
Displaying the settings in the onconfig file
Related reference:
onstat -g cfg command: Print the current values of configuration parameters
Database server files

onstat -C command: Print B-tree scanner information

Use the **-C** command to display information about the B-tree scanner subsystem and each B-tree scanner thread.

```
Syntax:

>>-onstat-- -C--+-prof--+--------------------------------------><
               +-hot---+
               +-part--+
               +-clean-+
               +-range-+
               +-map---+
               +-alice-+
               '-all---'
```

The following options are available with the **onstat -C** command and can be combined:

prof
: Prints the profile information for the system and each B-tree scanner thread. This is the default option.

hot
: Prints the hot list index key in the order to be cleaned

part
: Prints all partitions with index statistics

clean
: Prints information about all the partitions that were cleaned or need to be cleaned

range
: Prints the savings in pages processed by using index range scanning

map
: Displays the current bitmaps for each index being cleaned by the alice cleaning method

alice
: Displays the efficiency of the alice cleaning method option

all
: Prints all **onstat -C** options

Example output using the prof option

Figure 1. **onstat -C** command output with the prof option

```
Btree Cleaner Info
BT scanner profile Information
==============================
Active Threads                              1
Global Commands                       2000000   Building hot list
Number of partition scans               11003
Main Block                        0xc000000003c9dc68
BTC Admin                         0xc0000000024bc208


BTS info             id   Prio    Partnum      Key     Cmd
0xc000000003c9dee8    0   High    0x00000000     0       40  Yield N
    Number of leaves pages scanned                          77
    Number of leaves with deleted items                      6
    Time spent cleaning (sec)                                0
    Number of index compresses                               0
    Number of deleted items                                113
    Number of index range scans                              0
    Number of index leaf scans                               0
    Number of index alice scans                              2
```

Output description using the prof option

Id
: BTSCANNER ID

Prio
: Current priority of BTSCANNER

Partnum
: The partition number for the index this thread is currently working on

Cmd
: Command this thread is processing currently

Example output using the hot option

Figure 2. **onstat -C** command output with the hot option

```
Btree Cleaner Info

Index Hot List
==============
    Current Item            5     List Created          15:29:47
    List Size               4     List expires in              0 sec
    Hit Threshold         500     Range Scan Threshold        -1


Partnum         Key             Hits
0x00100191        1               14 *
0x00A00022        1               13 *
0x00100191        2                8 *
0x00100150        2                7 *
```

Output description using the hot option

Partnum
: The partition number for an index

Key
: Index Key

Hits
: The current value of the Hit counter

  *
  : Indicates that this partition has been cleaned during this hot list duration

Example output using the part option

Figure 3. **onstat -C** command output with the part option

```
Btree Cleaner Info


Index Statistics
================
 Partnum  Key      Positions   Compress      Split
0x00100002  1            146          0          0
0x00100004  1              4          0          0
0x00100004  2             13          0          0
0x00100005  1              1          0          0
0x00100005  2              0          0          0
0x00100006  1              1          0          0
0x00100006  2              0          0          0
0x00100007  2              1          0          0
0x00100008  2              1          0          0
0x0010000a  1              0          0          0
0x0010000e  3              1          0          0
0x00100011  1              1          0          0
0x00100013  2              2          0          0
```

Output description using the part option

Partnum
: The partition number for an index

Key
: Index Key

Positions
: Number of times index has been read

Compress
: Number of pages which have been compressed

Split
: Number of splits that have occurred

  C
  : Indicates partition is busy being cleaned

  N
  : Index partition no longer eligible for cleaning

Example output using the clean option

Figure 4. **onstat -C** command output with the clean option

```
Btree Cleaner Info

Index Cleaned Statistics
========================
 Partnum  Key      Dirty Hits  Clean Time  Pg Examined  Items Del  Pages/Sec
0x00100013  2               2           0            0          0       0.00
0x0010008b  3               1           0            0          0       0.00
0x001000c7  1               2           0            0          0       0.00
0x00100150  2               7           0            0          0       0.00
0x0010016f  2               2           0            0          0       0.00
0x00100191  1              14           0            0          0       0.00
```

```
0x00100191  2              8           0           0           0        0.00
0x00a00011  2              6           0           0           0        0.00
0x00a00013  1              0           0          24           0       24.00
0x00a00019  1              0           0         470         225      470.00
0x00a00022  1             13           0           0           0        0.00
0x00a00022  2              5           0           0           0        0.00
```

Output description using the clean option

Partnum
: The partition number for an index

Key
: Index Key

Dirty Hits
: Number of times a dirty page has been scanned

Clean Time
: Total time spent, in seconds

Pg Examined
: Number of pages examined by btscanner thread

Items Del
: Number of items removed form this index

Pages/Sec
: Number of pages examined per second

C
: Indicates partition is busy being cleaned

N
: index partition is no longer eligible for cleaning

Example Output

Figure 5. **onstat -C** range

```
Btree Cleaner Info

Cleaning Range Statistics
=========================
 Partnum  Key             Low          High          Size         Saving
0x001001bc  2              36            69            96       65.6 %
0x001001be  1              16            20            48       91.7 %
0x001001cd  1               8            21            32       59.4 %
0x001001cd  2              24            25            32       96.9 %
```

Output Description

Partnum
: The partition number

Key
: Index Key

Low
: Low boundary for range scan

High
: High boundary for index scan

Size
: Size of index in pages

Saving
: Percentage of time saved versus a full scan

C
: Indicates partition is busy being cleaned

N
: Index partition is no longer eligible for cleaning

Example Output

Figure 6. **onstat -C** map

```
Btree Cleaner Info

ALICE Bitmap of Deleted Index Items
===================================
Partnum   Key         Map
0x00100013  2 0000: 80000000 00000000
0x0010008b  3 0000: 80000000 00000000
0x001000c7  1 0000: 80000000 00000000
0x00100150  2 0000: 80000000 00000000
0x0010016f  2 0000: 80000000 00000000
0x00100191  1 0000: 80000000 00000000
0x00100191  2 0000: 80000000 00000000
0x00a00011  2 0000: 80000000 00000000
0x00a00013  1 0000: 00000000 00000000
0x00a00019  1 0000: 00000000 00000000
```

```
0x00a00022  1 0000: 80000000 00000000
0x00a00022  2 0000: 80000000 00000000
```

Output Description

Partnum
: The partition number

Key
: Index Key

Map
: Alice bitmap

Example Output

Figure 7. **onstat -C** alice

```
Btree Cleaner Info

ALICE Cleaning Statistics
=========================

System ALICE Info: Mode =    6, Eff =    30 %, Adj =     5

Partnum    Mode BM_Sz  Used_Pg  Examined  Dirty_Pg  # I/O  Found    Eff    Adj
0x00100013    6    64       97         0         0      0      0   0.0 %    0
0x0010008b    6    64        5         0         0      0      0   0.0 %    0
0x001000c7    6    64        2         0         0      0      0   0.0 %    0
0x00100150    6    64       91         0         0      0      0   0.0 %    0
0x0010016f    6    64       91         0         0      0      0   0.0 %    0
0x00100191    6    64       26         0         0      0      0   0.0 %    0
0x00100191    6    64       26         0         0      0      0   0.0 %    0
0x001001bc    0     0       91         0         0      0      0   0.0 %    0
0x001001cd    0     0       26         0         0      0      0   0.0 %    0
0x001001cd    0     0       26         0         0      0      0   0.0 %    0
0x00a00011    6    64       91         0         0      0      0   0.0 %    0
0x00a00013    6    64       25        24         3      3      1  33.3 %    1
0x00a00019    6    64      470       470         3      3      2  66.7 %    1
0x00a00022    6    64       26         0         0      0      0   0.0 %    0
0x00a00022    6    64       26         0         0      0      0   0.0 %    0
```

Output Description

Partnum
: The partition number for an index

Mode
: The alice mode for the current partition

BM_Sz
: The size allocated for the bitmap

Used_Pg
: The size of the index in pages (used)

Dirty_Pg
: Number of dirty pages

I/O
: Number of pages read

Found
: Number of dirty pages found in reads

Eff
: How efficient was the bitmap

Adj
: Number of times the alice efficiency level for the partition was insufficient and was adjusted

onstat -d command: Print chunk information

Use the **onstat -d** command to show information about chunks in each storage space.

Syntax:

```
>>-onstat-- -d--+--------+---------------------------------------><
               '-update-'
```

The **update** option updates shared memory to obtain accurate counts of free pages.

Using onstat -d with sbspaces

For information about using **onstat -d** to determine the size of sbspaces, user-data areas, and metadata areas, see Monitor sbspaces.

Using onstat -d with blobspaces

If you run the **onstat -d** command on a server that has blobspace chunks, the database server displays the following message:

```
NOTE: For BLOB chunks, the number of free pages shown is out of date.
 Run 'onstat -d update' for current stats.
```

To obtain the current statistics for blobspace chunks, run the **onstat -d update** command. The **onstat** utility updates shared memory with an accurate count of free pages for each blobspace chunk. The database server shows the following message:

```
Waiting for server to update BLOB chunk statistics ...
```

Example output

Figure 1. onstat -d command output

```
BM Informix Dynamic Server Version 14.10.F       -- On-Line -- Up 00:01:27 -- 133540 Kbytes

Dbspaces
address          number   flags      fchunk   nchunks  pgsize   flags    owner    name
4484a028         1        0x10020001 1        1        2048     N  BAE   informix rootdbs
45ed5b30         2        0x20001    2        1        2048     N  BA    informix space1
 2 active, 2047 maximum

Chunks
address          chunk/dbs     offset     size       free       bpages     flags pathname
4484a268         1      1      0          100000     65632                 PO-B-- /work3/AB/rootchunk
45f1f028         2      2      0          5000       4947                  PO-B-- /work3/AB/chunk5
 2 active, 32766 maximum

NOTE: The values in the "size" and "free" columns for DBspace chunks are
      displayed in terms of "pgsize" of the DBspace to which they belong.


Expanded chunk capacity mode: always
```

Output description for dbspaces

The first section of the output describes the storage spaces:

address
: Is the address of the storage space in the shared-memory space table

number
: Is the unique ID number of the storage space that is assigned at when it is created

flags
: Uses hexadecimal values to describe each storage space. The individual flag values can be summed to show cumulative properties of the dbspace. The following table describes each hexadecimal value:

Table 1. Descriptions for each hexadecimal value

Flag Value	Description
0x0001	Mirror is allowed and dbspace is unmirrored.
0x0002	Mirror is allowed and dbspace is mirrored.
0x0004	The dbspace contains disabled mirror chunks.
0x0008	Newly mirrored
0x0010	Blobspace
0x0200	Space is being recovered.
0x0400	Space is physically recovered.
0x0800	Logical log is being recovered.
0x2000	Temporary dbspace
0x4000	Blobspace is being backed up.
0x8000	Sbspace
0x10000	Physical or logical log changed.
0x20000	Dbspace or chunk tables changed.
0x040000	Blobspace contains large chunks.
0x080000	Chunk in this dbspace was renamed.
0x00100000	Temporary dbspace that is used by only by shared disk secondary server. It is one of the dbspaces listed in the SDS_TEMPDBS configuration parameter on the SD secondary server.
0x00200000	Temporary dbspace for the SD secondary server. Listed in the DBSPACETEMP configuration parameter on the shared disk secondary server.
0x00400000	The dbspace was externally backed up.
0x00800000	Dbspace is being defragmented.

Flag Value	Description
0x01000000	Plogspace
0x10000000	The space is encrypted.

fchunk
: The ID number of the first chunk

nchunks
: The number of chunks in the storage space

pgsize
: The size of the dbspace pages in bytes

flags
: Uses the following letter codes to describe each storage space:

Position 1:

Flag	Description
M	Mirrored
N	Not mirrored

Position 2:

Flag	Description
X	Newly mirrored
P	Physically recovered, waiting for logical recovery
L	Being logically recovered
R	Being recovered
D	Down

Position 3:

Flag	Description
B	Blobspace
P	Plogspace
S	Sbspace
T	Temporary dbspace
U	Temporary sbspace
W	Temporary dbspace on primary server (This flag is shown on SD secondary servers only.)

Position 4:

Flag	Description
B	The dbspace can have large chunks that are greater than 2 GB.

Position 5:

Flag	Description
A	The dbspace is auto-expandable because the SP_AUTOEXPAND configuration parameter is enabled and the dbspace is configured with a create size or extend size that is not zero.

Position 6:

Flag	Description
E	The storage space is encrypted.

owner
: The owner of the storage space

name
: The name of the storage space

In the line immediately following the storage-space list, **active** refers to the current number of storage spaces in the database server instance, including the root dbspace and **maximum** refers to total allowable spaces for this database server instance.

Output description - Chunks

The second section of the **onstat -d** command output describes the chunks:

address
: The address of the chunk

chk/dbs
: The chunk number and the associated space number

offset
: The offset into the file or raw device in base page size

size
: The size of the chunk in terms of the page size of the dbspace to which it belongs.

free
: The number of unallocated pages in the chunk in units of the page size of the associated dbspace. A value of `0` indicates that all the space in the chunk is allocated to tables, but does not indicate how much space is free inside the tables. For example, suppose you create a dbspace with one chunk of 200 MB and create one table with an extent size of 200 MB. The value of the **free** field is `0`, indicating that the chunk has no free space, however, the new empty table has 200 MB of free space.
  For a blobspace, a tilde indicates an approximate number of unallocated blobpages.

For an sbspace, indicates the number of unallocated pages of user data space and total user data space.

Note: The "onstat -d" output adds footnotes for each of the down chunks that are empty, and a special foot note for the first chunk of a down space that is empty.
Example Output for onstat -d with down space (due to first chunk out of 4), where 2 of the 4 chunks are empty and can be dropped:

```
Chunks
address          chunk/dbs     offset     size       free       bpages     flags pathname
44be2268         1      1      0          150000     83059                 PO-B-- /spaces/rootchunk
45c2b028         2      2      0          512        0                     PD-BE- /spaces/dbspace2_p_1
45c2c028         3      2      0          500        *                     PD-BE- /spaces/dbspace2_p_2
45c2d028         4      2      0          500        *                     PD-BE- /spaces/dbspace2_p_3
45c2e028         5      2      0          5000       0                     PD-BE- /spaces/dbspace2_p_4
5 active, 32766 maximum

NOTE: The values in the "size" and "free" columns for DBspace chunks are
      displayed in terms of "pgsize" of the DBspace to which they belong.

*  Down chunk is empty, and may be safely dropped.
```

Example Output for onstat -d when all chunks in down space are empty and can be dropped:

```
Chunks
address          chunk/dbs     offset     size       free       bpages     flags pathname
44be2268         1      1      0          150000     83059                 PO-B-- /spaces/rootchunk
45c2b028         2      2      0          512        **                    PD-BE- /spaces/dbspace2_p_1
45c2c028         3      2      0          500        *                     PD-BE- /spaces/dbspace2_p_2
45c2d028         4      2      0          500        *                     PD-BE- /spaces/dbspace2_p_3
45c2e028         5      2      0          5000       *                     PD-BE- /spaces/dbspace2_p_4
5 active, 32766 maximum

NOTE: The values in the "size" and "free" columns for DBspace chunks are
      displayed in terms of "pgsize" of the DBspace to which they belong.

*  Down chunk is empty, and may be safely dropped.
** Down space is empty, and may be safely dropped.
```

bpages

Is the size of the chunk in blobpages
Blobpages can be larger than disk pages; therefore, the **bpages** value can be less than the **size** value.

For an sbspace, is the size of the chunk in sbpages.

flags

Provides the chunk status information as follows:
Position 1:

Flag	Description
P	Primary
M	Mirror

Position 2:

Flag	Description
N	Renamed and either Down or Inconsistent
O	Online
D	Down
X	Newly mirrored
I	Inconsistent

Position 3:

Flag	Description
-	Dbspace
B	Blobspace
S	Sbspace

Position 4:

Flag	Description
B	The dbspace can have large chunks that are greater than 2 GB.

Position 5:

Flag	Description
E	Identifies the chunk as extendable
-	Identifies the chunk as not extendable

Position 6:

Flag	Description
-	The direct I/O or concurrent I/O option is not enabled for this cooked file chunk
C	On AIX®, the concurrent I/O option is enabled for this cooked file chunk
D	The direct I/O option is enabled for this cooked file chunk

pathname

The path name of the physical device

In the line immediately following the chunk list, **active** shows the number of active chunks (including the root chunk) and **maximum** shows the total number of chunks.

For information about page reads and page writes, run the **onstat -D** command.

Related reference:
DBSPACETEMP configuration parameter
onstat -D command: Print page-read and page-write information
DIRECT_IO configuration parameter (UNIX)
MIRROR configuration parameter
modify chunk extend argument: Extend the size of a chunk (SQL administration API)
DISK_ENCRYPTION configuration parameter

onstat -D command: Print page-read and page-write information

Use the **onstat -D** command to display page-read and page-write information for the first 50 chunks in each space.

Syntax:

```
>>-onstat-- -D-----------------------------------------------><
```

Example output

Figure 1. **onstat -D** command output

```
Dbspaces
address  number   flags      fchunk   nchunks pgsize  flags    owner    name
a40d7d8  1        0x1        1        1       2048    N        informix rootdbs
 1 active, 2047 maximum

Chunks
address  chunk/dbs  offset     page Rd  page Wr  pathname
a40d928  1      1   0          0        0        /work/11.1/dbspaces/stardbs3
 1 active, 2047 maximum

Expanded chunk capacity mode: disabled
```

Output description

The output of **onstat -D** is almost identical to the output of **onstat -d**. The following columns are unique to **onstat -D**. For information on the other output columns see onstat -d command: Print chunk information.

page Rd
Is the number of pages read

page Wr
Is the number of pages written

Related reference:
onstat -d command: Print chunk information

onstat -f command: Print dbspace information affected by dataskip

Use the **-f** command to list the dbspaces that the dataskip feature currently affects.

Syntax:

```
>>-onstat-- -f-----------------------------------------------><
```

The **-f** option lists both the dbspaces that were set with the DATASKIP configuration parameter and the **-f** option of **onspaces**. When you execute **onstat -f**, the database server displays one of the following three outputs:

- Dataskip is OFF for all dbspaces.
- Dataskip is ON for all dbspaces.
- Dataskip is ON for the following dbspaces:

  ```
  dbspace1 dbspace2...
  ```

Related reference:
DATASKIP Configuration Parameter

onstat -F command: Print counts

Use the **onstat -F** command to display a count for each type of write that flushes pages to disk.

Syntax:

```
>>-onstat-- -F-------------------------------------------------><
```

Example output

Figure 1. **onstat -F** command output

```
Fg Writes     LRU Writes    Chunk Writes
0             330           7631

address  flusher  state  data  # LRU  Chunk  Wakeups  Idle Time
c7c8850  0        I      0     9      29     16116    16093.557
      states: Exit Idle Chunk Lru
```

Output description

You can interpret output from this option as follows:

Fg Writes
: Is the number of times that a foreground write occurred

LRU *Writes*
: Is the number of times that an LRU write occurred

Chunk Writes
: Is the number of times that a chunk write occurred

address
: Is the address of the user structure assigned to this page-cleaner thread

flusher
: Is the page-cleaner number

state
: Uses the following codes to indicate the current page-cleaner activity:

  C
  : Chunk write

  E
  : Exit

  I
  : Cleaner is idle

  L
  : LRU queue

  The exit code indicates either that the database server is performing a shutdown or that a page cleaner did not return from its write in a specific amount of time. When an operation fails to complete within the allotted time, this situation is known as a time-out condition. The database server does not know what happened to the cleaner, so it is marked as `exit`. In either case, the cleaner thread eventually exits.

data
: Provides additional information in concert with the **state** field
  If **state** is `C`, **data** is the chunk number to which the page cleaner is writing buffers. If **state** is `L`, **data** is the LRU queue from which the page cleaner is writing. The **data** value is displayed as a decimal, followed by an equal sign, and repeated as a hexadecimal.

#LRU
: Corresponds to the **onstat -g ath** thread ID output

Chunk
: Number of chunks cleaned

Wakeups
: Number of times the flusher thread was awoken

Idle Time
: Time in seconds the flusher thread has been idle

Related reference:
CLEANERS configuration parameter

onstat -g monitoring options

The options that you can use with **onstat -g** command are used for support and debugging only. You can include only one of these options in the **onstat -g** command.

The **onstat -g imc** command prints information about Informix® MaxConnect instances that are connected to the database server. If Informix MaxConnect is not connected to the database server, this command displays No MaxConnect servers are connected.

The **onstat -g nta** command prints combined network statistics from the **-g ntd**, **-g ntm**, **-g ntt**, and **-g ntu** commands. If Informix MaxConnect is installed, this command prints statistics that you can use to tune Informix MaxConnect performance.

- onstat -g act command: Print active threads
  Use the **onstat -g act** command to display information about the active threads.
- onstat -g afr command: Print allocated memory fragments
  Use the **onstat -g afr** command to display information about the allocated memory fragments for a specified session or shared-memory pool. Each session is allocated a pool of shared memory.
- onstat -g all command: Print diagnostic information
  Use the **onstat -g all** command to gather diagnostic information if advised to do so by Support. For normal administrative purposes, use the **onstat -g** command with individual options.
- onstat -g aqt command: Print data mart and accelerated query table information
  Use the **onstat -g aqt** command to display information about the data marts and the associated accelerated query tables (AQTs).
- onstat -g arc command: Print archive status
  Use the **onstat -g arc** command to display information about the last committed archive for each dbspace and also information about any current ongoing archives.
- onstat -g ath command: Print information about all threads
  Use the **onstat -g ath** command to display information about all threads.
- onstat -g bth and -g BTH: Print blocked and waiting threads
  Use the **onstat -g bth** command to display the dependencies between blocking and waiting threads. Use the **onstat -g BTH** command to display session and stack information for the blocking threads.
- onstat -g buf command: Print buffer pool profile information
  Use the **onstat -g buf** command to show profile information for each buffer pool.
- onstat -g cac command: Print information about caches
  Use the **onstat -g cac** command to see summary and detailed information about all caches or about a single cache.
- onstat -g ckp command: Print checkpoint history and configuration recommendations
  Use the **onstat -g ckp** command to print checkpoint history and show configuration recommendations if a suboptimal configuration is detected.
- onstat -g cfg command: Print the current values of configuration parameters
  Use the **onstat -g cfg** command to print a list of configuration parameters with their current values. You can use more command options to print more information about the configuration parameters.
- onstat -g cluster command: Print high-availability cluster information
  Use the **onstat -g cluster** command to display information about the servers in a high-availability cluster environment.
- onstat -g cmsm command: Print Connection Manager information
  Use the **onstat -g cmsm** command to display information about a specific Connection Manager, or all of the Connection Managers that are attached to the database server the command is run on.
- onstat -g con command: Print condition and thread information
  Use the **onstat -g con** command to display information about conditions and the threads that are waiting for the conditions.
- onstat -g cpu: Print runtime statistics
  Use the **onstat -g cpu** command to display information about runtime statistics for each thread that is running in the server.
- onstat -g dbc command: Print dbScheduler and dbWorker thread statistics
  Use the **onstat -g dbc** command to display statistics about the Scheduler tasks that are currently running, which are handled by dbWorker threads, or scheduled to be run, which are handled by the dbScheduler thread.
- onstat -g defragment command: Print defragment partition extents
  Use the **onstat -g defragment** command to display information about the active requests to defragment partition extents.
- onstat -g dic command: Print table information
  Use the **onstat -g dic** command to display a line of information about each table that is cached in the shared-memory dictionary. If you specify a table name, this command prints internal SQL information about that particular table.
- onstat -g dis command: Print database server information
  Use the **onstat -g dis** command to display a list of database servers, the status of each server, and information about each server, including the location of the **INFORMIXDIR** directory, **sqlhosts** file, and ONCONFIG file. You can use this command in any database server mode, including offline.
- onstat -g dll command: Print dynamic link library file list
  Use the **onstat -g dll** command to display a list of and the status of dynamic link library (DLL) files that were loaded.
- onstat -g dmp command: Print raw memory
  Use the **onstat -g dmp** command to display information about raw memory at a given address for a number of given bytes.
- onstat -g dri command: Print high-availability data replication information
  Use the **onstat -g dri** command, either alone or with the **ckpt** or **que** options, to print information about high-availability data replication statistics on the current server.
- onstat -g dsc command: Print distribution cache information
  Use the **onstat -g dsc** command to display information about the distribution cache.
- onstat -g dsk command: Print the progress of the currently running compression operation
  Use the **onstat -g dsk** command to print information that shows the progress of currently running compression operations, such as compress, repack, and shrink.
- onstat -g env command: Print environment variable values
  Use the **onstat -g env** command to display the values of the environment variables that the database server currently uses.
- onstat -g ffr command: Print free fragments
  Use the **onstat -g ffr** command to display information about the free memory fragments for a specified session or shared-memory pool.
- onstat -g glo command: Print global multithreading information
  Use the **onstat -g glo** command to display global information about multithreading, information about each virtual processor that is running, and cumulative statistics for each virtual-processor class. This information includes CPU use information about the virtual processors, the total number of sessions, and other multithreading global counters.
- onstat -g his command: Print SQL trace information
  Use the **onstat -g his** command to display SQL trace information from the collection of **syssqltrace** tables (syssqltrace, syssqltrace_info, syssqltrace_hvar and syssqltrace_itr) in the **sysmaster** database.
- onstat -g ioa command: Print combined onstat -g information
  Use the **onstat -g ioa** command to display combined information from the **onstat -g iob**, **onstat -g iof**, **onstat -g ioq**, and **onstat -g iov** commands.
- onstat -g iob command: Print big buffer use summary
  Use the **onstat -g iob** command to display a summary of big buffer use.
- onstat -g iof command: Print asynchronous I/O statistics
  Use the **onstat -g iof** command to display the asynchronous I/O statistics by chunk or file.
- onstat -g iog command: Print AIO global information
  Use the **onstat -g iog** command to display global information about AIO.

- onstat -g ioq command: Print I/O queue information
  Use the **onstat -g ioq** command to display statistics about the number and types of operations performed by I/O queues.
- onstat -g ipl command: Print index page logging status information
  Use the **onstat -g ipl** command to display information about the status of index page logging.
- onstat -g iov command: Print AIO VP statistics
  Use the **onstat -g iov** command to display asynchronous I/O statistics for each virtual processor.
- onstat -g lap command: Print light appends status information
  Use the **onstat -g lap** command to display information about the status of light appends occurring in the system.
- onstat -g laq command: Print log apply queues
  Use the **onstat -g laq** command to print information about log recovery apply queues.
- onstat -g lmm command: Print low memory management information
  Use the **onstat -g lmm** command to display information about automatic low memory management settings and recent activity.
- onstat -g lmx command: Print all locked mutexes
  Use the **onstat -g lmx** command to display information about all locked mutexes.
- onstat -g lsc command: Print active light scan status (deprecated)
  The **onstat -g lsc** command has been superseded by the **onstat -g scn** command.
- onstat -g mem command: Print pool memory statistics
  Use the **onstat -g mem** command to display the memory statistics for a pool.
- onstat -g mgm command: Print MGM resource information
  Use the **onstat -g mgm** command to show resource information about Memory Grant Manager (MGM).
- onstat -g nbm command: Print a block bit map
  Use the **onstat -g nbm** command to display the block bit map for the nonresident segments.
- onstat -g nsc command: Print current shared memory connection information
  Use the **onstat -g nsc** command to display information about shared memory connections either for all of the current connections or for a specified connection ID.
- onstat -g nsd command: Print poll threads shared-memory data
  Use the **onstat -g nsd** command to display information about shared-memory data for poll threads.
- onstat -g nss command: Print shared memory network connections status
  Use the **onstat -g nss** *sessionid* command to display information about the status of the shared memory network connections.
- onstat -g ntd command: Print network statistics
  Use the **onstat -g ntd** command to display network statistics by service.
- onstat -g ntm command: Print network mail statistics
  Use the **onstat -g ntm** command to display statistics about network mail.
- onstat -g ntt command: Print network user times
  Use the **onstat -g ntt** command to display information about network user times.
- onstat -g ntu command: Print network user statistics
  Use the **onstat -g ntu** command to display information about network user statistics.
- onstat -g opn command: Print open partitions
  Use the **onstat -g opn** command to display a list of the partitions (tables and indexes), by thread ID, that are currently open in the system.
- onstat -g osi: Print operating system information
  Use the **onstat -g osi** command to display information on your operating system resources and parameters, including shared memory and semaphore parameters, the amount of memory currently configured on the computer, and the amount of memory that is unused.
- onstat -g pd command: Print push data session-related information
  Use the **onstat -g pd** command to display information about the push data session.
- onstat -g pd event command: Print push data event-related information
  Use the **onstat -g pd** event command to display information about the push data event.
- onstat -g pfsc command: Print partition free space cache information
  Use the **onstat -g pfsc** command to display information about all partition free space caches.
- onstat -g pos command: Print file values
  Use the **onstat -g pos** command to display the values in the **$INFORMIXDIR/etc/.infos.DBSERVERNAME** file.
- onstat -g ppd command: Print partition compression dictionary information
  Use the **onstat –g ppd** command to display information about the active compression dictionaries that were created for compressed tables and table fragments or compressed B-tree indexes. You can choose to print information for a particular numbered partition or for all open partitions.
- onstat -g ppf command: Print partition profiles
  Use the **onstat -g ppf *partition_number*** command to display the partition profile for the specified partition number.
- onstat -g pqs command: Print operators for all SQL queries
  Use the **onstat –g pqs** command to display information about the operators used in all of the SQL queries that are currently running.
- onstat -g prc command: Print sessions using UDR or SPL routines
  Use the **onstat -g prc** command to display the number of sessions that are currently using the UDR or SPL routine.
- onstat -g proxy command: Print proxy distributor information
  Use the **onstat -g proxy** command to display information about proxy distributors. The output of the **onstat -g proxy** command differs slightly depending on whether the command is run on a primary server or on a secondary server.
- onstat -g qst command: Print wait options for mutex and condition queues
  Use the **onstat -g qst** command to display the wait statistics for mutex queues and condition queues (queues of waiters for a mutex or a condition).
- onstat -g rah command: Print read-ahead request statistics
  Use the **onstat -g rah** command to display information about read-ahead requests.
- onstat -g rbm command: Print a block map of shared memory
  Use the **onstat -g rbm** command to display a hexadecimal bitmap of the free and used blocks within the resident segment of shared memory.
- onstat -g rea command: Print ready threads
  Use the **onstat -g rea** command to display information about the virtual processor threads whose current status is ready.
- onstat -g rss command: Print RS secondary server information
  Use the **onstat -g rss** commands to display information about remote standalone secondary servers.
- onstat -g rwm command: Print read and write mutexes
  Use the **onstat -g rwm** command to display information about read, write, and waiting mutex threads, and to list the addresses of the tickets that these threads have acquired.
- onstat -g sch command: Print VP information
  Use the **onstat -g sch** command to display information about thread migration and the number of semaphore operations, spins, and busy waits for each virtual processor.
- onstat -g scn command: Print scan information
  Use the **onstat -g scn** command to display the status of a current scan and information about the scan.

- onstat -g sds command: Print SD secondary server information
  Use the **onstat -g sds** command to display information about shared-disk secondary servers.
- onstat -g seg command: Print shared memory segment statistics
  Use the **onstat -g seg** command to show the statistics for shared memory segments.
- onstat -g ses command: Print session-related information
  Use the **onstat -g ses** command to display information about the session.
- onstat -g shard command: Print information about the shard definition
  Use the **onstat -g shard** command to display information about the sharding definition.
- onstat -g sle command: Print all sleeping threads
  Use the **onstat -g sle** command to print all sleeping threads.
- onstat -g smb command: Print sbspaces information
  Use the **onstat -g smb** command to display detailed information about sbspaces.
- onstat -g smx command: Print multiplexer group information
  Use the **onstat -g smx** command to display information about the server multiplexer group for servers using SMX.
- onstat -g spi command: Print spin locks with long spins
  Use the **onstat -g spi** command to display information about spin locks with long spins.
- onstat -g sql command: Print SQL-related session information
  Use the **onstat -g sql** command to display SQL-related information about a session.
- onstat -g spf: Print prepared statement profiles
  Use the **onstat -g spf** command to display current statistics about SQL queries.
- onstat -g src command: Patterns in shared memory
  Use the **onstat -g src** command to search for patterns in shared memory.
- onstat -g ssc command: Print SQL statement occurrences
  Use the **onstat -g ssc** command to monitor the number of times that the database server reads the SQL statement in the cache.
- onstat -g stk command: Print thread stack
  Use the **onstat -g stk** *tid* command to display the stack of the thread specified by thread ID.
- onstat -g stm command: Print SQL statement memory usage
  Use the **onstat -g stm** command to display the memory that each prepared SQL statement uses.
- onstat -g stq command: Print queue information
  Use the **onstat -g stq** command to display information about the queue.
- onstat -g sts command: Print stack usage for each thread
  Use the **onstat -g sts** command to display information about the maximum and current stack use for each thread.
- onstat -g sym command: Print symbol table information for the oninit utility
  Use the **onstat -g sym** command to display symbol table information for the **oninit** utility.
- onstat -g top command: Print top consumers of resources
  Use the **onstat -g top** command to display information about top consumers of various resources such as CPU time, I/O operations, and memory growth.
- onstat -g tpf command: Print thread profiles
  Use the **onstat -g tpf** command to display thread profiles.
- onstat -g ufr command: Print memory pool fragments
  Use the **onstat -g ufr** command to display a list of the fragments that are currently in use in the specified memory pool.
- onstat -g vpcache command: Print CPU virtual processor and tenant virtual processor private memory cache statistics
  Run the **onstat -g vpcache** command to display statistics about CPU virtual processor and tenant virtual processor private memory caches.
- onstat -g wai command: Print wait queue thread list
  Use the **onstat -g wai** command to display a list of the threads in the system that are currently in the wait queue and not currently executing. The output is sorted by thread ID.
- onstat -g wmx command: Print all mutexes with waiters
  Use the **onstat -g wmx** command to display all of the mutexes with waiters.
- onstat -g wst command: Print wait statistics for threads
  Use the **onstat -g wst** command to show the wait statistics for the threads within the system.

Related information:
onstat -g commands for Enterprise Replication

onstat -g act command: Print active threads

Use the **onstat -g act** command to display information about the active threads.

Syntax:

```
>>-onstat-- -g--act-----------------------------------------><
```

Following is sample output from the **onstat -g act** command. For a description of the output, see onstat -g ath command: Print information about all threads.

Example output

Figure 1. **onstat -g act** command output

```
Running threads:
tid  tcb         rstcb  prty  status    vp-class     name
2    b3132d8     0      1     running   *2adm        adminthd
40   c5384d0     0      1     running   *1cpu        tlitcppoll
```

onstat -g afr command: Print allocated memory fragments

Use the **onstat -g afr** command to display information about the allocated memory fragments for a specified session or shared-memory pool. Each session is allocated a pool of shared memory.

Syntax:

```
>>-onstat-- -g--afr--+-pool_name----+-------------------------><
                     +-sessionid----+
                     '-pool_address-'
```

This command requires an additional argument to specify either a pool name, a session ID, or a pool address. Each session is allocated a memory pool with the same name as the session ID.

The *pool_name* is the name of the shared-memory pool. Run the **onstat -g mem** command to identify the pool name.

The *sessionid* is the session ID. Run the **onstat -g ses** command to identify the session ID.

The *pool_address* is the address of the shared-memory pool. Run the **onstat -g mem** command or the **onstat -g ses** command to identify the pool address.

Example output

Figure 1. **onstat -g afr** command output

```
Allocations for pool name global:
addr             size       memid          fileid   location
4b231000         3288       overhead       306      mtshpool.c:617
4b231cd8         72         mcbmsg         1637     rldmsg.c:92
4b231d20         160        mcbmsg         1637     rldmsg.c:92
4b231dc0         64         osenv          2909     osenv.c:1164
4b231e00         64         osenv          2909     osenv.c:1971
4b231e40         64         osenv          2909     osenv.c:1164
4b231e80         64         osenv          2909     osenv.c:1971
```

Output description

addr (hexadecimal)
: Memory address of the pool fragment.

size (decimal)
: Size, in bytes, of the pool fragment.

memid (string)
: Memory ID of the pool fragment.

fileid (decimal)
: Internal use only. Code file identifier for the allocation.

location (string)
: Internal use only. Line number in the code for the allocation.

onstat -g all command: Print diagnostic information

Use the **onstat -g all** command to gather diagnostic information if advised to do so by Support. For normal administrative purposes, use the **onstat -g** command with individual options.

Syntax:

```
>>-onstat-- -g--all-------------------------------------------><
```


onstat -g aqt command: Print data mart and accelerated query table information

Use the **onstat -g aqt** command to display information about the data marts and the associated accelerated query tables (AQTs).

Syntax:

```
>>-onstat-- -g--aqt--+----------+------------------------------><
                    '-aqt_name-'
```

Example output

Figure 1. **onstat -g aqt** command output

```
AQT Dictionary Cache for database school:

mart: school
accelerator: DWAFINAL
last load: 2011/07/29 07:00:39

AQT name                                  FactTab #tab #matched     address
------------------------------------------------------------------------------
aqt4d11b552-7d41-4b0c-824b-7714b6cb580a    103    1      328         0x4d187e08
aqt61498fab-3617-4c8c-ab40-fd8af4253998    103    2       42         0x4d84a448
aqtbc2da77c-bca8-4ce7-9191-8180a860da34    103    2      768         0x4d187f60
aqt88757e9d-81ee-43b4-87b2-0bf48c98fa55    103    3       15         0x4d84a190
aqta786d0dc-8e95-4de0-a1bd-773aa03a52db    103    3     1475        0x4d84a650
aqt8dd61c80-2c1c-4f0e-8f0c-91babe789f41    103    4      632         0x4d84a908

mart: school2
accelerator: DWAFINAL
last load: 2011/07/29 07:01:04

AQT name                                  FactTab #tab #matched     address
------------------------------------------------------------------------------
aqt56d5aea7-32f4-44e6-8d98-02a7af37630f    103    1      845         0x4d84ac70
aqt03ec4c20-7ba8-4c3a-ae56-4134b005269d    103    2       27         0x4d95c298
aqt4ae7c2fd-5b94-423d-bc49-9ca3f5f38799    103    2     3912         0x4d84adc8
aqt5ed69a75-15e3-45cc-9892-4f5386257895    103    3       83         0x4d95c4a0
aqtdf314aa6-177d-4443-9f6d-f14ba766995a    103    3       37         0x4d95c028
aqt7e36b1f2-4646-4075-ac0b-5fdee475cd7e    103    4      518         0x4d95c758

mart: school3
accelerator: DWAFINAL
last load: 2011/07/29 07:01:50

AQT name                                  FactTab #tab #matched     address
------------------------------------------------------------------------------
aqt92b36a8a-1567-4146-833c-385cd103f5d4    103    1      678         0x4d95cac0
aqt3189bec1-b6c9-417d-b969-92c687ef2e44    103    2       59         0x4d95cc18
aqt8d3b3dc8-59b6-4e34-822b-75b06b99c900    103    2     4487         0x4d90c0d8
aqt5f9c2a05-9131-4738-a929-036fcf77f65c    103    3       71         0x4d90c2e0
aqtee08ed16-6a5c-4478-ac57-fc4f99539c74    103    3      795         0x4d95ce20
aqt04d1c96a-022b-4ed7-938d-caf765bc9926    103    4      367         0x4d90c598

18 entries
```

If you use the AQT name for the optional *aqt_name* parameter, the command prints information about the specific AQT.

Figure 2. **onstat -g aqt aqt_name** command output

```
AQT: aqt6de1afdd-f10a-45b0-93e9-0c208405fefd
database: iwadb
AQT tabid: 125
Fact table: 111
Number of times matched: 8947

Join structure: alias(tabid)[colno,...] = alias(tabid)[colno,...] {u:unique}
0(111)[1] = 1(110)[1] u
            1(110)[2] = 2(109)[1] u
                        2(109)[5] = 3(101)[1] u
                                    3(101)[3] = 4(100)[1] u
0(111)[2] = 5(106)[1] u
            5(106)[2] = 6(103)[1] u
            5(106)[3] = 7(104)[1] u
            5(106)[4] = 8(105)[1] u
                        8(105)[3] = 9(101)[1] u
                                    9(101)[3] = 10(100)[1] u
            5(106)[5] = 11(102)[1] u
0(111)[2,3] = 15(108)[1,2] u
              15(108)[1] = 16(106)[1] u
                           16(106)[2] = 17(103)[1] u
                           16(106)[3] = 18(104)[1] u
                           16(106)[4] = 19(105)[1] u
                                        19(105)[3] = 20(101)[1] u
                                                     20(101)[3] = 21(100)[1] u
                           16(106)[5] = 22(102)[1] u
              15(108)[2] = 23(107)[1] u
                           23(107)[2] = 24(101)[1] u
                                        24(101)[3] = 25(100)[1] u
0(111)[3] = 12(107)[1] u
            12(107)[2] = 13(101)[1] u
                         13(101)[3] = 14(100)[1] u
```

Output description

The AQTs are grouped by the data mart that they belong to. The groups are sorted by accelerator name, and then by data mart name. Within the data mart groups, the AQTs are sorted in the following order: Fact table tabid (`FactTab`), number of tables (`#tab`), and AQT name.

The output comes from the entries in the dictionary cache that refer to the AQTs of the data marts. The output is shown only if the AQTs have been loaded into the dictionary cache, which normally occurs when a query is being matched against the AQTs.

Before the server attempts to match a query against the AQTs, the AQTs do not have any entries in the dictionary cache. The **onstat -g aqt** command will not show any entries in the output. When the dictionary cache is initialized during the database server startup, the columns `#matched` and `address` get new values.

The **onstat -g aqt** command prints the following information:

mart
: The name of the data mart

accelerator
: The name of the accelerator instance

last load
: The time stamp for when the data mart was last loaded

AQT name
: The unique system-generated name of the AQT

FactTab
: The tabid of the fact table for the AQT

#tab
: The number of tables that are part of the AQT

#matched
: The counter for query matches that have occurred for the AQT

address
: The internal database server memory address for the AQT

The **onstat -g aqt *aqt_name*** command prints the following information:

AQT
: The unique system-generated name of the AQT

database
: The name of the database to which the AQT belongs

AQT tabid
: The tabid for the entry that constitutes the AQT in the `systables` system catalog table of the database server.

Fact table
: The tabid of the fact table of the AQT

Number of times matched
: The counter for query matches that have occurred for the AQT

The information about the AQT is followed by a textual representation of the star schema of the data mart. The textual representation shows how the columns of the tables are related to each other in the star join.

For information about the Informix® Warehouse Accelerator, see the *IBM® Informix Warehouse Accelerator Administration Guide*.

---

onstat -g arc command: Print archive status

Use the **onstat -g arc** command to display information about the last committed archive for each dbspace and also information about any current ongoing archives.

Syntax:

```
>>-onstat-- -g--arc-----------------------------------------><
```

Example output

Figure 1. **onstat -g arc** command output

```
Dbspaces - Ongoing archives
number   name            Q Size Q Len  buffer partnum    size     Current-page
1        rootdbs         100    3      100    0x1001c9   0        1:128
3        datadbs01       0      0
4        datadbs02       0      0

Dbspaces - Archive Status
name               number level date              log         log-position
rootdbs            1      0     07/30/2009.09:59 28          0x320018
datadbs01          3      0     07/30/2009.09:59 28          0x320018
datadbs02          4      0     07/30/2009.09:59 28          0x320018
```

Output description - Ongoing archives

This output section represents current information about the archives. If no archives are active in the system, this section is not displayed.

Column	Description
Number	The number of the dbspace
Name	The name of the dbspace
Q Size	The before-image queue list size. This information is primarily for support.
Q Len	The before-image queue length. This information is primarily for support.
Buffer	The number of pages used in the before-image buffer
Partnum	The partition number of the before-image bin
Size	The number of pages in the before-image bin

Current-page	The current page that is being archived

Note: The before-image bin is a temporary table created in a temporary dbspace, or in the root dbspace if you do not have any temporary dbspaces. If the before-image bin becomes too small, it can extend to additional partitions, in which case the output will display see multiple Partnum and Size fields for the same dbspace.

Output description - Archive status

This output section contains information about the last backup that has occurred for each dbspace.

Column	Description
Name	The name of the dbspace
Number	The dbspace number
Level	The archive level
Date	The date and time of the last archive
Log	The unique ID (UNIQID) of the checkpoint that was used to start the archive
Log-position	The log position (LOGPOS) of the checkpoint that was used to start the archive

onstat -g ath command: Print information about all threads

Use the **onstat -g ath** command to display information about all threads.

Syntax:

```
>>-onstat-- -g--ath-------------------------------------------><
```

Example output

Figure 1. onstat -g ath command output

```
Threads:
 tid      tcb              rstcb         prty status              vp-class       name
 2        10bbf36a8         0             1    IO Idle              3lio         lio vp 0
 3        10bc12218         0             1    IO Idle              4pio         pio vp 0
 4        10bc31218         0             1    running              5aio         aio vp 0
 5        10bc50218         0             1    IO Idle              6msc         msc vp 0
 6        10bc7f218         0             1    running              7aio         aio vp 1
 7        10bc9e540         10b231028     1    sleeping secs: 1     1cpu         main_loop()
 8        10bc12548         0             1    running              1cpu         tlitcppoll
 9        10bc317f0         0             1    sleeping forever     1cpu         tlitcplst
 10       10bc50438         10b231780     1    IO Wait              1cpu         flush_sub(0)
 11       10bc7f740         0             1    IO Idle              8aio         aio vp 2
 12       10bc7fa00         0             1    IO Idle              9aio         aio vp 3
 13       10bd56218         0             1    IO Idle             10aio         aio vp 4
 14       10bd75218         0             1    IO Idle             11aio         aio vp 5
 15       10bd94548         10b231ed8     1    sleeping forever     1cpu         aslogflush
 16       10bc7fd00         10b232630     1    sleeping secs: 34    1cpu         btscanner 0
 32       10c738ad8         10b233c38     1    sleeping secs: 1     1cpu         onmode_mon
 50       10c0db710         10b232d88     1    IO Wait              1cpu         sqlexec
```

Output description

tid
: Thread ID

tcb
: Thread control block access

rstcb
: RSAM thread control block access

prty
: Thread priority

status
: Thread status

vp-class
: Virtual processor class

name
: Thread name. For threads that are participating in parallel storage optimization operations, the name of the operation and the thread number.

- compress.*number* = The thread is compressing data
- repack.*number* = The thread is repacking data
- uncompress.*number* = The thread is uncompressing data
- update_ipa.*number* = The thread is removing outstanding in-place alter operations

Related reference:
onstat -g wst command: Print wait statistics for threads
NUMFDSERVERS configuration parameter

table or fragment arguments: Compress data and optimize storage (SQL administration API)

Related information:

Threads shown by the onstat -g ath command

onstat -g bth and -g BTH: Print blocked and waiting threads

Use the **onstat -g bth** command to display the dependencies between blocking and waiting threads. Use the **onstat -g BTH** command to display session and stack information for the blocking threads.

Syntax:

```
>>-onstat-- -g--+-bth-+-----------------------------------------><
               '-BTH-'
```

Example output for onstat -g bth

Figure 1. onstat -g bth command output

```
This command attempts to identify any blocking threads.

Highest level blocker(s)
 tid      name                 session
 48       sqlexec              26

Threads waiting on resources
 tid      name                 blocking resource              blocker
 49       sqlexec              MGM                            48
 13       readahead_0          Condition (ReadAhead)           -
 50       sqlexec              Lock (0x4411e578)              49
 51       sqlexec              Lock (0x4411e578)              49
 52       sqlexec              Lock (0x4411e578)              49
 53       sqlexec              Lock (0x4411e578)              49
 57       bf_priosweep()       Condition (bp_cond)             -
 58       scan_1.0             Condition (await_MC1)           -
 59       scan_1.0             Condition (await_MC1)           -

Run 'onstat -g BTH' for more info on blockers.
```

Ouput description for onstat -g bth

tid
: Thread ID

name
: Thread name

session
: Session ID

blocking resource
: Type of resource for which the listed thread is waiting

blocker
: ID of the thread that is blocking the listed thread

Example output for onstat -g BTH

```
Stack for thread: 48 sqlexec
 base: 0x00000000461a3000
  len:   69632
   pc: 0x00000000017b32c3
  tos: 0x00000000461b2e30
state: ready
   vp: 1

0x00000000017b32c3 (oninit) yield_processor_svp
0x00000000017bca6c (oninit) mt_wait
0x00000000019d4e5c (oninit) net_buf_get
0x00000000019585bf (oninit) recvsocket
0x00000000019d1759 (oninit) tlRecv
0x00000000019ce62d (oninit) slSQIrecv
0x00000000019c43ed (oninit) pfRecv
0x00000000019b2580 (oninit) asfRecv
0x000000000193db2a (oninit) ASF_Call
0x0000000000c855dd (oninit) asf_recv
0x0000000000c8573c (oninit) _iread
0x0000000000c835cc (oninit) _igetint
0x0000000000c72a9e (oninit) sqmain
0x000000000194bb38 (oninit) listen_verify
0x000000000194ab8a (oninit) spawn_thread
0x0000000001817de3 (oninit) th_init_initgls
0x00000000017d3135 (oninit) startup
```

```
This command attempts to identify any blocking threads.

Highest level blocker(s)
 tid      name                session
 48       sqlexec             26

session          effective                      #RSAM    total    used    dynamic
id       user    user      tty    pid    hostname threads  memory   memory  explain
26       informix -        45     31041  mors     2        212992   186568  off

Program :
/work3/JC/VIEWS/jc_dct_phase2.view/.s/00055/80003fd351f804d3dbaccess

tid      name     rstcb            flags    curstk   status
48       sqlexec  448bc5e8         ---P---  4560     ready-
58       scan_1.0 448bb478         Y------  896      cond wait  await_MC1 -

Memory pools    count 2
name          class addr            totalsize  freesize   #allocfrag #freefrag
26            V     45fcc040        208896     25616      189        16
26*00         V     462ad040        4096       808        1          1

name            free       used            name            free       used
overhead        0          6576            mtmisc          0          72
resident        0          72              scb             0          240
opentable       0          7608            filetable       0          1376
log             0          33072           temprec         0          17744
blob            0          856             keys            0          176
ralloc          0          55344           gentcb          0          2240
ostcb           0          2992            sqscb           0          21280
sql             0          11880           xchg_desc       0          1528
xchg_port       0          1144            xchg_packet     0          440
xchg_group      0          104             xchg_priv       0          336
hashfiletab     0          1144            osenv           0          2520
sqtcb           0          15872           fragman         0          1024
shmblklist      0          416             sqlj            0          72
rsam_seqscan    0          368

sqscb info
scb               sqscb          optofc   pdqpriority optcompind  directives
4499c1c0          461c1028       0        100         2           1

Sess       SQL             Current       Iso Lock        SQL  ISAM F.E.
Id         Stmt type       Database      Lvl Mode        ERR  ERR  Vers  Explain
26         SELECT          jc            CR  Not Wait   0    0    9.24  Off

Current statement name : unlcur

Current SQL statement (5) :
  select * from systables,syscolumns,sysfragments

Last parsed SQL statement :
  select * from systables,syscolumns,sysfragments
```

Ouput description for onstat -g BTH

tid
: Thread ID

name
: Thread name

session
: Session ID

The session information section contains the same information that is output from the **onstat -g ses** command. See onstat -g ses command: Print session-related information.

The remainder of the information displays the stack information for the thread.

Related information:
Monitor blocking threads with the onstat -g bth and onstat -g BTH commands

onstat -g buf command: Print buffer pool profile information

Use the **onstat -g buf** command to show profile information for each buffer pool.

Syntax:

```
>>-onstat-- -g--buf-------------------------------------------><
```

Example output

The output of the **onstat -g buf** command varies slightly depending on whether the BUFFERPOOL configuration parameter setting contains the **memory** field or the **buffers** field. The output for the **memory** setting is shown. The output for the **buffers** setting contains the **max extends** and **next buffers** fields instead of the **max memory** and **next memory** fields.

Figure 1. onstat -g buf output for the memory setting

```
Profile

Buffer pool page size: 2048
dskreads   pagreads   bufreads   %cached dskwrits   pagwrits   bufwrits   %cached
1190       1773       661359     99.82   16863      83049      185805     90.92
bufwrits_sinceckpt  bufwaits   ovbuff     flushes
11243               115        0          42

Fg Writes     LRU Writes    Avg. LRU Time Chunk Writes  Total Mem
0             0             nan           10883         32Mb

                                    cache
# extends  max memory  next memory  hit ratio  last
0          128Mb       32Mb         90         11:31:17

Bufferpool Segments
id segment      size     # buffs
0  0x449f0000   32Mb     13025

----------------------------------

Buffer pool page size: 8192
dskreads   pagreads   bufreads   %cached dskwrits   pagwrits   bufwrits   %cached
0          0          11         100.00  4          16         4          0.00
bufwrits_sinceckpt  bufwaits   ovbuff     flushes
0                   0          0          1

Fg Writes     LRU Writes    Avg. LRU Time Chunk Writes  Total Mem
0             0             nan           4             128Mb

                                    cache
# extends  max memory  next memory  hit ratio  last
0          1280Mb      128Mb        90         11:31:41

Bufferpool Segments
id segment      size     # buffs
0  0x4928e000   128Mb    14988

----------------------------------

Fast Cache Stats
gets       hits       %hits   puts
246854     244407     99.01   111147
```

Output description

Buffer pool page size
: The number of bytes in a page in the buffer pool

dskreads
: The number of disk read operations that are performed to bring pages into this buffer pool. Each read operation reads one or more pages.

pagreads
: The number of pages that are read from disk to this buffer pool.

bufreads
: The number of times a memory image for a page was read from this buffer pool.

%cached
: The percentage of page reads for this buffer pool that were satisfied by a cached page image (rather than having to perform a disk read). Computed as `(bufreads - dskreads) / bufreads x 100`. Higher percentages indicate better caching performance.

dskwrits
: The number of disk write operations that are performed to write changed pages from this buffer pool back to disk. Each write operation writes one or more pages.

pagwrits
: The number of pages that are written to disk from this buffer pool.

bufwrits
: The number of times a memory image of a page was written to in this buffer pool.

%cached
: The percentage of page writes for this buffer pool that were satisfied by a cached page image (rather than having to perform a disk write). Computed as `(bufwrits - dskwrits) / bufwrits x 100`.

bufwrits_sinceckpt
: The number of times a memory image of a page was written to in this buffer pool since the last checkpoint.

bufwaits
: The number of times a thread had to wait for a lock on a buffer in this buffer pool. Higher numbers indicate more contention among multiple threads for mutually incompatible locks on the same pages.

ovbuff
: The number of times a changed buffer from this buffer pool was written to disk specifically to create a free buffer to read another requested page. If the `ovbuff` value is high, the buffer pool might not be large enough to hold the working set of pages that are needed by applications. An insufficient buffer pool can lead to performance degradation.

flushes
: The number of times the server flushed all dirty buffers at once in the buffer pool. Mass flushing can occur for various reasons, including as part of checkpoint processing or if the buffer pool is running out of clean buffers despite normal LRU cleaning activity.

Fg Writes
: Number of changed buffers from this buffer pool that were written to disk by a non-I/O flusher thread that was accessing the buffer. This number is a superset of the value of the `ovbuff` field. In addition to the writes to service page faults that are counted in the `ovbuff` field, this value also includes foreground writes to

maintain the consistency of database logs and reserved pages to ensure a correct recovery.

LRU Writes
: The number of changed buffers from this buffer pool that were written to disk by an LRU cleaner thread. LRU cleaners are activated if the buffer pool exceeds the value that is specified in the **lru_max_dirty** field of the BUFFERPOOL configuration parameter or if foreground writes occur due to buffer pool overflows.

Avg. LRU Time
: The average amount of time that is taken by an LRU cleaner thread to clean a single LRU chain.

Chunk Writes
: The number of changed buffers that were written to disk by a chunk cleaning operation. Chunk cleaning writes out all changed buffers of a chunk that are in the buffer pool. Chunk cleaning is done to clean many buffers quickly, such as during checkpoint processing and fast recovery.

Total Mem
: The size of the buffer pool.

extends
: The number of times that the buffer pool was extended.

max memory (memory setting)
: The target maximum size of the buffer pool. The actual size of the buffer pool can exceed this value, but not more than the size of one segment.

max extends (buffers setting)
: The maximum number of times that the buffer pool can be extended. (This field is not shown in the example output.)

next memory (memory setting)
: The size of the next extension of the buffer pool.

next buffers (buffers setting)
: The number of buffers for the next extension of the buffer pool. (This field is not shown in the example output.)

cache hit ratio
: The read cache hit ratio below which the buffer pool is extended.

last
: The time of the last extension of the buffer pool.

id
: The ID of the buffer pool segment.

segment
: The internal address of the buffer pool segment.

size
: The size of the buffer pool segment.

buffs
: The number of buffers in the buffer pool segment.

Fast Cache Stats
: Statistics for the fast cache, which is a type of cache that reduces the time that is needed for accessing the buffer pool.

gets
: The number of times the server looked for a buffer in the fast cache.

hits
: The number of times that the server found the buffer it was searching for in the fast cache.

%hits
: The percentage of hits, which is `hits*100/gets`.

puts
: The number of times that the server inserted buffers inserted into the fast cache.

Related reference:
BUFFERPOOL configuration parameter
Related information:
Monitor buffers

onstat -g cac command: Print information about caches

Use the **onstat -g cac** command to see summary and detailed information about all caches or about a single cache.

Syntax:

Use the **onstat -g cac** command without any options to see information about all caches.

Use the following options to see information about a specific cache:

agg
: Prints information about the aggregate cache.

aqt
: Prints information about the AQT dictionary cache. Prints the same output as the **onstat -g aqt** command. See onstat -g aqt command: Print data mart and accelerated query table information.

am
: Prints information about the access method cache. To see information for a specific access method, include the access method name.

cast
: Prints information about the cast cache.

dic
: Prints information about the data dictionary cache. Prints the same output as the **onstat -g dic** command. See onstat -g dic command: Print table information.

dsc
: Prints information about the data distribution cache. Prints the same output as the **onstat -g dsc** command. See onstat -g dsc command: Print distribution cache information.

ed
: Prints information about the external directives cache.

lbacplcy
: Prints information about the LBAC security policy information cache.

lbacusrc
: Prints information about the LBAC credential memory cache.

opci
: Prints information about the operator class instance cache.

prc
: Prints information about the UDR cache. Prints the same output as the **onstat -g prc** command. See onstat -g prc command: Print sessions using UDR or SPL routines.

prn
: Prints information about the procedure name cache.

rr
: Prints information about the routine resolution cache.

ssc
: Prints information about the SQL statement cache. Prints the same output as the **onstat -g ssc** command. See onstat -g ssc command: Print SQL statement occurrences.

ttype
: Prints information about the secondary transient cache.

typei
: Prints information about the extended type by ID cache. To see information for a specific extended type, include the extended type ID.

typen
: Prints information about the extended type by name cache. To see information for a specific extended type, include the extended type name.

Example output

The output of most **onstat -g cac** commands contains similar format and information.

The following output is an example of the **onstat -g cac lbacplcy** command:

```
Security Policy Info Cache Entries:

list id   ref  drop hits       last_access          heap_ptr         item
--------------------------------------------------------------------------------

9    2    0    0    0          2020-05-12 10:36:34  65f1b8d0         test@informix: : secpolicyid 2

15   1    0    0    0          2020-05-12 10:36:34  65f1b4d0         test@informix: : secpolicyid 1


    Total number of entries  : 2
    Number of entries in use : 0
```

Output description

The output of most **onstat -g cac** commands contains the following fields:

Number of lists
: Number of lists in the distribution cache

configuration parameter name
: Number of entries that can be cached at one time

list
: Distribution cache hash chain ID

id
: The unique ID assigned to the cache entry

ref
: Number of statements that reference a cache entry

drop
: Whether this entry was dropped after it was added to the cache

hits
: The number of times the cache entry is accessed

last_access
: The time at which the cache entry was last accessed.

heap_ptr
: Heap address that is used to store this entry

item name
: The name of the item in the cache

Total number of entries
: Number of entries in the cache

Number of entries in use
: Number of entries that are being used

Related information:
Configure and monitor memory caches

onstat -g ckp command: Print checkpoint history and configuration recommendations

Use the **onstat -g ckp** command to print checkpoint history and show configuration recommendations if a suboptimal configuration is detected.

Syntax:

```
>>-onstat-- -g--ckp-----------------------------------------><
```

Example output

Figure 1. onstat -g ckp command output

```
Auto Checkpoints=On   RTO_SERVER_RESTART=60 seconds   Estimated recovery time 7 seconds

                                                          Critical Sections
         Clock                          Total  Flush  Block  #      Ckpt  Wait  Long  #Dirty
Interval Time     Trigger  LSN          Time   Time   Time   Waits  Time  Time  Time  Buffers
1        18:41:36 Startup  1:f8         0.0    0.0    0.0    0      0.0   0.0   0.0   4
2        18:41:49 Admin    1:11c12cc    0.3    0.2    0.0    1      0.0   0.0   0.0   2884
3        18:42:21 Llog     8:188        2.3    2.0    2.0    1      0.0   2.0   2.0   14438
4        18:42:44*User     10:19c018    0.0    0.0    0.0    1      0.0   0.0   0.0   39
5        18:46:21 RTO      13:188       54.8   54.2   0.0    30     0.6   0.4   0.6   68232

           Physical Log    Logical Log
  Dskflu   Total   Avg      Total   Avg
  /Sec     Pages   /Sec     Pages  /Sec
  4        3       0        1       0
  2884     1966    163      4549    379
  7388     318     10       65442   2181
  39       536     21       20412   816
  1259     210757  1033     150118  735

Max Plog   Max Llog    Max Dskflush    Avg Dskflush   Avg Dirty   Blocked
pages/sec  pages/sec   Time            pages/sec      pages/sec   Time
8796       6581        54              43975          2314         0
```

Output description

Auto Checkpoints
: Indicates if the AUTO_CKPTS configuration parameter is on or off

RTO_SERVER_RESTART
: Displays the RTO time in seconds. Zero (0) means that RTO is off.

Estimated recovery time ## seconds
: Indicates the estimated recovery time if the data server stops responding. This value appears only if RTO_SERVER_RESTART is active.

Interval
: Checkpoint interval ID

Clock Time
: Clock time when checkpoint occurred.

Trigger
: Event that triggered the checkpoint. An asterisk (*) indicates that the checkpoint that was requested was a transaction-blocking checkpoint.

Trigger name	Description
Admin	Administrator-related tasks. For example: • Create, drop, or rename a dbspace • Add or drop a chunk • Add or drop a log file • Change physical log size or location • After "shrink" operation on partition • Turn on or off mirroring
Backup	Back up related operations. For example:

	• Fake backup • Start of an archive • After the completion of a physical restore
CDR	ER subsystem is started for the first time, or is restarted after all of the replication participants were removed.
CKPTINTVL	When the checkpoint interval expires. The checkpoint interval is the value that is specified for the CKPTINTVL parameter in the onconfig file.
Conv/Rev	Conversion reversion checkpoint. After the check phase of convert and before the actual conversion of disk structures. After the reversion is completed also triggers a checkpoint.
HA	High availability. For example: • A new RSS or SDS node is added to a High Availability cluster • A secondary server is promoted to a primary server • The physical log file is low on a secondary server
HDR	High-Availability Data Replication. For example: • The mode of the server is changed • The start of the first transfer after HDR is set up • There is the potential for a physical log overflow on primary or secondary servers
IPL	Trigger checkpoint to reduce physical log usage on the secondary server. Index page logging can cause foreground writes and heavy physical log usage on secondary servers.
Lightscan	Before the look aside is turned off on partitions.
Llog	Running out of logical log resources.
LongTX	Long Transaction. If a long transaction was found but not stopped, a checkpoint is initiated to stop the transaction. During rollback, a checkpoint is initiated in the rollback phase if a checkpoint has not already happened after long transaction was aborted.
Misc	Miscellaneous events. For example: • A dbspace or chunk is being brought down because of I/O errors • During rollback when the addition of the chunk is being undone: for example, when removing the chunk.
Pload	When the High-Performance Loader starts in the Express mode.
Plog	Physical log has one of the following conditions: • Physical log is 75% full • The amount of physical log used plus the number of dirty partitions is more than 90% of physical log size
Restore Pt	Restore Point. Checkpoints at the start and end of a restore point. The restore point is (used by conversion guard) CONVERSION_GUARD configuration parameter is enabled and a temporary directory is specified in the RESTORE_POINT_DIR configuration parameter.
Recovery	During a restore, at the start of a fast recovery.
Reorg	At the start of online index build.
RTO	Maintaining the Recovery Time Objective (RTO) policy. During normal operations, when the restart time after a crash might exceed the value that is set for the RTO_SERVER_RESTART configuration parameter.
Stamp Wrap	Checkpoint timestamp. If the new checkpoint timestamp appears to be before the last written checkpoint, then the timestamp is advanced out of interval between checkpoints. Another checkpoint is triggered.
Startup	At the startup of the database server.
Uncompress	Uncompress commands that are issued on a table or partition. This applies only for checkpoints on tables or databases that are not logged.
User	A checkpoint request is submitted by the user.

LSN
: Logical log position where checkpoint is recorded

Total Time
: Total checkpoint duration, in seconds, from request time to checkpoint completion

Flush Time
: Time, in seconds, to flush buffer pools

Block Time
: Time a transaction was blocked, in seconds, by a checkpoint that was triggered by a scarcity of some needed resource. For example, running out of physical log, or wrap-around of the logical log.

Waits
: Number of transactions that are blocked waiting for checkpoint

Ckpt Time
: Time, in seconds, for all transactions to recognize a requested checkpoint

Wait Time
: Average time, in seconds, that transactions waited for checkpoint

Long Time
: Longest amount of time, in seconds, a transaction waited for checkpoint

Dirty Buffers
: Number of dirty buffers that are flushed to disk during checkpoint

Dskflu/sec
: Number of buffers that are flushed per second

Physical Log Total Pages
: Total number of pages that are physically logged during checkpoint interval

Physical Log Avg/Sec
: Average rate of physical log activity during checkpoint interval

Logical Log Total Pages
: Total number of pages that are logically logged during checkpoint interval

Logical Log Avg/Sec
: Average rate of logical log activity during checkpoint interval

Max Plog pages/sec
: Maximum rate of physical log activity during checkpoint interval

Max Llog pages/sec
: Maximum rate of logical log activity during checkpoint interval

Max Dskflush Time
: Maximum time, in seconds, to flush buffer pools to disk

Avg Dskflush pages/sec
: Average rate buffer pools are flushed to disk

Avg Dirty pages/sec
: Average rate of dirty pages between checkpoints

Blocked Time
: Longest blocked time, in seconds, since the database server was last started

Performance advisory messages

If the data server detects a configuration that is less than optimal, a performance advisory message with tuning recommendations appears below the checkpoint history. This performance advisory message also appears in the message log. Following are examples of performance advisory messages:

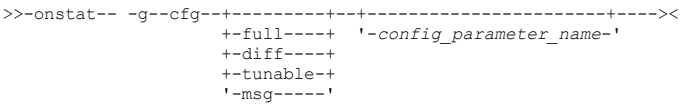
```
Physical log is too small for bufferpool size. System performance may be
less than optimal.
Increase physical log size to at least %ldKb

Physical log is too small for optimal performance.
Increase the physical log size to at least $ldKb.

Logical log space is too small for optimal performance.
Increase the total size of the logial log space to at least %ld Kb.

Transaction blocking has taken place. The physical log is too small.
Please increase the size of the physical log to %ldKb

Transaction blocking has taken place. The logical log space is too small.
Please increase the size of the logical log space to %ldKb
```

Related reference:
sysckptinfo

onstat -g cfg command: Print the current values of configuration parameters

Use the **onstat -g cfg** command to print a list of configuration parameters with their current values. You can use more command options to print more information about the configuration parameters.

Syntax:



This **onstat -g cfg** command has the following formats:

Command	Description
onstat -g cfg	Displays a list of configuration parameters and their current values.
onstat -g cfg *config_parameter_name*	Displays only the current value of the specified configuration parameter.
onstat -g cfg full	Displays all of the information about each configuration parameter, including the current value, the default value, the onconfig file value, and a description of the parameter.
onstat -g cfg full *config_parameter_name*	Displays all of the information about the specified parameter.
onstat -g cfg diff	Displays information about the configuration parameters with current values that are different from the permanent values that are in the onconfig file.
onstat -g cfg tunable	Displays the default, original, and current values for all tunable parameters. An asterisk indicates that you can tune a configuration parameter dynamically.
onstat -g cfg msg	Displays any messages, such as warnings or adjustments, that are associated with configuration parameters.

Example output

The following portion of sample output of the **onstat -g cfg** command shows that the value of the DEADLOCK_TIMEOUT configuration parameter was dynamically changed to 90 seconds after the database server started:

```
id   name                 type   units   rsvd  tunable
26   DEADLOCK_TIMEOUT     INT4   Seconds       *

     min/max : 0,2147483647
     default : 60
     onconfig:
     current : 90

     Description:
     Use the DEADLOCK_TIMEOUT configuration parameter to specify the
     maximum number of seconds that a database server thread can wait
     to acquire a lock.

ROOTNAME                rootdbs
```

The following portion of sample output of the **onstat -g cfg diff** command shows the default, current, and onconfig file values of the TBLTBLFIRST and TBLTBLNEXT configuration parameters:

```
id   name                      type   units   rsvd  tunable
53   TBLTBLFIRST               INT4   KB            *

     default : 500
     onconfig: 0
     current : 250


id   name                      type   units   rsvd  tunable
54   TBLTBLNEXT                INT4   KB            *

     default : 100
     onconfig: 0
     current : 150
```

The following portion of sample output shows information for the MSGPATH configuration parameter. Here, there is no default value that is built into the configuration parameter and the onconfig file and current values are the same.

```
id   name                      type   units   rsvd  tunable
10   MSGPATH                   CHAR           *     *

     default :
     onconfig: /work2/JC/online.log
     current : /work2/JC/online.log
```

The following portion of sample output of the **onstat -g cfg msg** command shows messages that identify configuration parameters with changed values:

```
Configuration Parameters With Messages

name             message
TBLTBLFIRST      Parameter's user-configured value was adjusted.
TBLTBLNEXT       Parameter's user-configured value was adjusted.
BUFFERPOOL       Parameter's user-configured value was adjusted.
STACKSIZE        Parameter's user-configured value was adjusted.
VPCLASS          Parameter's user-configured value was adjusted.
```

Output description

name
: Name of the configuration parameter

type
: Data type for the value

units
: Units in which the value is expressed

rsvd
: Indicates (with an asterisk) that the configuration parameter and its value are stored on the configuration reserved page
If an asterisk is not present, the configuration parameter and its value are not stored on the configuration reserved page.

tunable
: Indicates (with an asterisk) that the configuration parameter can be tuned dynamically, for example, with an **onmode -wm** or **-wf** command
If an asterisk is not present, the configuration parameter cannot be tuned dynamically.

min/max
: Minimum and maximum values for the configuration parameter

default
: Default value that is built into the server for the configuration parameter

onconfig
: Value of the configuration parameter, if any, that is in the onconfig.std file

current
: Current® value of the configuration parameter
A current value is different if it was modified dynamically, for example, with an **onmode -wm** command.

Description
: Description of the configuration parameter

message

Message that identifies a changed configuration parameter value

Related tasks:
Displaying the settings in the onconfig file
Related reference:
onstat -c command: Print ONCONFIG file contents
onmode -wf, -wm: Dynamically change certain configuration parameters
Database server files

onstat -g cluster command: Print high-availability cluster information

Use the **onstat -g cluster** command to display information about the servers in a high-availability cluster environment.

Syntax:

```
>>-onstat-- -g--cluster--+---------+--------------------------><
                         '-verbose-'
```

The **onstat -g cluster** command combines the functionality of **onstat -g dri**, **onstat -g sds**, and **onstat -g rss**. The output of the **onstat -g cluster** command differs slightly depending on whether the command is run on the primary server or on one of the secondary servers.

Example output (primary server)

Following is sample output from the **onstat -g cluster** command. The sample shows output when the command is run on the primary server.

Figure 1. **onstat -g cluster** command output (run on the primary server)

```
Primary Server:serv1
Current Log Page:16,476
Index page logging status: Enabled
Index page logging was enabled at: 2013/12/11 14:05:17


Server ACKed Log    Applied Log  Supports     Status
       (log, page)  (log, page)  Updates
serv2  16,476       16,476       Yes          SYNC(SDS),Connected,Active
serv3  16,476       16,476       Yes          ASYNC(HDR),Connected,On
serv4  16,476       16,476       Yes          ASYNC(RSS),Connected,Active
```

Output description (primary server)

Primary server
: The name assigned to the primary server.

Current® log page
: The log ID and page number of the current log page.

Index page logging status
: Indicates whether index page logging is enabled or disabled.

Index page logging was enabled at
: The date and time that index page logging was enabled.

Server
: The name of the secondary server.

ACKed Log (log, page)
: The log ID and page number of the last acknowledged log transmission.

Applied Log (log, page)
: The log ID and page number of the last applied log transmission.

Supports Updates
: Displays whether client applications can perform update, insert, and delete operations on the secondary server (as specified by the UPDATABLE_SECONDARY configuration parameter).

Status
: Displays the connection status of the secondary server

Example output (primary server, verbose output)

Figure 2. **onstat -g cluster verbose** command output (run on the primary server)
Following is sample output from the **onstat -g cluster verbose** command. The sample shows output when the command is run on the primary server with the verbose option.

```
Primary Server:serv1
        Current Log Page:16,479
        Index page logging status: Enabled
        Index page logging was enabled at: 2013/12/11 14:05:17


        ------------------------------------------------------------

        server name: serv3
```

```
        type: ASYNC (HDR)
        control block: 0x4b673018
        server status: On
        connection status: Connected
        Last log page sent (log id, page): 16,479
        Last log page acked (log id, page); 16,479
        Last log page applied (log id, page); 16,479
        Approximate log page backlog: 0
        SDS cycle not used
        Delayed Apply Not Used
        Stop Apply Not Used
        Time of last ack: 2013/12/11 14:09:12
        Supports Updates: Yes


        ------------------------------------------------------------

        server name: serv2
        type: SYNC (SDS)
        control block: 0x4c2de0b8
        server status: Active
        connection status: Connected
        Last log page sent (log id, page): 16,479
        Last log page acked (log id, page); 16,479
        Last log page applied (log id, page); 16,479
        Approximate log page backlog: 0
        SDS cycle current: 20 ACKed: 20
        Delayed Apply Not Used
        Stop Apply Not Used
        Time of last ack: 2013/12/11 14:09:13
        Supports Updates: Yes
```

Output description (primary server, verbose output)

Primary server
: The name of the primary server

Current log page
: The log ID and page number of the current log page.

Index page logging status
: Indicates whether index page logging is enabled or disabled.

Index page logging was enabled at
: The date and time that index page logging was enabled.

Server name
: The name of the secondary server.

type
: Displays whether the secondary server is connected synchronously (SYNC) or asynchronously (ASYNC). Also displays the type of secondary server: HDR, SDS, or RSS.

control block
: The in-memory address of the thread control block.

server status
: Displays the current status of the secondary server.

connection status
: Displays the current network connection status of the secondary server.

Last log page sent (log id, page)
: The log ID and page number of the most recent log page sent by the primary server to the secondary server.

Last log page acked (log id, page)
: The log ID and page number of the most recent log page the secondary server acknowledged.

Last log page applied (log id, page)
: The log ID and page number of the most recent log page the secondary server applied.

Approximate log page backlog
: Indicates the approximate number of log pages that have yet to be processed by the secondary server.

SDS cycle
: Indicates the cycle number to which the primary server has advanced and which the shared disk secondary server has acknowledged. Used internally by support to monitor coordination of the primary server with the secondary server.

Delayed Apply
: Indicates whether the secondary server waits for a specified amount of time before applying logs (as specified by the DELAY_APPLY configuration parameter).

Stop Apply
: Indicates whether the secondary server has stopped applying log files received from the primary server (as specified by the STOP_APPLY configuration parameter).

Time of last ack
: The date and time of the last acknowledged log.

Supports Updates
: Displays whether client applications can perform update, insert, and delete operations on the secondary server (as specified by the UPDATABLE_SECONDARY configuration parameter).

Example output (secondary server)

Following is sample output from the **onstat -g cluster** command. The sample shows output when the command is run on a secondary server.

Figure 3. **onstat -g cluster** command output (run on the secondary server)

```
Primary Server:serv1
        Index page logging status: Enabled
        Index page logging was enabled at: 2010/01/11 14:05:17
```

```
        Server ACKed Log    Applied Log   Supports     Status
               (log, page)  (log, page)   Updates
        serv2  16,479       16,479        Yes          SYNC(SDS),Connected,Active
```

Output description (secondary server)

Primary server
The name of the primary server
Index page logging status
Indicates whether index page logging is enabled or disabled.
Index page logging was enabled at
The date and time that index page logging was enabled.
Server
The name of the secondary server.
ACKed Log (log, page)
The log ID and page number of the last acknowledged log.
Applied Log (log, page)
The log ID and page number of the last applied log transmission.
Supports Updates
Displays whether client applications can perform update, insert, and delete operations on the secondary server (as specified by the UPDATABLE_SECONDARY configuration parameter).
Status
Displays the connection status of the secondary server.

Related reference:
DELAY_APPLY Configuration Parameter
STOP_APPLY configuration parameter
UPDATABLE_SECONDARY configuration parameter

onstat -g cmsm command: Print Connection Manager information

Use the **onstat -g cmsm** command to display information about a specific Connection Manager, or all of the Connection Managers that are attached to the database server the command is run on.

Syntax:

```
>>-onstat-- -g--cmsm--+-------------------------+--------------><
                      '-connection_manager_name-'
```

Usage

onstat -g cmsm displays information about connection units the Connection Manager connects to, the number of connections each Connection Manager service-level-agreement (SLA) has processed, SLA definitions, failover-order rules, failover arbitration, and primary server status.

Use *connection_manager_name* to display information for a specific Connection Manager instance. If *connection_manager_name* is not specified, **onstat -g cmsm** displays information about all Connection Manager instances that are connected to the database server.

Example output 1: Output for a specific Connection Manager

In the following example, **onstat -g cmsm connection_manager_1** is run on the primary server of **my_cluster_1**.

```
Unified Connection Manager: connection_manager_1                Hostname: my_host_1

CLUSTER          my_cluster_1       LOCAL
   SLA           Connections    Service/Protocol    Rule
   oltp_1                 35      19910/onsoctcp    DBSERVERS=primary
   report_1               33      19810/onsoctcp    DBSERVERS=(HDR,SDS,RSS)

   Failover Arbitrator: Active Arbitrator, Primary is up
   ORDER=SDS,HDR,RSS PRIORITY=1
```

The command displays output for **connection_manager_1**. **connection_manager_1** manages a CLUSTER connection unit, and is the active failover arbiter.

Example output 2: Output for a high-availability cluster

In the following example, **onstat -g cmsm** is run on the primary server of **my_cluster_2**.

```
Unified Connection Manager: connection_manager_2                Hostname: my_host_2

CLUSTER          my_cluster_2       LOCAL
   SLA           Connections    Service/Protocol    Rule
   sla_1                1535      19910/onsoctcp    DBSERVERS=primary
   sla_2                2133      19810/onsoctcp    DBSERVERS=(HDR,SDS,RSS)

        Failover Arbitrator: Active Arbitrator, Primary is up
        ORDER=SDS,HDR,RSS PRIORITY=1
```

```
CLUSTER         my_cluster_3
   SLA          Connections   Service/Protocol   Rule
   sla_3                730     19930/onsoctcp   DBSERVERS=primary
   sla_4                901     19830/onsoctcp   DBSERVERS=(HDR,SDS,RSS)

   Failover Arbitrator: Active Arbitrator, Primary is up
   ORDER=SDS,HDR,RSS PRIORITY=1

Unified Connection Manager: connection_manager_3               Hostname: my_host_3

CLUSTER         my_cluster_2      LOCAL
   SLA          Connections   Service/Protocol   Rule
   sla_5                614     19920/onsoctcp   DBSERVERS=primary
   sla_6                483     19820/onsoctcp   DBSERVERS=(HDR,SDS,RSS)

   Failover Arbitrator: Failover is enabled
   ORDER=SDS,HDR,RSS PRIORITY=2

CLUSTER         my_cluster_3
   SLA          Connections   Service/Protocol   Rule
   sla_7                678     19940/onsoctcp   DBSERVERS=primary
   sla_8                270     19840/onsoctcp   DBSERVERS=(HDR,SDS,RSS)

        Failover Arbitrator: Failover is enabled
        ORDER=SDS,HDR,RSS PRIORITY=2
```

The command displays output for the two Connection Managers that connect to the primary server of the cluster. **connection_manager_2** and **connection_manager_3** are installed on separate hosts, and together they manage two CLUSTER connection units. **connection_manager_2** is the active failover arbiter for both CLUSTER connection units.

Example 3: Output for a replicate set

In the following example, **onstat -g cmsm** is run on a replicate server in **my_replicate_set_1**.

```
Unified Connection Manager: connection_manager_4               Hostname: my_host_4

REPLSET         my_replicate_set_1
   SLA          Connections   Service/Protocol   Rule
   sla_1                160     19810/onsoctcp   DBSERVERS=ANY

Unified Connection Manager: connection_manager_5               Hostname: my_host_5

REPLSET         my_replicate_set_1
   SLA          Connections   Service/Protocol   Rule
   sla_2                240     19820/onsoctcp   DBSERVERS=ANY
```

The command displays output for the two Connection Managers that connect to the replicate server. **connection_manager_4** and **connection_manager_5** are installed on separate hosts, and together they manage the replication servers.

Example 4: Output for a grid

In the following example, **onstat -g cmsm** is run on a node of **my_grid_1**.

```
Unified Connection Manager: connection_manager_6               Hostname: my_host_6

GRID    my_grid_1
   SLA          Connections   Service/Protocol   Rule
   sla_1                456     19830/onsoctcp   DBSERVERS=(group_name_1,group_name_2) POLICY=FAILURE

Unified Connection Manager: connection_manager_7               Hostname: my_host_7

GRID    my_grid_1
   SLA          Connections   Service/Protocol   Rule
   sla_2                785     19840/onsoctcp   DBSERVERS=(group_name_1,group_name_2) POLICY=FAILURE
```

The command displays output for the two Connection Managers that connect to the grid. The command displays output for the two Connection Managers that connect to the node. **connection_manager_6** and **connection_manager_7** are installed on separate hosts, and together they manage the grid.

Example 5: Output for a server set

In the following example, **onstat -g cmsm** is run on a stand-alone server in the server set.

```
Unified Connection Manager: connection_manager_8                    Hostname: my_host_8

SERVERSET       server_1,server_2
   SLA          Connections   Service/Protocol   Rule
   sla_1                 63     19810/onsoctcp   DBSERVERS=(server_1,server_2) POLICY=ROUNDROBIN

Unified Connection Manager: connection_manager_9                    Hostname: my_host_9

SERVERSET       server_1,server_2
   SLA          Connections   Service/Protocol   Rule
   sla_2                 63     19810/onsoctcp   DBSERVERS=(server_1,server_2) POLICY=ROUNDROBIN
```

The command displays output for the two Connection Managers that connect to the server set. **connection_manager_8** and **connection_manager_9** are installed on separate hosts, and together they manage the server set.

Output description

The output of the **onstat -g cmsm** command contains sections for each Connection Manager. Each section displays the Connection Manager instance name and host name, followed by subsections that contain information on each connection unit the Connection Manager connects to.

Unified Connection Manager
: The name of the Connection Manager instance.

Hostname
: The name of the Connection Manager's host.

SLA
: The names of service level agreements, as defined in the Connection Manager's configuration file.

Connections
: The numbers of connections each SLA processed since the Connection Manager started.

Service/Protocol
: The port number or service name that is associated with the SLA, followed by the connection protocol type.

Rule
: The SLA definition.

Failover Arbitrator:
: Specifies whether the Connection Manager is the active failover arbiter, if the primary server is active, and if failover is enabled. Displays only for CLUSTER connection units.

ORDER
: Specifies the failover order for a cluster. Displays only for CLUSTER connection units.

PRIORITY
: Specifies the priority of the connection between the Connection Manager and the primary server of a cluster. Displays only for CLUSTER connection units.

Related information:
Connection management through the Connection Manager
Monitoring and troubleshooting connection management

onstat -g con command: Print condition and thread information

Use the **onstat -g con** command to display information about conditions and the threads that are waiting for the conditions.

Syntax:

```
>>-onstat-- -g--con-------------------------------------------><
```

Example output

Figure 1. **onstat -g con** command output

```
Conditions with waiters:
cid      addr             name             waiter   waittime
271      c63d930          netnorm          1511     6550
```

Output description

cid
: Condition identifier

addr
: Condition control block address

name
: Name of condition the thread is waiting on

waiter
: ID of thread waiting on condition

waittime
: Time, in seconds, thread has been waiting on this condition

onstat -g cpu: Print runtime statistics

Use the **onstat -g cpu** command to display information about runtime statistics for each thread that is running in the server.

Syntax:

```
>>-onstat-- -g--cpu-------------------------------------------><
```

Example output

Figure 1. **onstat -g cpu** command output

```
onstat -g cpu

Thread CPU Info:
tid     name              vp       Last Run            CPU Time     #scheds    status
2       lio vp 0          3lio*    07/18 08:35:35        0.0000          1    IO Idle
3       pio vp 0          4pio*    07/18 08:35:36        0.0102          2    IO Idle
4       aio vp 0          5aio*    07/18 08:35:47        0.6876         68    IO Idle
5       msc vp 0          6msc*    07/18 11:47:24        0.0935         14    IO Idle
6       main_loop()       1cpu*    07/18 15:02:43        2.9365      23350    sleeping secs: 1
7       soctcppoll        7soc*    07/18 08:35:40        0.1150          1    running
8       soctcpio          8soc*    07/18 08:35:40        0.0037          1    running
9       soctcplst         1cpu*    07/18 11:47:24        0.1106         10    sleeping forever
10      soctcplst         1cpu*    07/18 08:35:40        0.0103          6    sleeping forever
11      flush_sub(0)      1cpu*    07/18 15:02:43        0.0403      23252    sleeping secs: 1
12      flush_sub(1)      1cpu*    07/18 15:02:43        0.0423      23169    sleeping secs: 1
13      flush_sub(2)      1cpu*    07/18 15:02:43        0.0470      23169    sleeping secs: 1
14      flush_sub(3)      1cpu*    07/18 15:02:43        0.0407      23169    sleeping secs: 1
15      flush_sub(4)      1cpu*    07/18 15:02:43        0.0307      23169    sleeping secs: 1
16      flush_sub(5)      1cpu*    07/18 15:02:43        0.0323      23169    sleeping secs: 1
17      flush_sub(6)      1cpu*    07/18 15:02:43        0.0299      23169    sleeping secs: 1
18      flush_sub(7)      1cpu*    07/18 15:02:43        0.0314      23169    sleeping secs: 1
19      kaio              1cpu*    07/18 14:56:42        1.4560    2375587    IO Idle
20      aslogflush        1cpu*    07/18 15:02:43        0.0657      23166    sleeping secs: 1
21      btscanner_0       1cpu*    07/18 15:00:53        0.0484        784    sleeping secs: 61
37      onmode_mon        1cpu*    07/18 15:02:43        0.3467      23165    sleeping secs: 1
43      dbScheduler       1cpu*    07/18 14:58:14        1.6613        320    sleeping secs: 31
44      dbWorker1         1cpu*    07/18 13:48:10        0.4264        399    sleeping forever
45      dbWorker2         1cpu*    07/18 14:48:11        1.9346       2936    sleeping forever
94      bf_priosweep()    1cpu*    07/18 15:01:42        0.0431         77    cond wait  bp_cond
```

Output description

tid
: The ID of the thread

name
: The name of the thread

vp
: The ID of the virtual processor in which the thread is running

Last Run
: The timestamp when the thread last ran

CPU Time
: The time taken until now by the thread

#scheds
: The number of times the thread was scheduled to run

status
: The status of the thread. Possible status values are:

- cond wait
- IO Idle
- join wait
- mutex wait
- ready
- sleeping
- terminated
- running
- yield

onstat -g dbc command: Print dbScheduler and dbWorker thread statistics

Use the **onstat -g dbc** command to display statistics about the Scheduler tasks that are currently running, which are handled by dbWorker threads, or scheduled to be run, which are handled by the dbScheduler thread.

Syntax:

```
>>-onstat-- -g--dbc-----------------------------------------><
```

Example output

Figure 1. **onstat -g dbc** command output

```
Worker Thread(0)    46fa6f10
====================================
Task:                47430c18
Task Name:           mon_config_startup
Task ID:             3
Task Type:           STARTUP SENSOR
Last Error
    Number              -310
    Message             Table (informix.mon_onconfig) already exists in database.
```

```
    Time                  09/11/2007 11:41
    Task Name             mon_config_startup

Task Execution:      onconfig_save_diffs

WORKER PROFILE
    Total Jobs Executed            10
    Sensors Executed                8
    Tasks Executed                  2
    Purge Requests                  8
    Rows Purged                     0

Worker Thread(1)    46fa6f80
====================================
Task:                4729fc18
Task Name:           mon_sysenv
Task ID:             4
Task Type:           STARTUP SENSOR
Task Execution:      insert into mon_sysenv select 1, env_name, env_value FROM
                     sysmaster:sysenv


WORKER PROFILE
    Total Jobs Executed             3
    Sensors Executed                2
    Tasks Executed                  1
    Purge Requests                  2
    Rows Purged                     0


Scheduler Thread    46fa6f80
====================================
Run Queue
    Empty
Run Queue Size         0
Next Task              7
Next Task Waittime    57
```

Output description

Worker Thread
: Address of the worker thread in shared memory

Task
: Name of the last executed task

Task ID
: The task ID from the **tk_id** column in the **sysadmin:ph_task** table for this task

Task Type
: Type of the task

Last Error
: Error number, error message, time (in seconds), and task name from the last error the dbWorker thread encountered. It could be from the previously executed task or from a task executed days ago.

Task Execution
: SQL statement or SPL procedure or routine executed as part of the task

WORKER PROFILE
: The dbWorker thread profile data shows the total jobs executed, number of sensors executed, number of tasks executed, number of purge requests, and the number of rows purged from the result tables for all sensors executed by this dbWorker thread.

Scheduler Thread
: Address of the scheduler thread in shared memory

Run Queue
: The task ID for the next scheduled task. If no task is scheduled, the value is `Empty`.

Run Queue Size
: The number of tasks that are waiting to be executed by the dbWorker thread

Next Task
: The task ID of the next task that will be scheduled to be executed

Next Task Waittime
: The number of seconds before the `Next Task` will be scheduled for execution

Related reference:
scheduler argument: Stop or start the scheduler (SQL administration API)
Related information:
Monitor the scheduler

onstat -g defragment command: Print defragment partition extents

Use the **onstat -g defragment** command to display information about the active requests to defragment partition extents.

Syntax:

```
>>-onstat-- -g--defragment-------------------------------------><
```

Example output

Figure 1. **onstat -g defragment** command output

```
Defrag info
id  table name                      tid  dbsnum  partnum  status                substatus  errnum
15  stores_demo:informix.stdtab2    49   2       2097155  SEARCHING_FOR_EXTENT  0          0
```

Note: This command displays information about defragment requests that are active. If there are no active defragment requests, only the column headings are returned.

Output description

id
: The ID of the defragment request.

table name
: The fully-qualified name of the table that is being defragmented.

tid
: The thread ID.

dbsnum
: The dbspace number that is being defragmented.

partnum
: The partition number that is being defragmented.

status
: - SEARCHING_FOR_EXTENT
  - MERGING_EXTENTS
  - DEFRAG_COMPLETED
  - DEFRAG_FAILED

substatus
: The detailed status number, if any.

errnum
: The last error number returned from the defragmentation request.

onstat -g dic command: Print table information

Use the **onstat -g dic** command to display a line of information about each table that is cached in the shared-memory dictionary. If you specify a table name, this command prints internal SQL information about that particular table.

Syntax:

```
>>-onstat-- -g--dic-----------------------------------------><
```

Example output

Figure 1. **onstat -g dic** command output

```
Dictionary Cache:  Number of lists: 31, Maximum list size: 10
list#  size  refcnt  dirty?  heapptr        table name
---------------------------------------------------------
   1     3      1      no     14b5d890   wbe@oninit_shm:informix.t0010url
                1      no     14cbb820   wbe@oninit_shm:informix.t9051themeval
                0      no     14b63c20   wbe@oninit_shm:informix.t0060hits

   2     2      0      no     14b97420   wbe@oninit_shm:informix.t0120import
                1      no     14b6c820   wbe@oninit_shm:informix.t9110domain

   3     3      0      no     14bce020   wbe@oninit_shm:informix.t0150url
                0      no     14d3d820   contact@oninit_shm:informix.wbtags
                0      no     14c87420   wbe@oninit_shm:informix.wbtags

   4     1      0      no     14b7a420   drug@oninit_shm:abcdef.product      ..
Total number of dictionary entries: 36
```

Output description

list#
: Data dictionary hash chain ID

size
: Number of entries in this hash

refcnt
: Number of SQL statements currently referencing one of the cache entries.

dirty?
: Whether the entry has been modified since last written to disk.

heapptr
: Address for the heap used to store this table

table name
Name of table in cache

onstat -g dis command: Print database server information

Use the **onstat -g dis** command to display a list of database servers, the status of each server, and information about each server, including the location of the **INFORMIXDIR** directory, **sqlhosts** file, and ONCONFIG file. You can use this command in any database server mode, including offline.

Syntax:

```
>>-onstat-- -g--dis-----------------------------------------><
```

Example output

Figure 1. **onstat -g dis** command output

```
There are 2 servers found
Server         : ol_tuxedo
Server Number  : 53
Server Type    : IDS
Server Status  : Up
Server Version: IBM Informix Version 11.50.UC1
Shared Memory  : 0xa000000
INFORMIXDIR    : /local1/engines/ol_tuxedo/dist
ONCONFIG       : /local1/engines/ol_tuxedo/dist/etc/onconfig.ol_tuxedo
SQLHOSTS       : /local1/engines/ol_tuxedo/dist/etc/sqlhosts
Host           : avocet

Server         : ol_9next
Server Number  : 0
Server Type    : IDS
Server Status  : Down
Server Version:
Shared Memory  : 0
INFORMIXDIR    : /local1/engines/ol_9next/dist
ONCONFIG       :
SQLHOSTS       :
Host           :
```

Output description

Server
Server name
Server Number
Number of the server.
Server Type
Type of server
Server Status
Up means that the server is online, Down means that the server is offline
Server Version
Version of the server
Shared Memory
Location of the shared memory address
INFORMIXDIR
Location of the **$INFORMIXDIR/** directory on UNIX and in the **%INFORMIXDIR%** directory on Windows.
ONCONFIG
Location of the ONCONFIG file
SQLHOSTS
Location of the **sqlhosts** file
Host
Host name of the server

onstat -g dll command: Print dynamic link library file list

Use the **onstat -g dll** command to display a list of and the status of dynamic link library (DLL) files that were loaded.

Syntax:

```
>>-onstat-- -g--dll-----------------------------------------><
```

Example output

The output displays the names of the library files only one time each process group. The flags indicate if the library was loaded when the server was started.
Figure 1. **onstat -g dll** command output

```
addr             slot   vp baseaddr         flags    filename
0x4af55310       15     1  0x2a985e3000     PM       /finance/jeffzhang/mylib.udr
0x4b6f2310              2  0x2a985e3000
0x4b71b310              3  0x2a985e3000
0x4c09f310       16     1  0x2a985e3000     M        /deptxyz/udrs/geodetic.bld
0x4c0c0310              2  0x2a985e3000
0x4c0f1310              3  0x2a985e3000
0x4c112310       17     1  0x7a138e9000              /home/informix/extend/blade.so
0x4c133310              2  0x3a421e1000
0x4c133310              3  0x3a421e1000
```

Output description

addr
: Address of the DLL file

slot
: Slot number entry in the library table

vp
: ID of the virtual processor

baseaddr
: Base address of the shared library

flags

- M indicates that the thread calling the UDR can migrate from one CPU virtual processor to another CPU virtual processor.
- P indicates that the shared library was loaded when the database server was started.

filename
: Name of the DLL file

Related reference:
PRELOAD_DLL_FILE configuration parameter

onstat -g dmp command: Print raw memory

Use the **onstat -g dmp** command to display information about raw memory at a given address for a number of given bytes.

Syntax:

```
>>-onstat-- -g--dmp--address--length---------------------------><
```

Each address and length must be within the allocated memory shown from **onstat -g seg** output. The address specified can be in decimal or hexadecimal format. Hexadecimal addresses must begin with **0x**. You can specify the address in decimal, but doing so requires converting the memory shown from **onstat -g seg** to decimal before using it as a command line argument.

Example output

Figure 1. **onstat -g dmp** command output

```
%onstat -g dmp 0x700000011a19d48 100

address           bytes in mem
0700000011a19d48: 07000000 118e0fa8 07000000 11942b40 ........ ......+@
0700000011a19d58: 07000000 10137120 00000000 00000000 ......q  ........
0700000011a19d68: 00000000 00000000 00000000 00000000 ........ ........
0700000011a19d78: 07000000 11a19d48 07000000 11a19d48 .......H .......H
0700000011a19d88: 00000000 00000000 00000000 00000000 ........ ........
0700000011a19d98 *
0700000011a19da8: 00000000 ....
```

Output description

address
: Memory address of the raw memory.

bytes in mem
: Hexadecimal and ASCII representations of the memory contents.

Output from the command is divided into three columns: memory address, hexadecimal values for the bytes in memory, and the ASCII representation of the bytes in memory. The bytes in memory (middle) section displays the first 16 bytes of memory starting at the address specified on the command line. The third column shows the ASCII representation of the hexadecimal data. Periods are displayed for all hexadecimal values that do not have an ASCII character equivalent. ASCII values are shown in order to make searching for plain text easier.
In the example output shown, the fifth line of data displays zeros and the sixth line contains an asterisk. The asterisk indicates an unknown number of repetitions of the previous line, which means that there is no more data after the fourth line.

onstat -g dri command: Print high-availability data replication information

Use the **onstat -g dri** command, either alone or with the **ckpt** or **que** options, to print information about high-availability data replication statistics on the current server.

Use the **onstat -g dri** command to print information about HDR server states and HDR-related configuration parameters.

Syntax:

```
>>-onstat-- -g--dri--+------+---------------------------------><
                    +-ckpt-+
                    '-que--'
```

Example output and output description for onstat -g dri

Figure 1. **onstat -g dri** command output

```
Data Replication at 0x4d676028:
  Type            State         Paired server        Last DR CKPT (id/pg)    Supports Proxy Writes
  primary         on            my_server                  4 / 5              NA

  DRINTERVAL   5
  DRTIMEOUT    30
  DRAUTO       3
  DRLOSTFOUND  /etc/dr.lostfound
  DRIDXAUTO    0
  ENCRYPT_HDR  0  Backlog      0
  Last Send    2013/12/11 16:39:48
  Last Receive 2013/12/11 16:39:48
  Last Ping    2013/12/11 16:39:44
  Last log page applied(log id,page): 4,6
```

Type
: Current® type of server: primary, secondary, or standard

State
: on or off

Paired server
: Name of the primary or secondary server that this server is paired with

Last DR CKPT
: Last checkpoint ID and page

Supports Proxy Writes
: Displays whether the server is configured to allow secondary server updates. **Y** = supports secondary server updates, **N** = does not support secondary server updates.

DRINTERVAL
: The value of the configuration parameter in the onconfig file.

DRTIMEOUT
: The value of the configuration parameter in the onconfig file.

DRAUTO
: The value of the configuration parameter in the onconfig file.

DRLOSTFOUND
: The value of the configuration parameter in the onconfig file.

DRIDXAUTO
: The value of the configuration parameter in the onconfig file.

ENCRYPT_HDR
: The value of the configuration parameter in the onconfig file.

Backlog
: Number of log pages in the HDR data replication buffer that are not yet sent to the HDR secondary server

Last Send
: The time that the last message was sent to the peer node

Last Receive
: The time that the last message was received from the peer node

Last Ping
: The time of the last ping

Last log page applied(log id,page)
: The log ID and page number of the last applied log

Example output and output description for onstat -g dri ckpt

Use the **onstat -g dri ckpt** command to print information about nonblocking checkpoints in HDR servers.

Figure 2. **onstat -g dri ckpt** command output

```
Data Replication:
  Type            State         Paired server        Last DR CKPT (id/pg)   Supports Proxy Writes
  primary         on            BB_1                       554 / 558       Y

  DRINTERVAL   30
  DRTIMEOUT    30
```

```
  DRAUTO       0
  DRLOSTFOUND  /vobs/tristarm/sqldist/etc/dr.lostfound
  DRIDXAUTO    0
  ENCRYPT_HDR  0
DR Checkpoint processing:
  Save State               N
  Pages Saved              0
  Save Area                none
  Received log id, page    17,68
  Saved log id, page       0,0
  Drain log id, page       0,0
  Processed log id, page   17,68
  Pending checkpoints      0
```

Save State
: **B** (buffering) when the server is adding logs to the staging area
: **D** (draining) when the server is removing logs from the staging area
: **N** (normal) when the server is operating normally, meaning that no logs are saved

Pages Saved
: Displays the number of log pages saved in the staging area that have yet to be applied.

Save Area
: Displays the location of the staged log files.

Received log id, page
: Displays the last log ID and page that were received from the primary server.

Processed log id, page
: Displays the last log ID and page that are queued to the recovery pipeline.

Saved log id, page
: Displays the last log ID and page that was stored in the staging area (if stage state is either **B** or **D**).

Drain log id, page
: Displays the last log ID and page that were removed from the staging area.

Pending checkpoints
: Displays the number of checkpoints that are staged but not yet applied.

Pending ckpt log id, page
: Displays the position of any pending checkpoint records.

Example output and output description for onstat -g dri que

Use the **onstat -g dri que** command to print information that is related to nearly synchronous HDR replication.

Figure 3. **onstat -g dri que** command output

```
        Pending Msg to Send 1
        ACK QUEUE  5199:1256fff
        thread 0x893de6c8 (85) 5199:1258018
        thread 0x893a16b8 (83) 5199:1258048
        thread 0x89229968 (72) 5199:1258078
        thread 0x89381508 (82) 5199:12580a8
        thread 0x87e81658 (69) 5199:12580d8
        thread 0x89215968 (71) 5199:1259018
        thread 0x89336bc8 (80) 5199:1259048
        thread 0x89370018 (81) 5199:12590f8
        thread 0x892eb018 (77) 5199:125a018
        thread 0x89308018 (78) 5199:125b018
        thread 0x89290138 (75) 5199:125b048
        thread 0x893c1658 (84) 5199:125c018
        thread 0x891fe8e8 (70) 5199:125c048
        thread 0x89325018 (79) 5199:125d018
        thread 0x893ff738 (86) 5199:125d048
        thread 0x894207a8 (87) 5199:125d078

        Applied QUEUE  5199:1251018
        ----------------------------------------
```

Pending message to send
: The number of unprocessed data replication buffers queued to the **drprsend** thread.

ACK QUEUE
: The log unique value, the page number, and the value 0xfff for the most recently paged log.

thread
: The pointer to the thread-control block (TCB), the thread id in parentheses, and the log sequence number (LSN) of the commit that was performed by that thread

Applied QUEUE
: The LSNs of commits that are waiting for acknowledgement of being received on the HDR secondary.

Related reference:
DRAUTO configuration parameter
DRIDXAUTO configuration parameter
DRINTERVAL configuration parameter
DRLOSTFOUND configuration parameter
DRTIMEOUT configuration parameter
Related information:
Fully synchronous mode for HDR replication
Asynchronous mode for HDR replication
Nearly synchronous mode for HDR replication
Replication of primary-server data to secondary servers
HDR_TXN_SCOPE session environment option

onstat -g dsc command: Print distribution cache information

Use the **onstat -g dsc** command to display information about the distribution cache.

Syntax

```
>>-onstat-- -g--dsc-------------------------------------------><
```

Example output

Figure 1. onstat -g dsc command output

```
Data Distribution Cache:
    Number of lists         : 31
    DS_POOLSIZE             : 127

Distribution Cache Entries:
list id   ref  drop hits       last_access          heap_ptr         distribution name
-------------------------------------------------------------------------------

3    0    0    0    42         2020-05-12 10:36:36  4d6b7838         sysadmin:informix.aus_work_info.aus_info_tabid
3    0    0    0    192        2020-05-12 10:36:36  45c264d8         sysadmin:informix.aus_work_dist.aus_dist_tabid

6    0    0    0    204        2020-05-12 10:36:36  45c268d8         sysadmin:informix.aus_work_icols.aus_icols_tabid


Total number of distribution entries: 58
    Number of entries in use        : 0
```

Output description

Number of lists
: Number of lists in the distribution cache

DS_POOLSIZE
: Number of entries that can be cached at one time

list
: Distribution cache hash chain ID

id
: Number of hash entries

ref
: Number of statements that reference a cache entry

drop
: Whether this entry was dropped after it was added to the cache

hits
: The number of times the cache entry is accessed.

last_access
: The time at which the cache entry was last accessed.

heap_ptr
: Heap address that is used to store this entry

distribution name
: The name of the distribution in the cache

Total number of distribution entries
: Number of entries in the distribution cache

Number of entries in use
: Number of entries that are being used

Related reference:
DS_HASHSIZE configuration parameter
DS_POOLSIZE configuration parameter

onstat -g dsk command: Print the progress of the currently running compression operation

Use the **onstat -g dsk** command to print information that shows the progress of currently running compression operations, such as compress, repack, and shrink.

Syntax:

```
>>-onstat-- -g -dsk-------------------------------------------><
```

Example output

Figure 1. **onstat -g dsk** command output for a compress operation

```
                    Processed        Remaining Duration Remaining
Partnum    OP       Rows       Blobs Rows      Time(s)  Time(s)   Table Name
400002     Compress 6325       1752  1497      00:00:00 00:00:00 db:sl:t1
```

Figure 2. **onstat -g dsk** command output for a repack operation

```
                          Processed        Remaining Duration Remaining
Partnum   OP     Pass     Rows       Blobs Rows      Time(s)  Time(s)   Table Name
400002    Repack 1        6325       1752  1497      00:00:00 00:00:00 db:sl:t1
```

Output description

partnum
: Partition number of the table or fragment

OP
: Compression operation, such as compress, repack, or shrink.

Pass
: For repack operations, 1 indicates the first pass of reading the rows, and 2 indicates the second pass.

Processed Rows
: Number of rows that are processed so far for the specified operation

Blobs
: The number of simple large objects that were operated on

Remaining Rows
: The number of remaining rows to process. For repack operations, the number of rows that remain in the current pass.

Duration Time(s)
: The amount of time since the beginning of the operation

Remaining Time(s)
: Approximate amount of remaining time for the operation. For repack operations, the amount of time that remains for the current pass.

onstat -g env command: Print environment variable values

Use the **onstat -g env** command to display the values of the environment variables that the database server currently uses.

Syntax:

```
>>-onstat-- -g--env--+------------------------+---------------><
                    +-all--------------------+
                    +-variable---------------+
                    '-sessionid--+----------+-'
                                 '-variable-'
```

You can specify one of the following invocations.

Invocation	Explanation
onstat -g env	Displays the settings of environment variables when the database server was started Does not display environment variables that have not been set explicitly.
onstat -g env all	Displays the settings used by all sessions This display is the same as the output of **onstat -g env** and **onstat -g env*sessionid*** iteratively on all current sessions.
onstat -g env *variable*	Displays the default value of the specified environment variable This ***variable*** argument eliminates the need to pipe the output to **grep** (or some other utility) to locate an environment variable among many that might be set.
onstat -g env *sessionid*	Displays the settings that a specific session uses. This display includes the following values: • Set in the environment of the session • Assigned by the database server, as **onstat -g env** displays
onstat -g env *sessionid variable*	Displays the value of the specified environment variable that the specified session uses The ***sessionid*** and ***variable*** arguments eliminate the need to pipe the output to **grep** (or some other utility) to locate an environment variable among many that might be set.

The **onstat -g env** command displays the current setting of an environment variable and the complete list of values each time the variable was set in the environment. For example, if PDQPRIORITY is set to `10` in the **.informix.rc** file and set to `55` in the shell environment, **onstat -g env** command displays both values.

However, if you change the PDQPRIORITY with the **onmode -q pdqpriority *sessionid*** command, the **onstat -g env** command does not display the new value for the session. The **onstat -g env** command displays only the values of environment variables set in the environment. It does not display values modified while the session is running.

You might want to display the values of environment variables in the following situations:

- The database server instance has been up for months, and you cannot remember the setting of an environment variable (such as the server locale setting **SERVER_LOCALE**).
- You want to display the complete list of values for an environment variable to identify when an environment variable has been set in multiple places.
- Environment files on disk might have changed or been lost in the interim.

- A support engineer wants to know settings of specific environment variables.

Example output

The following figure shows the output for the **onstat -g env** command.
Figure 1. **onstat -g env** command output

```
Variable             Value [values-list]
DBDATE               DMY4/
DBDELIMITER          |
DBPATH               .
DBPRINT              lp -s
DBTEMP               /tmp
INFORMIXDIR          /build2/11.50/tristarm/sqldist
                     [/build2/11.50/tristarm/sqldist]
                     [/usr/informix]
INFORMIXSERVER       parata1150
INFORMIXTERM         termcap
LANG                 C
LC_COLLATE           C
LC_CTYPE             C
LC_MONETARY          C
LC_NUMERIC           C
LC_TIME              C
LD_LIBRARY_PATH      /usr/openwin/lib:/lib:/usr/lib
LKNOTIFY             yes
LOCKDOWN             no
NODEFDAC             no
NON_M6_ATTRS_OK      1
PATH                 /build2/11.50/tristarm/sqldist/bin:.:
                     /root/bin:/opt/SUNWspro/bin:/usr/ccs/bin:
                     /usr/openwin/bin:/usr/sbin:/usr/bin:/usr
                     /local/bin
SERVER_LOCALE        en_US.819
SHELL                /bin/ksh
SINGLELEVEL          no
SUBQCACHESZ          10
TBCONFIG             onconfig
TERM                 xterm
                     [xterm]
                     [dumb]
TERMCAP              /etc/termcap
TZ                   GB
```


onstat -g ffr command: Print free fragments

Use the **onstat -g ffr** command to display information about the free memory fragments for a specified session or shared-memory pool.

This command requires an additional argument to specify either a pool name or session ID whose memory pool information is to be displayed. Each session is allocated a memory pool with the same name as the session ID. Use the **onstat -g mem** command to identify the pool name and the **onstat -g ses** command to identify the session ID.

Syntax:

```
>>-onstat-- -g--ffr--+-pool name-+-----------------------------><
                     '-sessionid-'
```

Example output

Figure 1. **onstat -g ffr aio** command output

```
Free lists for pool name aio:
addr      size      idx
165dcfa0  96        10
1659cf68  152       17
165b2f20  224       26
165c7f20  224       26
1666ec38  968       79
149f2ba0  1120      84
```

Output description

addr (hexadecimal)
: Memory address of the pool fragment.

size (decimal)
: Size, in bytes, of the pool fragment.

idx (decimal)
: For internal use. Index in the array of free list pointers.

onstat -g glo command: Print global multithreading information

Use the **onstat -g glo** command to display global information about multithreading, information about each virtual processor that is running, and cumulative statistics for each virtual-processor class. This information includes CPU use information about the virtual processors, the total number of sessions, and other multithreading global counters.

Syntax:

```
>>-onstat-- -g--glo-------------------------------------------><
```

Example output

Figure 1. onstat -g glo command output

```
MT global info:
sessions threads  vps      lngspins time
0        23       14       0        142

          sched calls     thread switches yield 0   yield n   yield forever
total:    85240           70451           16956     868       37319
per sec:  0               0               0         0         0

Virtual processor summary:
 class       vps       usercpu   syscpu    total
 cpu         1         92.12     0.59      92.71
 aio         1         0.05      0.08      0.13
 lio         1         0.00      0.00      0.00
 pio         1         0.00      0.00      0.00
 adm         1         0.00      0.01      0.01
 soc         4         0.01      0.01      0.02
 msc         1         0.00      0.00      0.00
 jvp         1         0.00      0.00      0.00
 fifo        1         0.00      0.00      0.00
 nyevp       1         0.00      0.00      0.00
 yevp        1         0.00      0.00      0.00
 total       14        92.18     0.69      92.87

Individual virtual processors:
 vp     pid       class       usercpu   syscpu    total     Thread    Eff
 1      26328     cpu         92.12     0.59      92.71     122.65    75%
 2      26330     adm         0.00      0.01      0.01      0.00       0%
 3      26331     lio         0.00      0.00      0.00      0.00       0%
 4      26332     pio         0.00      0.00      0.00      0.00       0%
 5      26333     aio         0.05      0.08      0.13      0.28      45%
 6      26334     msc         0.00      0.00      0.00      0.19       0%
 7      26335     fifo        0.00      0.00      0.00      0.00       0%
 8      26336     nyevp       0.00      0.00      0.00      0.00       0%
 9      26337     yevp        0.00      0.00      0.00      0.00       0%
 10     26338     jvp         0.00      0.00      0.00      0.00       0%
 11     26339     soc         0.00      0.00      0.00      NA         NA
 12     26340     soc         0.00      0.00      0.00      NA         NA
 13     26341     soc         0.01      0.01      0.02      NA         NA
 14     26342     soc         0.00      0.00      0.00      NA         NA
                  tot         92.18     0.69      92.87
```

Output description

The following table explains each column in the global information section of the example output.

Table 1. Description of the columns in the virtual processor summary

Column name	Description
sessions	The number of sessions
threads	The total number of threads
vps	The total number of virtual processors
lngspins	The number of times a thread had to spin more than 10,000 times to acquire a latch on a resource
time	The number of seconds over which the statistics were gathered. Statistics start when the server starts or the statistics are reset by running the **onstat -z** command.
sched calls	The total number of scheduled calls.
thread switches	The total number of switches from one thread to another.
yield	Statistics on thread yields, which occur when a thread can no longer continue its task until some condition occurs

The following table explains each column in the virtual processor summary section of the example output.

Table 2. Description of the columns in the virtual processor summary

Column name	Description
class	The type of virtual processor.
vps	The number of instances of the class of virtual processor.
usercpu	The total user time, in seconds, that the class of virtual processor spent running on the CPU.

Column name	Description
syscpu	The total system time, in seconds, the class of virtual processor spent running on the CPU.
total	The total CPU time for the virtual processor class, as the sum of the user time plus the system time.

The following table explains each column in the individual virtual processors section of the example output.

Table 3. Description of the columns for the individual virtual processors

Column name	Description
vp	The virtual processor number. On Windows, the values are thread IDs.
pid	The Process ID of the **oninit** process.
class	The type of virtual processor.
usercpu	The total user time, in seconds, that the virtual processor spent running on the CPU.
syscpu	The total system time, in seconds, that the virtual processor spent running on the CPU.
total	The total CPU time for the virtual processor, as the sum of the user time plus the system time.
Thread	The total time the threads ran on the virtual processor.
Eff	Efficiency. The ratio of the total CPU time to the total time the threads ran on the virtual processor.

onstat -g his command: Print SQL trace information

Use the **onstat -g his** command to display SQL trace information from the collection of **syssqltrace** tables (syssqltrace, syssqltrace_info, syssqltrace_hvar and syssqltrace_itr) in the **sysmaster** database.

The **level** setting of the SQLTRACE configuration parameter affects what SQL trace information is stored and displayed by the set of **syssqltrace** tables, and what information **onstat -g his** displays. Each row of the **syssqltrace** table describes a previously executed SQL statement. By default, only the DBSA can view the **syssqltrace** information from the **onstat -g his** command. However, when the UNSECURE_ONSTAT configuration parameter is set to `1`, all users can view this information.

Syntax:

```
>>-onstat-- -g--his-------------------------------------------><
```

Example output

The content of the output depends on the trace settings.

The **Statement history** section in the output provides information about the current settings for tracing.

```
Statement history:

Trace Level                 Low
Trace Mode                Global
Number of traces            1000
Current Stmt ID                2
Trace Buffer size           2008
Duration of buffer           293 Seconds
Trace Flags           0x00001611
Control Block         0x4c2f0028
```

The following table describes this output:

Information	Description
Trace Level	Amount of information traced. Valid values are LOW, MED, HIGH, and OFF.
Trace Mode	Type of tracing performed. Global refers to all users on the system User refers to only those users who have tracing enabled by an SQL administration API function.
Number of traces	The number of SQL statements that are being traced. This is the value set in your onconfig file unless the ntraces parameter is changed dynamically through SQL Administration API functions. The range is 500 to 2147483647. If you have 100,000 trace buffers and your organization runs 1000 SQL statements a second, and are tracing all of the statements, then the buffers would last for 100 seconds before they would begin being overwritten.
Current® Stmt ID	The ID for the current SQL statement. Each statement being traced gets a unique ID.
Trace Buffer size	The amount of data each trace buffer will capture, in bytes. If you set the size to 2KB, but have an SQL statement that is 12KB, the statement is truncated by at least 10KB. More data might be truncated, depending on what else is being traced.
Duration of buffer	The amount of time, in seconds, that the trace data in the current trace buffer spans. This is not how long the **sqltrace** feature has been running. In the above example **Duration of buffer** is 293 seconds which indicates the number of seconds between the first and last SQL statement that are traced.
Trace Flags	The current SQL trace flags that are set.
Control Block	The memory address of the SQL trace control block.

The information displayed below is repeated one time for each time a statement was run. In this example there are two variables being called.

```
Statement # 2:      @ 0x4c2f3028

Database:         sysmaster
```

```
Statement text:
  select count(*) from systables,syscolumns where systables.tabid > ? and
    systables.nrows < ?

  SELECT using tables [ systables syscolumns ]
```

The following table describes this output:

Information	Description
Database	The name of the database or part number of the **systables** entry for the database.
Statement text	The statement text for this SQL statement. If the statement is a stored procedure, then the statement text would display the procedure stack trace. The statement text might be truncated if the statement and the numeric statistics are larger than the trace buffer.

```
Iterator/Explain
================
ID    Left  Right   Est Cost   Est Rows   Num Rows   Partnum   Type
 3       0      0         17         42        146   1048579   Index Scan
 4       0      0       5249       2366       2366   1048580   Seq Scan
 2       3      4       5266      99372     345436         0   Nested Join
 1       2      0          1          1          1         0   Group
```

The following table describes this output:

Information	Description
ID	SQL iterator ID
Left	ID of the left input to the iterator
Right	ID of the right input to the iterator
Est Cost	Estimated cost of this iterator
Est Rows	Estimated rows for this iterator
Num Rows	Actual number of rows for this iterator
Partnum	The table or index partition number.
Type	Type of operation

If the SQL statement contains one or more variables, and you are tracing host variables, the **Host Variables** section is included in the output.

```
Host Variables
==============
1 integer    100
2 float      1000.0000000000000000
```

The following table describes this output:

Information	Description
Column 1	The position of the variable in the statement.
Column 2	The data type of the variable.
Column 3	The value of the variable.

```
Statement information:
  Sess_id  User_id  Stmt Type  Finish Time   Run Time   TX Stamp  PDQ
  5        2053     SELECT     01:08:48      0.4247     340a6e9    0
```

The following table describes this output:

Information	Description
Sess_id	The session ID
User_id	The operating system user ID
Stmt Type	The type of SQL statement
Finish Time	The time of day that the SQL statement finished
Run Time	The total amount of time consumed by the virtual processors or threads used to process the statement. For example, if the Finish Time is 1:15:00 and the Run Time is 9 minutes and the start time is not necessarily 1:06:00. There might be multiple virtual processors or threads involved in processing parts of the statement in parallel.
TX Stamp	The time the BEGIN WORK statement was logged in this transaction
PDQ	The SQL statement PDQ level

The **Statement Statistics** section in the output provides specific information about the statement.

```
  Statement Statistics:
  Page       Buffer     Read       Buffer     Page       Buffer     Write
  Read       Read       % Cache    IDX Read   Write      Write      % Cache
  1285       19444      93.39      0          810        17046      95.25

  Lock       Lock       LK Wait    Log        Num        Disk       Memory
  Requests   Waits      Time (S)   Space      Sorts      Sorts      Sorts
  10603      0          0.0000     60.4 KB    0          0          0

  Total      Total      Avg        Max        Avg        I/O Wait   Avg Rows
  Executions Time (S)   Time (S)   Time (S)   IO Wait    Time (S)   Per Sec
  1          30.8660    30.8660    30.8660    0.0141     29.2329    169.8959

  Estimated  Estimated  Actual     SQL        ISAM       Isolation  SQL
```

```
  Cost       Rows       Rows       Error      Error      Level      Memory
  102        1376       5244       0          0          CR         32608
```

Information	Description
Page Read	Number of pages that have been read from disk for this SQL statement
Buffer Read	Number of times a page has been read from the buffer pool and not read from disk for this SQL statement
Read % Cache	Percentage of times the page was read from the buffer pool
Buffer IDX Read	This Currently not implemented
Page Write	Number of pages written to disk
Buffer Write	Number of pages modified and sent back to the buffer pool
Write % Cache	Percentage of time that a page was written to the buffer pool but not to disk
Lock Requests	Total number of locks required by this statement
Lock Waits	Number of times this SQL statement waited on locks
LK Wait Time (S)	Amount of time the statement waited for application locks, in seconds
Log Space	Amount of storage space that the SQL statement used in the logical log
Num Sorts	Total number of sorts used to execute the statement
Disk Sorts	Number of sorts which required disk space to execute the sort for this SQL statement
Memory Sorts	Number of sorts executed which executed entirely in memory for this SQL statement
Total Executions	Total number of times this prepared statement has been executed, or the number of times this cursor has been re-used
Total Time (S)	Total time this prepared statement ran, in seconds
Avg Time (S)	Average time this prepared statement required to execute, in seconds
Max Time (S)	Total time to run the prepared SQL statement, in seconds, excluding any time taken by the application. If you prepare a query then run the query 5 times, each time the query is run a trace is added to the trace buffer. The Max Time is the maximum time any one execution took.
Avg IO Wait	Average amount of time the statement waited for I/O, excluding any asynchronous I/O
I/O Wait Time (S)	Amount of time the statement waited for I/O, excluding any asynchronous I/O, in seconds
Avg Rows Per Sec	Average number of rows a second produced by this statement
Estimated Cost	The query optimizer cost associated with the SQL statement
Estimated Rows	Number of rows returned by the statement, as estimated by the query optimizer
Actual Rows	Number of rows returned for this statement
SQL Error	The SQL error number
ISAM Error	The RSAM or ISAM error number
Isolation Level	Isolation level this statement was run with
SQL Memory	Number of bytes this SQL statement required

For the complete schema of the **syssqltrace** System Monitoring Interface table, see syssqltrace.

For details of setting the SQLTRACE configuration parameter, see SQLTRACE configuration parameter.

Related reference:
SQLTRACE configuration parameter

onstat -g ioa command: Print combined onstat -g information

Use the **onstat -g ioa** command to display combined information from the **onstat -g iob**, **onstat -g iof**, **onstat -g ioq**, and **onstat -g iov** commands.

Syntax

```
>>-onstat-- -g--ioa-------------------------------------------><
```

Example output

```
AIO global info:
   9 aio classes
   9 open files
  64 max global files

AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
 fifo    0     0      0        0        0        0        0
drda_dbg 0     0      0        0        0        0        0
sqli_dbg 0     0      0        0        0        0        0
```

```
  adt    0    0       0        0        0        0        0
  msc    0    0       1      231        0        0        0
  aio    0    0       5    13069    10895        0        0
  pio    0    0       1     1580        0     1580        0
  lio    0    0       1    37900        0    37900        0
  gfd    3    0      87    42115    15806    26309        0
  gfd    4    0       4        5        1        4        0
  gfd    5    0      12       35       22       13        0
  gfd    6    0      11       33       21       12        0
  gfd    7    0       1        4        3        1        0
  gfd    8    0       1        4        3        1        0

AIO I/O vps:
class/vp/id s  io/s totalops dskread dskwrite dskcopy wakeups io/wup errors tempops
 fifo  7  0 i   0.0        0       0        0       0       1  0.0      0        0
  msc  6  0 i   0.0      231       0        0       0     221  1.0      0      231
  aio  5  0 i   0.0    39285   26358    10793       0   37531  1.0      0        5
  aio  9  1 i   0.0     5770    3795     1944       0    5926  1.0      0        0
  aio 10  2 i   0.0     2308     717     1585       0    1953  1.2      0        0
  aio 11  3 i   0.0     1463     166     1295       0    1166  1.3      0        0
  aio 12  4 i   0.0     1219      46     1172       0     943  1.3      0        0
  aio 13  5 i   0.0     1041      34     1007       0     805  1.3      0        0
  aio 15  6 i   0.0      425       2      423       0     438  1.0      0        0
  aio 16  7 i   0.0      342       5      337       0     395  0.9      0        0
  pio  4  0 i   0.0     1580       0     1580       0    1581  1.0      0     1580
  lio  3  0 i   0.0    37900       0    37900       0   29940  1.3      0    37900

AIO global files:
gfd pathname          bytes read     page reads  bytes write    page writes io/s
3   ./rootdbs         85456896       41727       207394816      101267      572.9
        op type     count          avg. time
        seeks       0              N/A
        reads       13975          0.0015
        writes      51815          0.0018
        kaio_reads  0              N/A
        kaio_writes 0              N/A

4   tempsbs.chunk     2048           1           8192           4           113.6
        op type     count          avg. time
        seeks       0              N/A
        reads       1              0.0131
        writes      3              0.0074
        kaio_reads  0              N/A
        kaio_writes 0              N/A

5   sbs1.chunk        45056          22          26624          13          173.4
        op type     count          avg. time
        seeks       0              N/A
        reads       22             0.0063
        writes      6              0.0038
        kaio_reads  0              N/A
        kaio_writes 0              N/A

6   sbs2.chunk        43008          21          24576          12          76.1
        op type     count          avg. time
        seeks       0              N/A
        reads       21             0.0148
        writes      6              0.0072
        kaio_reads  0              N/A
        kaio_writes 0              N/A

7   qhdr.chunk        6144           3           2048           1           550.5
        op type     count          avg. time
        seeks       0              N/A
        reads       3              0.0019
        writes      1              0.0016
        kaio_reads  0              N/A
        kaio_writes 0              N/A

8   ./dbs1            6144           3           2048           1           403.0
        op type     count          avg. time
        seeks       0              N/A
        reads       3              0.0027
        writes      1              0.0018
        kaio_reads  0              N/A
        kaio_writes 0              N/A


AIO big buffer usage summary:
class                reads                                    writes
       pages     ops  pgs/op  holes  hl-ops hls/op      pages    ops  pgs/op
fifo       0       0   0.00       0       0   0.00          0      0   0.00
drda_dbg   0       0   0.00       0       0   0.00          0      0   0.00
sqli_dbg   0       0   0.00       0       0   0.00          0      0   0.00
 kio       0       0   0.00       0       0   0.00          0      0   0.00
 adt       0       0   0.00       0       0   0.00          0      0   0.00
 msc       0       0   0.00       0       0   0.00          0      0   0.00
 aio  228709   20228  11.31    1005     203   4.95     213272  18556  11.49
 pio       0       0   0.00       0       0   0.00      19672   1580  12.45
 lio       0       0   0.00       0       0   0.00      55287  37900   1.46
```

Output description

For a description of each output column, see the individual onstat -g iob command: Print big buffer use summary, onstat -g ioq command: Print I/O queue information, and onstat -g iov command: Print AIO VP statistics commands.

onstat -g iob command: Print big buffer use summary

Use the **onstat -g iob** command to display a summary of big buffer use.

Syntax:

```
>>-onstat-- -g--iob-----------------------------------------><
```

Example output

Figure 1. **onstat -g iob** command output

```
AIO big buffer usage summary:
                reads                                      writes
      pages     ops  pgs/op  holes  hl-ops hls/op      pages     ops  pgs/op
fifo      0       0    0.00      0       0   0.00          0       0    0.00
 kio      0       0    0.00      0       0   0.00          0       0    0.00
 adt      0       0    0.00      0       0   0.00          0       0    0.00
 msc      0       0    0.00      0       0   0.00          0       0    0.00
 aio      0       0    0.00      0       0   0.00        607     607    1.00
 pio      0       0    0.00      0       0   0.00          0       0    0.00
 lio      0       0    0.00      0       0   0.00          0       0    0.00
```


onstat -g iof command: Print asynchronous I/O statistics

Use the **onstat -g iof** command to display the asynchronous I/O statistics by chunk or file.

This command is similar to the **onstat -D** command, except that **onstat -g iof** also displays information on nonchunk files. It includes information about temporary files and sort-work files.

Syntax:

```
>>-onstat-- -g--iof-----------------------------------------><
```

Example output

Figure 1. **onstat -g iof** command output

```
AIO global files:
gfd pathname      bytes read   page reads   bytes write   page writes  io/s
3   rootdbs       1918976      937          145061888     70831        36.5

      op type     count        avg. time
      seeks       0            N/A
      reads       937          0.0010
      writes      4088         0.0335
      kaio_reads  0            N/A
      kaio_writes 0            N/A
```

Output description

gfd
: Global file descriptor number for this chunk or file.

pathname
: The pathname of the chunk or file.

bytes read
: Number of byte reads that have occurred against the chunk or file.

page reads
: Number of page reads that have occurred against the chunk or file.

bytes write
: Number of byte writes that have occurred against the chunk or file.

page writes
: Number of page writes that have occurred against the chunk or file.

io/s
: Number of I/O operations that can be performed per second. This value represents the I/O performance of the chunk or file.

op type
: Type of operation.

count
: Number of times the operation occurred.

avg time
: Average time the operation took to complete.

onstat -g iog command: Print AIO global information

Use the **onstat -g iog** command to display global information about AIO.

Syntax:

```
>>-onstat-- -g--iog-----------------------------------------><
```

Example output

Figure 1. **onstat -g iog** command output

```
AIO global info:
   8 aio es
   5 open files
  64 max global files
```


onstat -g ioq command: Print I/O queue information

Use the **onstat -g ioq** command to display statistics about the number and types of operations performed by I/O queues.

Syntax:

```
>>-onstat-- -g--ioq--+------------+----------------------------><
                    '-queue_name-'
```

If a *queue_name* is given then only queues with that name are shown. If no *queue_name* is given then information is given for all queues.

Example output

Figure 1. **onstat -g ioq** command output

```
AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
sqli_dbg   0       0       0         0         0         0         0
 fifo   0       0       0         0         0         0         0
  adt   0       0       0         0         0         0         0
  msc   0       0       1       537         0         0         0
  aio   0       0       3      6537       238      5777         0
  pio   0       0       2      1103         0      1102         0
  lio   0       0       2     11795         0     11794         0
  gfd   3       0      17     17489      1526     15963         0
  gfd   4       0      17     18347      2384     15963         0
  gfd   5       0      16       220        41       179         0
  gfd   6       0       4         4         0         4         0
  gfd   7       0       4         4         0         4         0
  gfd   8       0       4         4         0         4         0
  gfd   9       0       9        54        24        30         0
  gfd  10       0      16       149        40       109         0
  gfd  11       0      16       621       128       493         0
  gfd  12       0      16      1953      1146       807         0
  gfd  13       0      16       409        71       338         0
  gfd  14       0      16       378        60       318         0
```

Output description

q name/id
: The name and number of the I/O queue. The name indicates what type of queue it is. The number is used to tell queues of the same name apart. Here is a list of the possible queue names and what each type of queue handles:

  sqli_dbg
  : Handles I/O for Technical Support's SQL Interface Debugging feature

  fifo
  : Handles I/O for FIFO VPs

  adt
  : Handles auditing I/O

msc
: Handles miscellaneous I/O

aio
: Handles IBM® Informix® asynchronous I/O

kio
: Handles kernel AIO

pio
: Handles physical logging I/O

lio
: Handles logical logging I/O

gfd
: Global File Descriptor - Each primary and mirror chunk is given a separate global file descriptor. Individual gfd queues are used depending on whether kaio is on and the associated chunk is cooked or raw.

len
: The number of pending I/O requests in the queue

maxlen
: The largest number of I/O requests that have been in the queue at the same time

totalops
: The total number of I/O operations that have been completed for the queue

dskread
: Total number of completed read operations for the queue

dskwrite
: Total number of completed write operations for the queue

dskcopy
: Total number of completed copy operations for the queue

onstat -g ipl command: Print index page logging status information

Use the **onstat -g ipl** command to display information about the status of index page logging.

Syntax:

```
>>-onstat-- -g--ipl-----------------------------------------><
```

Example output

Figure 1. **onstat -g ipl** command output

```
Index page logging status: Enabled
Index page logging was enabled at: 2008/12/20 16:01:02
```

Output description

Index page logging status
: Status of index page logging: Enabled or Disabled.

Index page logging was enabled at
: The date and time at which index page logging was enabled.

onstat -g iov command: Print AIO VP statistics

Use the **onstat -g iov** command to display asynchronous I/O statistics for each virtual processor.

Syntax:

```
>>-onstat-- -g--iov-----------------------------------------><
```

Example output

Figure 1. **onstat -g iov** command output

```
AIO I/O vps:
class/vp/id  s  io/s  totalops  dskread dskwrite  dskcopy  wakeups  io/wup  errors tempops
fifo  7  0   i   0.0       0        0        0        0        1     0.0       0       0
msc   6  0   i   0.1    9988        0        0        0     7833     1.3       0    9988
aio   5  0   i   0.0    4894     3341     1426        0     4393     1.1       0       0
aio   9  1   i   0.0      41        0       41        0       33     1.2       0       0
```

```
pio   4  0   i   0.0     199       0      199         0       200     1.0      0     199
lio   3  0   i   0.0    6344       0     6344         0      6344     1.0      0    6344
```

Output description

class
: The class of the virtual processor.

vp
: The ID number of the virtual processor within its class.

s
: Current® status of the AIO virtual processor

  f
  : Fork

  i
  : Idle

  s
  : Search

  b
  : Busy

  o
  : Open

  c
  : Close

io/s
: The average I/O speed (measured in operations per second) for the virtual processor since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

totalops
: Total number of I/O operations performed by this virtual processor since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

dskread
: Total number of read operations performed by this virtual processor since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

dskwrite
: Total number of write operations performed by this virtual processor since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

dskcopy
: Total number of copy operations performed by this virtual processor since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

wakeups
: For AIO VPs, the number of times the virtual processor has gone idle since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

io/wup
: For AIO VPs, the average number of I/O operations performed per wake-up by this virtual processor since the time the database server started or since the **onstat -z** command was last run, whichever happened last.

errors
: Total number of KAIO out of resource errors.

tempops (decimal)
: For internal use only. This is I/O operation counter that is maintained to determine when a new AIO VP should be added. It is applicable only when the AUTO_AIOVPS configuration parameter is enabled.

---

onstat -g lap command: Print light appends status information

Use the **onstat -g lap** command to display information about the status of light appends occurring in the system.

Syntax:

```
>>-onstat-- -g--lap-----------------------------------------><
```

Example output

Figure 1. **onstat -g lap** command output

```
Light Append Info
session id  address   cur_ppage   la_npused   la_ndata    la_nrows    bufcnt
31          b60a5e8   ffbff494    2938        2937        93990       4
```

Output description

Session id (decimal)
: Session ID performing the light append operation

address (hexadecimal)

Address of the light append buffer

cur_ppage (hexadecimal)
Current® physical page address

la_npused (decimal)
Number of pages allocated

la_ndata (decimal)
Number of data pages appended

la_nrows (decimal)
Number of rows appended

bufcnt (decimal)
Number of light append buffers

onstat -g laq command: Print log apply queues

Use the **onstat -g laq** command to print information about log recovery apply queues.

Use the onstat -g laq command to print information about log recovery apply queues. This includes logical log recovery on secondary servers as well as logical restore or logical recovery part of fast recovery. Log records from logical logs are assigned to replay worker threads according to the tablespace ID (partnum) associated with them; a subset of log records will be applied by the replay master thread.

For instance, in a high-availability cluster, the primary server sends log records to one or more secondary servers over the network. Each secondary server continuously replays the transaction logs from the primary server to ensure that data is replicated on the secondary server. Each tblspace on the primary server is assigned a queue on the secondary server in which to receive log records. A replay thread applies the log records stored in the queue to the secondary server. The log records are applied in the order in which they were received.

You use the **onstat -g laq** command to monitor the performance of the log apply queues, on a secondary server or during any other form of log recovery. Use this command if you suspect that the primary server performance is slowed because logs are not replaying quickly enough on the secondary server, or to monitor the progress made during logical restore. The Avg Depth (average depth) column indicates the average number of log records in the queue(Queue Size) incurred whenever putting a new log record on a queue. The Current/Last LSN column specifies the log record a replay thread currently is active on, or the last one it was replaying, with the Partval column typically specifying the tablespace ID this log record refers to. Transaction pointer and ID shown for a replay thread indicate a log record currently being applied.

When used in repeat mode, using -r [<seconds>][.<fraction>] option, an overall log record apply rate is calculated and shown.

On a secondary server, the transaction latency measured, in full seconds, with each end-of-transaction(COMMIT, ROLLBACK) log record as difference between local apply time and primary server's EoT time, is shown.

The **onstat -g laq** command is valid only when some form of logical log recovery is going on, otherwise only an **onstat** header is printed.

Syntax:

```
>>-onstat-- -g-- laq ------------------------------------------><
```

Example output

Figure 1. **onstat -g laq -r .3** command output from a remote standalone secondary server

```
IBM Informix Dynamic Server Version 14.10.FC6AEE -- Read-Only (RSS) -- Up 20:41:04 -- 116724 Kbytes
2021-05-20 18:24:22
Log Apply Info:
Thread           Queue         Total      Avg
                  Size        Queued    Depth   Current/Last LSN     Partval Txp       (Txid)
wreplay_1            1        938310    19.66     14087,0x482ec      100540 0x450a8c28 (29)
wreplay_2            0        782865    12.91     14087,0x48184
wreplay_3            3        937766    19.86     14087,0x45598      100542 0x450a8c28 (29)
wreplay_4            2        529755    14.43     14087,0x483bc      100543 0x450a8c28 (29)
wreplay_5            6        389432    10.93     14087,0x46500      10054e 0x450a8c28 (29)
wreplay_6            0        789238    10.90     14087,0x4318c
wreplay_7            0       1317820    20.26     14087,0x440b0
wreplay_8            0        991836    12.29     14086,0xb3d8
wreplay_9            0        851854    19.60     14086,0xb52c
wreplay_10           1        913434     9.42     14087,0x4849c          1d 0x450a8c28 (29)
mreplay                       689544              14087,0x4849c
Total:              13       9131854   150.26  Avg:    15.03

Secondary Apply Queue:          Total Buffers:12 Size:1024K Free Buffers:11
Log Recovery Queue:             Total Buffers:12 Size:16K Free Buffers:10
Log Page Queue:                 Total Buffers:512 Size:2K Free Buffers:512
Log Record Queue:               Total Buffers:50 Size:16K Free Buffers:42

Transaction Latency: 1 seconds
Apply rate:  30213.33 recs/sec - 9064 new recs in 300ms
```

Output description

Thread
The name of the apply thread for a given log record queue.

Queue Size
The number of log records queued for a given apply thread.

Total Queued
The total number of queued log records for a given apply thread.

Avg Depth
: The average number of logs in the queue at the time a queue insert operation occurred.

Secondary Apply Queue
: The secondary apply queue receives log buffers from the primary server. The values displayed represent the total number of buffers allocated to receiving log buffer records(SEC_DR_BUFS), the size of the buffers(LOGBUFF), and the number of currently unused buffers.

Log Recovery Queue
: The log recovery queue receives output from the secondary apply queue. The log buffers are converted to a format compatible with the **ontape** utility. The values displayed represent the total number of stream buffers in the recovery queue, the size of the stream buffers(LTAPEBLK), and the number of unused buffers.

Log Page Queue
: The log page queue receives output from the log recovery queue. The **ontape** formatting is removed and the data is divided into individual log pages. The values displayed represent the total number of log pages in the queue, the size of the queue, and the number of unused buffers.

Log Record Queue
: The log record queue receives output from the log page queue. The log pages are divided into individual log records. The values displayed represent the total number of log records in the recovery queue, the size of the queue, and the number of unused buffers.

Transaction Latency
: Time difference between last replayed transaction commit time at primary server and local server. For this to be accurate, operating system time must match between primary and secondary servers.

Apply rate
: Number of log records replayed per second. Apply rate is only shown with -r option.

onstat -g lmm command: Print low memory management information

Use the **onstat -g lmm** command to display information about automatic low memory management settings and recent activity.

Syntax:

```
>>-onstat-- -g lmm-----------------------------------------------><
```

Example output

Figure 1. **onstat -g lmm** command output

```
Low Memory Manager

Control Block       0x4cfca220
Memory Limit        300000 KB
Used                149952 KB
Start Threshold     10240 KB
Stop Threshold      10 MB
Idle Time           300 Sec
Internal Task       Yes
Task Name           'Low Memory Manager'
Low Mem TID         0x4cfd7178
# Extra Segments    0

Low Memory Manager Tasks

Task                 Count    Last Run
Kill User Sessions   267      04/04/2011.16:57
Kill All Sessions    1        04/04/2011.16:58
Reconfig(reduce)     1        04/04/2011.16:59
Reconfig(restore)    1        04/04/2011.17:59

Last 20 Sessions Killed

Ses ID Username Hostname PID   Time
194    sfisher  host01   13433 04/04/2011.16:57
201    sfisher  host01   13394 04/04/2011.16:57
198    sfisher  host01   13419 04/04/2011.16:57
190    sfisher  host01   13402 04/04/2011.16:57
199    sfisher  host01   13431 04/04/2011.16:57

Total Killed 177
```

Output description

Control Block
: Address of the internal control structure for automatic low memory management

Memory Limit
: Amount of memory to which the server is attempting to adhere

Used
: Amount of memory currently used by the server

Start Threshold
: Value for the automatic low memory management start threshold

Stop Threshold
: Value for the automatic low memory management stop threshold

Idle Time

The amount of time after which automatic low memory management considers a session idle

Internal Task
: Yes = using Informix® procedures
No = using user-defined procedures

Task Name
: Name of user-defined procedure

Low Mem TID
: Address of the automatic low memory management thread

Task
: Kill = Automatic processes ran and terminated sessions.
Reconfig(reduce) = Automatic processes ran and freed blocks of unused memory.
Reconfig(restore) = Automatic processes ran and restored services and configuration.

Count
: Number of times that the task ran

Last Run
: Date and time when the last task ran

Ses ID
: ID of session that was terminated (with an **onmode –z** command)

Username
: User name of the owner of the session

Hostname
: Name of the host where the session originated

PID
: Process ID

Time
: Date and time when the session was terminated

You use the LOW_MEMORY_MGR configuration parameter to enable the automatic low memory management.

Related reference:
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)
LOW_MEMORY_MGR configuration parameter
Related information:
Reserve memory for critical activities

onstat -g lmx command: Print all locked mutexes

Use the **onstat -g lmx** command to display information about all locked mutexes.

Syntax:

```
>>-onstat-- -g--lmx-----------------------------------------><
```

Example output

Figure 1. onstat -g lmx command output

```
Locked mutexes:
mid      addr              name              holder   lkcnt  waiter   waittime
119006   7000001e684b928   td_mutex          298      0
134825   7000002043a9148   free_lock         11009    0      200      22921
                                                             11010    22918
587817   70000022ddb3268   sync_lock1        200      0
593614   700000239ce7b68   SB_LTH_LATCH      875      0

Number of mutexes on VP free lists: 49
```

Output description

mid
: Internal mutex identifier

addr
: Address of locked mutex

name
: Name of the mutex

holder
: Thread ID of the thread that is holding the mutex
0 = The read/write mutex is held in shared mode

lkcnt
: For a read/write mutex, the current number of threads that are locking the mutex in shared mode. For a relockable mutex, the number of times the mutex was locked or relocked by the thread that is holding the mutex.

waiter
: List of IDs of the threads that are waiting for this mutex

waittime

Amount of time in seconds that the thread is waiting

onstat -g lsc command: Print active light scan status (deprecated)

The **onstat -g lsc** command has been superseded by the **onstat -g scn** command.

Syntax:

```
>>-onstat-- -g--lsc---------------------------------------------><
```

Example output

Figure 1. **onstat -g lsc** command output

```
Light Scan Info
descriptor  address           next_lpage  next_ppage        ppage_left  bufcnt  look_aside
3           474b74b0          4a0         7e2c80            416         1            N
```

Output description

descriptor (decimal)
: Light scan ID

address (hex)
: Memory address of the light scan descriptor

next_lpage (hex)
: Next logical page address to scan

next_ppage (hex)
: Next physical page address to scan

ppage_left (decimal)
: Number of physical pages left to scan in the current extent

#bufcnt (decimal)
: Number of light scan buffers used for this light scan

#look_aside (char)
: Whether look aside is needed for this light scan (`Y` = yes, `N` = no). Look asides occur when a thread needs to examine the buffer pool for existing pages to obtain the latest image of a page being light scanned.

Use the **onstat -g scn** command to display the status of a current scan, based on rows scanned on compressed tables, tables with rows that are larger than a page, and tables with VARCHAR, LVARCHAR, and NVARCHAR data. For more information, see onstat -g scn command: Print scan information.

onstat -g mem command: Print pool memory statistics

Use the **onstat -g mem** command to display the memory statistics for a pool.

If you run an SQL query that allocates memory from the PER_STMT_EXEC and PER_STMT_PREP memory duration pools, the **onstat -g mem** command displays information about the **PRP.sessionid.threadid** pool and the **EXE.sessionid.threadid** pool.

Syntax:

```
>>-onstat-- -g--mem--+-----------------------+-----------------><
                    '-pool name--session id-'
```

Session pools are named with the session number. If no argument is provided, information about all pools is displayed.

Example output

Figure 1. **onstat -g mem** command output

```
Pool Summary:
name                 addr              totalsize  freesize  #allocfrag  #freefrag
resident         R   10a001028         2420736    7960      2           2
res-buff         R   10a250028         8269824    7960      2           2
global           V   10aac0028         9351168    32648     650         11
...
...
...
onmode_mon       V   10b983028         20480      2752      108         1
13               V   10bd5d028         16384      5200      12          2
Blkpool Summary:
name                 addr              size       #blks       pre-hint    szavail|
```

```
global        V     10aac8920        0         0          0          0
xmf_msc_pl    V     10ac84ca0        954368    73         0          0
```

Output description

Pool Summary

name
: Pool name
: Shared memory segment type where the pool is created

addr
: Pool memory address

totalsize
: Pool size, in bytes

freesize
: Free memory in pool

#allocfrag
: Allocated fragments in pool

#freefrag
: Free fragments in pool

Blkpool Summary

name
: Pool name
: Shared memory segment type where pool is created

addr
: Pool memory address

size
: Pool size, in bytes

#blks
: Number of blocks in pool

Related information:
The PER_STMT_EXEC memory duration
The PER_STMT_PREP memory duration

onstat -g mgm command: Print MGM resource information

Use the **onstat -g mgm** command to show resource information about Memory Grant Manager (MGM).

You can use the **onstat -g mgm** command to monitor how MGM coordinates memory use and scan threads. This command reads shared-memory structures and provides statistics that are accurate at the instant that the command runs.

Syntax:

```
>>-onstat-- -g--mgm-----------------------------------------><
```

The **onstat -g mgm** output shows a unit of memory that is called a *quantum*. The *memory quantum* represents a unit of memory, as follows:

```
memory quantum = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

The following calculation shows the memory quantum for the values that the **onstat -g mgm** output shows:

```
memory quantum = 4000 kilobytes / 31
               = 129 kilobytes
```

The database server adjusts the value of a quantum as needed when it grants memory. Therefore, the value of the quantum as shown by the **onstat -g mgm** command is not always accurate.

The *scan thread quantum* is always equal to `1`.

Example output

Figure 1. onstat -g mgm command output

```
Memory Grant Manager (MGM)
--------------------------
MAX_PDQPRIORITY:  100
DS_MAX_QUERIES:    31
DS_MAX_SCANS:      1048576
DS_NONPDQ_QUERY_MEM: 128 KB
DS_TOTAL_MEMORY:   4000 KB

Queries:   Active     Ready   Maximum
                        0         0        31
Memory:     Total      Free   Quantum
(KB)         4000      4000       128

Scans:      Total      Free   Quantum
               1048576   1048576         1
```

```
Load Control:    (Memory)       (Scans)   (Priority)   (Max Queries)    (Reinit)
                       Gate 1        Gate 2       Gate 3          Gate 4      Gate 5
(Queue Length)          0             0            0               0           0

Active Queries:  None
Ready Queries:  None
Free Resource         Average #         Minimum #
--------------     ---------------     ---------
Memory                0.0 +- 0.0           500
Scans                 0.0 +- 0.0       1048576

Queries               Average #         Maximum #    Total #
--------------     ---------------     ---------    -------
Active                0.0 +- 0.0             0          0
Ready                 0.0 +- 0.0             0          0

Resource/Lock Cycle Prevention count:  0
```

Output description

The first portion of the output shows the values of the PDQ configuration parameters.

The second portion of the output describes MGM internal control information. It includes four groups of information. The first group is **Queries**:

Active
: Number of PDQ queries that are currently running

Ready
: Number of user queries ready to run but whose execution the database server deferred for load-control reason

Maximum
: Maximum number of queries that the database server allows to be active. Reflects current value of the DS_MAX_QUERIES configuration parameter

The next group is **Memory**:

Total
: KB of memory available for use by PDQ queries (DS_TOTAL_MEMORY specifies this value.)

Free
: KB of memory for PDQ queries not currently in use

Quantum
: Approximate number of KB of memory in a memory quantum

The next group is **Scans**:

Total
: The total number of scan threads as specified by the DS_MAX_SCANS configuration parameter

Free
: Number of scan threads currently available for decision-support queries

Quantum
: The number of scan threads in a scan-thread quantum

The last group in this portion of the output describes MGM **Load Control**:

Memory
: Number of queries that are waiting for memory

Scans
: Number of queries that are waiting for scans

Priority
: Number of queries that are waiting for queries with higher PDQ priority to run

Max Queries
: Number of queries that are waiting for a query slot

Reinit
: Number of queries that are waiting for running queries to complete after an **onmode -M** or **-Q** command

The next portion of the output, **Active Queries**, describes the MGM active and ready queues. This portion of the output shows the number of queries that are waiting at each gate:

Session
: The session ID for the session that initiated the query

Query
: Address of the internal control block that is associated with the query

Priority
: PDQ priority that is assigned to the query

Thread
: Thread that registered the query with MGM

Memory
: Memory that is currently granted to the query or memory that is reserved for the query (Unit is MGM pages, which is 8 KB.)

Scans
: Number of scan threads currently used by the query or number of scan threads that are allocated to the query

Gate
: Gate number at which query is waiting

The next portion of the output, **Free Resource**, provides statistics for MGM free resources. The numbers in this portion and in the final portion reflect statistics since system initialization or the last **onmode -Q**, **-M**, or **-S** command. This portion of the output contains the following information:

Average

Average amount of memory and number of scans

Minimum
: Minimum available memory and number of scans

The next portion of the output, **Queries**, provides statistics about MGM queries:

Average
: Average active and ready queue length

Maximum
: Maximum active and ready queue length

Total
: Total active and ready queue length

Resource/Lock Cycle Prevention count
: Number of times the system immediately activated a query to avoid a potential deadlock. (The database server can detect when some of the queries in its queue might create a deadlock situation if the queries are not run immediately.)

Related reference:
DS_MAX_QUERIES configuration parameter
DS_MAX_SCANS configuration parameter
MAX_PDQPRIORITY configuration parameter
DS_NONPDQ_QUERY_MEM configuration parameter
DS_TOTAL_MEMORY configuration parameter

onstat -g nbm command: Print a block bit map

Use the **onstat -g nbm** command to display the block bit map for the nonresident segments.

Each bit of the bitmap represents a 4 KB block. If the block is used, then the bit is set to 1. If the block is free, the bit is set to 0. The bitmap is shown as a series of hexadecimal numbers. The bits, and therefore the blocks, are numbered starting at 0 so the first block is block 0, the second is block 1, and so on.

Syntax:

```
>>-onstat-- -g--nbm-------------------------------------------><
```

Example output

This example shows the bitmap for the segment of virtual memory at 0x10CC00000. The bitmap itself is at 0x10CC00290. All 1792 blocks of the segment are free except for block 0 and block 1023.
Figure 1. **onstat -g nbm** command output

```
Block bitmap for virtual segment address 0x10cc00000:
 address = 0x10cc00290, size(bits) = 1792
 used = 1, largest_free = -1
      0:8000000000000000 0000000000000000 0000000000000000 0000000000000000
    256:0000000000000000 0000000000000000 0000000000000000 0000000000000000
    512:0000000000000000 0000000000000000 0000000000000000 0000000000000000
    768:0000000000000000 0000000000000000 0000000000000000 0000000000000001
   1024:0000000000000000 0000000000000000 0000000000000000 0000000000000000
   1280:0000000000000000 0000000000000000 0000000000000000 0000000000000000
   1536:0000000000000000 0000000000000000 0000000000000000 0000000000000000
```

Output description

address
: The starting address of the bitmap.

size
: The number of bits in the bitmap. This is also the number of 4 KB blocks in the memory segment.

used
: The total number of bits in the bitmap that are set to 1. This is also the number of 4 KB blocks that are in use in the memory segment.

largest free
: If this is a value other than -1 it is the largest number of consecutive bits that are free, which is also the number of 4 KB blocks in the largest contiguous set of blocks in the memory segment.
A value of -1 means that the largest free space has not been calculated. The database server only calculates the largest free space if it tries to allocate a set of blocks starting at the *lastalloc* block but there is not enough free space. The value is set to -1 again as soon as another block is allocated in the segment.

onstat -g nsc command: Print current shared memory connection information

Use the **onstat -g nsc** command to display information about shared memory connections either for all of the current connections or for a specified connection ID.

Syntax:

```
>>-onstat-- -g--nsc--+-----------+----------------------------><
                    '-client_id-'
```

If no *client_id* is provided, information about all current shared memory connections to the database server is given. If a *client_id* is provided then this command gives more detailed information about the shared memory connection with that ID.

Example output

This is output of **onstat -g nsc** with no *client_id*. It shows that there is only one user currently connecting to the database server through shared memory. That connection has an ID of 0.

Figure 1. **onstat -g nsc** command output

```
  clientid  clientPID      state #serverbufs #clientbufs    #rdwrts
         0       6031  Connected           4           4         12
```

This example shows output from running the command using a *client_id* of 0.

Figure 2. **onstat -g nsc** command with client id output

```
Network Shared Memory Status for Client: 0

    clientid    clientPID        state  #serverbufs  #clientbufs      #rdwrts
           0        18949    Connected            4            4       447048

     needbuf        segid        semid       semnum     be_semid    be_semnum
           0         1303       851969            0       851969           10

  be_curread  be_curwrite   fe_curread  fe_curwrite
          -1            1            0            2

 be_nextread be_nextwrite  fe_nextread fe_nextwrite
           2            2            4            3

readyqueue
  -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```

```
    Server Buffers                              Client Buffers
  i: bufid     status   offset  fe_addr         bufid     status   offset  fe_addr
  0:     4      inuse     4474   804474             0      avail     3424   803424
  1:     5      inuse     4888   804888             1      avail     3838   803838
  2:     6      avail     4c9c   804c9c             2      inuse     3c4c   803c4c
  3:     7      avail     50b0   8050b0             3      avail     4060   804060
  4:    -1       free        0        0            -1       free        0        0
  5:    -1       free        0        0            -1       free        0        0
```

Output description

clientid
: Server assigned ID

clientPID
: Client process ID

state
: State of connection

  Connected
  : The client has established a connection with the server.

  Con1
  : The server has successfully set up a connection with the client, but the client has not yet been notified of it.

  Waiting
  : The server is in the process of setting up a connection with the client.

  Reject
  : Client connection has been rejected by the server, normally because the server is shutting down or not yet in on-line mode.

  Closed
  : Server has closed the connection with the client. Client might not be aware of the fact yet.

  Not connected
  : Server is initializing internal structures for the connection.

  Unknown
  : Connection has been closed and the client is aware of the fact. Server is cleaning up internal structures.

#serverbufs
: Database server buffers currently allocated

#clientbufs
: Client buffers currently allocated

#rdwrts
: The total number of reads and writes performed through this connection since it was created.

The following items are only in the output if you run the **onstat -g nsc** command with a *client_id*:

needbuf
: Indicates if server is waiting for a buffer to be freed

  0

False
1
True

segid
Shared memory segment ID
semid
Semaphore ID
semnum
Semaphore number in the semaphore ID
be_semid
Backend semaphore ID
be_semnum
Backend semaphore number in the semaphore ID
be_curread
ID of backend buffer being read
be_curwrite
ID of backend buffer being written
fe_curread
ID of frontend buffer being read
fe_currwrite
ID of frontend buffer being written
be_nextread
ID of next backend buffer to be read
be_nextwrite
ID of next backend buffer to be written
fe_nextread
ID of next frontend buffer to be read
fe_nextwrite
ID of next frontend buffer to be written
readyqueue
Queue of the shared memory buffer ids

Buffers
i
Internal location key of message buffer
bufid
Message buffer ID
status
Status of message buffer
offset
Offset of memory buffer in shared memory segments
fe_addr
Frontend address of message buffer

Related reference:
NETTYPE configuration parameter

onstat -g nsd command: Print poll threads shared-memory data

Use the **onstat -g nsd** command to display information about shared-memory data for poll threads.

Syntax:

```
>>-onstat-- -g--nsd-------------------------------------------><
```

Example output

Figure 1. **onstat -g nsd** command output

```
Network Shared Memory Data for Poll Thread: 0
Free Message Buffer Bitmap
(bitmap address = 10b9eef80, bitmap size 480)
000000010b9eef80:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
000000010b9eefa0:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
Free Message Buffer Status Bitmap
(bitmap address = 10ca0a9b0, bitmap size 50)
000000010ca0a9b0:ffffffff ffffff
Message Buffer Table
bufid   clientid            addr
Message Buffer Status Table
clientid      netscb addr             addr           offset
```

Related reference:
NETTYPE configuration parameter

onstat -g nss command: Print shared memory network connections status

Use the **onstat -g nss** *sessionid* command to display information about the status of the shared memory network connections.

Syntax:

```
>>-onstat-- -g--nss--+-----------+----------------------------><
                    '-sessionid-'
```

If no *sessionid* is provided, a one-line summary for each shared memory connection is listed.

Example output

Figure 1. **onstat -g nss** command output

```
clientid  clientPID   state     #serverbufs #clientbufs    #rdwrts
       1      14018  Connected         4           4        331
       0      12398  Connected         4           4        294
       2      14036  Connected         4           4         59
```

Output description

clientid (decimal)
: Server assigned value for lookups

clientPID (decimal)
: Client process ID

state (string)
: Current® state of the connection.

  - Connected
  - Con1
  - Waiting
  - Reject
  - Bedcover
  - Closed
  - Not connected
  - Unknown

#serverbufs (dec)
: Number of database server buffers currently allocated

#clientbufs (dec)
: Number of client buffers currently allocated

#rdwrts (dec)
: Total number of buffers in use

Related reference:
NETTYPE configuration parameter

onstat -g ntd command: Print network statistics

Use the **onstat -g ntd** command to display network statistics by service.

Syntax:

```
>>-onstat-- -g--ntd-------------------------------------------><
```

Example output

Figure 1. **onstat -g ntd** command output

```
global network information:
  #netscb connects     read    write  q-limits  q-exceed alloc/max
   4/   5       11        0     3546 3549/  10   10/   0    0/   0

Client Type     Calls    Accepted    Rejected        Read       Write
sqlexec         yes            11           0        3531        3540
srvinfx         yes             0           0           0           0
onspace         yes             0           0           4           9
onlog           yes             0           0           0           0
onparam         yes             0           0           0           0
oncheck         yes             0           0           0           0
onload          yes             0           0           0           0
onunload        yes             0           0           0           0
onmonitor       yes             0           0           0           0
```

```
dr_accept       yes             0          0          0          0
cdraccept       no              0          0          0          0
ontape          yes             0          0          0          0
srvstat         yes             0          0          0          0
asfecho         yes             0          0          0          0
listener        yes             0          0         11          0
crsamexec       yes             0          0          0          0
onutil          yes             0          0          0          0
drdaexec        yes             0          0          0          0
smx             yes             0          0          0          0
safe            yes             0          0          0          0
Totals                         11          0       3546       3549
```


onstat -g ntm command: Print network mail statistics

Use the **onstat -g ntm** command to display statistics about network mail.

Syntax:

```
>>-onstat-- -g--ntm-------------------------------------------><
```

Example output

Figure 1. **onstat -g ntm** command output

```
global network information:
  #netscb connects      read    write  q-limits  q-exceed alloc/max
   4/   5       11         0     3546 3549/  10   10/   0    0/   0

Network mailbox information:
 box    netscb thread name      max received   in box    max in box full signal
   5  f07e8b0 soctcppoll        10       24         0          1         0    yes
   6  f0b6ad8 soctcplst         10        0         0          0         0     no
   7  f0e8b18 soctcplst         10        0         0          0         0     no
```


onstat -g ntt command: Print network user times

Use the **onstat -g ntt** command to display information about network user times.

Syntax:

```
>>-onstat-- -g--ntt-------------------------------------------><
```

Example output

Figure 1. **onstat -g ntt** command output

```
global network information:
  #netscb connects      read    write    q-limits  q-exceed alloc/max
   3/   3        0         0        0    135/  10    0/   0    2/   0
Individual thread network information (times):
 netscb   thread name   sid  open     read     write     address
c76ea28   ontape        61   14:34:48 14:34:50 14:34:50
c63e548   tlitcplst      4   14:30:43 14:34:48           server.ibm.com|5006|tlitcp
c631028   tlitcppoll     3   14:32:32
```


onstat -g ntu command: Print network user statistics

Use the **onstat -g ntu** command to display information about network user statistics.

Syntax:

```
>>-onstat-- -g--ntu-------------------------------------------><
```

Example output

Figure 1. **onstat -g ntu** command output

```
global network information:
  #netscb connects     read    write    q-free  q-limits  q-exceed alloc/max
   2/   3       16     2611     2603    1/   1  135/  10    0/   0    1/   1

Individual thread network information (basic):
  netscb type   thread name    sid   fd poll    reads   writes q-nrm q-pvt q-exp
 d1769f0 soctcp soctcplst        3    1    5       16        0  0/ 0  0/ 0  0/ 0
 d1199f0 soctcp soctcppoll       2    0    5     2595        0  0/ 0  0/ 0  0/ 0
```


onstat -g opn command: Print open partitions

Use the **onstat -g opn** command to display a list of the partitions (tables and indexes), by thread ID, that are currently open in the system.

Use the ***thread_id*** option to restrict the list to a specified ID.

Syntax:

```
>>-onstat-- -g--opn--+-----------+-----------------------------><
                    '-thread_id-'
```

Output description

This information is used by Software Support. The output might change over time and depends on your product version or fix pack.

onstat -g osi: Print operating system information

Use the **onstat -g osi** command to display information on your operating system resources and parameters, including shared memory and semaphore parameters, the amount of memory currently configured on the computer, and the amount of memory that is unused.

Example Output

The **onstat -g osi** command also displays statistics on the hardware processors on your computer.

Use this command when the server is not online.

Figure 1. **onstat -g osi** Command Output

```
Machine Configuration....
OS Name                              Linux
OS Release                           2.6.9-34.ELsmp
OS Node Name                         idas
OS Version                           #1 SMP
OS Machine                           x86_64
Number of processors                 4
Number of online processors          4
System memory page size              4096 bytes
System memory                        7970 MB
System free memory                   1536 MB
Number of open files per process     1024
shmmax            33554432
shmmin            1
shmids            4096
shmNumSegs        2097152
semmap            << UnSupported >>
semids            128
semnum            32000
semundo           << UnSupported >>
semNumPerID       250
semops            32
semUndoPerProc    << UnSupported >>
semUndoSize       20
semMaxValue       32767
```


onstat -g pd command: Print push data session-related information

Use the **onstat -g pd** command to display information about the push data session.

Syntax:

```
>>-onstat-- -g--pd--+------------+----------------------------><
                    '-session_id-'
```

You can specify one of the following invocations.

onstat -g pd
Displays a one-line summary for each session

onstat -g pd *session_id*
Displays information for a specific session

Example output for all sessions

Figure 1. **onstat -g pd** command output

```
push-data subsystem structure at 0x4eebb028
push-data session structure at 0x4eecc028
push-data sql session id: 0 0x0
Marked as detachable session, session unique id: 2
Smartblob file descriptor: 39
Number of event conditions: 0
Number of pending event operations: 0
Number of discarded event operations: 0
Total event operations returned to client:
```

Example output for a specific session

Figure 2. **onstat -g pd** *98* command output

```
    push-data subsystem structure at 0x4eebb028
push-data session structure at 0x4eecc028
push-data sql session id: 98 0x62
Marked as detachable session, session unique id: 2
Smartblob file descriptor: 39
Number of event conditions: 1
Number of pending event operations: 0
Number of discarded event operations: 0
Total event operations returned to client: 0
```


onstat -g pd event command: Print push data event-related information

Use the **onstat -g pd** event command to display information about the push data event.

Syntax:

```
>>-onstat-- -g--pd--+------------+-- event---------------------><
                    '-session_id-'
```

You can specify one of the following invocations.

onstat -g pd event
Displays a one-line summary for each event

onstat -g pd *session_id* **event**
Displays information of an event for a specific session

Example output for all events

Figure 1. **onstat -g pd event** command output

```
IBM Informix Dynamic Server Version 14.10.F -- On-Line -- Up 00:20:13 -- 185676 Kbytes
push-data subsystem structure at 0x4eebb028
push-data session structure at 0x4eecc028
push-data sql session id: 98 0x62
Marked as detachable session, session unique id: 2
Number of event conditions: 1
    Push-data event structure at 0x4ece0028
        Full Table Name: test:informix.t1
User data:
Replicate name: pushrepl_98_1497908205_1628814989
```

Note:

- Events can only be registered on tables with logging enabled.
- Events require a primary key, a unique index, or ER key to register events on a table.
- Events cannot be registered on sysmaster pseudo tables.
- Events cannot be registered on timeseries VTI tables.
- Event condition SELECT statements cannot include large objects such as byte, text, blob, clob, or collection datatypes.

- WHERE clauses of event condition SELECT statements cannot refer to other tables or contain sub-queries.
- A read call always returns completed event documents.
- The following message is returned upon timeout from the read API:

  ```
  {ifx_isTimeout:"true"}
  ```

- The following message is returned if event documents are discarded from exceeding the max_pending_ops attribute threshold. The document contains the cumulative count of the total number of discarded event documents.

  ```
  {ifx_warn_total_skipcount:10}
  ```

- The following error message is returned with the ifx_error attribute if the input buffer size is too small, or when other fatal error conditions arise:

  ```
  {ifx_error:" Smartblob read API buffer size ## is too small, expected
  size should be atleast ##"}
  ```

- Event data will not be staged if push-data client is disconnected from the server.
- The client cannot read events for the past time. The commit_logid, commit_logpos and commit_time values in the input document cannot be set for the past time.
- The smartblob read API always returns data in JSON document format. This includes event data, warning messages, and error conditions.
- The input buffer value that is passed to the smartblob read API should be at least 1KB in size.
- When the maxrecs attribute for the session is set to more than one record, then the smartblob read API can return data for multiple events in one read call. The format of the output document format is as follows:

  ```
  { [ {document1}, {document2}]}
  ```

- When the server is restarted, the push-data client might receive duplicates of event data. Therefore, it is recommended to discard duplicate event data by saving the last read event commit_logid, and commit_logpos records, and use this commit log position to register push-data event conditions with the server.
- While registering new push-data event conditions, an internal transient cascade replicate definition is created. The replicate definition gets deleted when a session disconnects from the server. Cascading logic are added to capture changes applied by ER apply threads.
- Detached sessions will be marked as detachable session, session unique id: 2

onstat -g pfsc command: Print partition free space cache information

Use the **onstat -g pfsc** command to display information about all partition free space caches.

```
Syntax:
>>onstat -g pfsc [full] [<partnum>]
```

Example output

Figure 1. onstat -g pfsc command output

```
Partition Free Space Cache
tblnum    flg  npages   avgfr nsearch   nmiss     pgsrch    pgskp  pgbusy mvbin
100004    L-   1        80    409       19        390       0      0      1
100005    L-   1        322   5039      75        4964      0      0      2
100006    L-   7        187   301       30        271       0      0      1
10000c    L-   0        0     0         0         0         0      0      0
```

Output description

tblnum
: The partition number

flg
: First position: 'L' = "Lite", 'B' = "Boosted"

  Second position: 'I' = "Boosted cache creation in progress"

npages
: Number of pages described by cache

avgfr
: Average free bytes per page in cache

nsearch
: Number of times a bin in the cache has been searched by inserts

nmiss
: Number of times a search has turned up empty

pgsrch
: Number of cached pages searched

mvbin
: Pages moved from one bin to another

For more information, see Partition Free Space Cache (PFSC) code.

onstat -g pos command: Print file values

Use the **onstat -g pos** command to display the values in the **$INFORMIXDIR/etc/.infos.DBSERVERNAME** file.

Syntax:

```
>>-onstat-- -g--pos-----------------------------------------><
```

Example output

Figure 1. **onstat -g pos** command output

```
  1   7   0 infos ver/size 3 264
  2   1   0 snum     0 52564801 44000000 4139 demo_on
  3   4   0 onconfig path /opt/IBM/informix/etc/onconfig.demo_on
  4   5   0 host informixva
  5   6   0 oninit ver IBM Informix Dynamic Server Version 11.70.UC2DE
  6   8   0 sqlhosts path /data/IBM/informix/etc/sqlhosts.demos
  7   3 -32767 sema 32769
  8   2 -32768 shm  32768 52564801 44000000 114176000 R
  9   2   1 shm      1 52564802 4ace3000 67108864 V
```


onstat -g ppd command: Print partition compression dictionary information

Use the **onstat –g ppd** command to display information about the active compression dictionaries that were created for compressed tables and table fragments or compressed B-tree indexes. You can choose to print information for a particular numbered partition or for all open partitions.

The **onstat –g ppd** command prints the same information that the **syscompdicts_full** table and the **syscompdicts** view in the **sysmaster** database display. The only difference is that the **syscompdicts_full** table and the **syscompdicts** view display information about all compression dictionaries, not just the active dictionaries.

Syntax:

```
>>-onstat-- -g--ppd--+------------------+----------------------><
                     +-partition number-+
                     '- 0---------------'
```

If you specify a partition number, **onstat -g ppd** prints the partition profile for that partition. If you specify `0`, this option prints profiles for all partitions.

Example output

Figure 1. **onstat –g ppd** Output

partnum	ColOffset	DbsNum	CrTS	CrLogID	CrLogPos	DrTS	DrLogID	DrLogPos
0x1001d5	-1	1	1393371661	4	16339024	0	0	0
0x1001d5	4	1	1393371661	4	16355408	0	0	0

Output description

partnum
: Partition number to which the compression dictionary applies

ColOffset
: The byte offset for a compressed partition blob column. `-1` means that only the row is compressed

DbsNum
: Number of the dbspace that the dictionary resides in

CrTS
: Timestamp that shows when the dictionary was created

CrLogID
: Unique ID for the logical log that was created when the dictionary was created

CrLogPos
: Position within the logical log when the dictionary was created

DrTS
: Timestamp that shows when the dictionary was purged

DrLogID
: Unique ID for the logical log that was created when the dictionary was purged

DrLogPos
: Position within the logical log when the dictionary was purged

onstat -g ppf command: Print partition profiles

Use the **onstat -g ppf *partition_number*** command to display the partition profile for the specified partition number.

Use the **onstat -g ppf** or the **onstat -g ppf 0** command to display the profiles for all partitions. If the TBLSPACE_STATS configuration parameter is set to 0, then the **onstat -g ppf** command displays: `Partition profiles disabled`.

For more information on the **onstat -g ppf** command, see the *IBM® Informix® Performance Guide*.

Syntax:

```
>>-onstat-- -g--ppf--+-partition_number-+----------------------><
                     '-0----------------'
```

Example output

Figure 1. **onstat -g ppf** command output

```
Partition profiles
partnum    lkrqs lkwts dlks  touts isrd  iswrt isrwt isdel bfrd  bfwrt seqsc rhitratio
0x100001   0     0     0     0     0     0     0     0     0     0     0     0
0x100002   1506  0     0     0     416   4     0     4     1282  20    0     97
0x100003   15    0     0     0     5     0     0     0     20    0     0     75
0x1000a5   0     0     0     0     0     0     0     0     12    0     0     67
0x1000e3   4     0     0     0     1     0     0     0     4     0     0     25
0x200001   0     0     0     0     0     0     0     0     0     0     0     0
0x300001   0     0     0     0     0     0     0     0     0     0     0     0
0x400001   0     0     0     0     0     0     0     0     0     0     0     0
```

Output description

partnum (hex)
: The partition number

lkrqs (decimal)
: The number of lock requests for a partition

lkwts (decimal)
: The number of lock waits for a partition

dlks (decimal)
: The number of deadlocks for a partition

touts(decimal)
: The number of remote deadlock timeouts for a partition

isrd (decimal)
: The number of read operations for a partition

iswrt (decimal)
: The number of write operations for a partition

isrwt (decimal)
: The number of rewrite or update operations for a partition

isdel (decimal)
: The number of delete operations for a partition

bfrd (decimal)
: The number of buffer read operations, in pages

bfwrt (decimal)
: The number of buffer write operations, in pages

seqsc (decimal)
: The number of sequential scans for a partition

rhitratio (percentage)
: The ratio of disk read operations to buffer read operations

Related reference:
TBLSPACE_STATS configuration parameter

onstat -g pqs command: Print operators for all SQL queries

Use the **onstat –g pqs** command to display information about the operators used in all of the SQL queries that are currently running.

You can use this command to troubleshoot an application, to find which operators are running for the query and for how long, and how many rows each operator returns. While the EXPLAIN file contains information that will give you a general sense of the query plan, the **onstat –g pqs** command displays the runtime operator information for the query and the query plan.

Syntax:

```
>>-onstat-- -g--pqs--+-----------+-----------------------------><
                     '-sessionid-'
```

You can specify one of the following invocations:

Table 1. Descriptions of each **onstat -g pqs** command invocation

Invocation	Explanation

Invocation	Explanation
onstat -g pqs	Displays a one-line summary for each session.
onstat -g pqs sessionid	Displays information for the session that you specify.

Example output

The following example shows the results when three separate SQL statements are run in different sessions. The statements are:

```
select * from syscolumns;
select * from systables a, systables b;
update t1 set rowsize = rowsize +100;
```

Figure 1. **onstat –g pqs** command output

```
Query Operators:
addr     ses-id   opname    phase   rows    time           in1          in2     stmt-type
ae50b3a  23       scan      open    0       00:00.00         0            0     SELECT
af269d0   5       nljoin    next    224717  00:01.82   af26a90      aeb4478     SELECT
af26a90   5       scan      next    472     00:00.20         0            0     SELECT
aeb4478   5       scan      next    50      00:01.63         0            0     SELECT
ad3c530  26       scan      open    0       00:00.00         0            0     UPDATE (all)
```

Output description

addr
: The address of the operator in memory. You can use this address to track which SCAN operator belongs to each JOIN operator.

ses-id
: The session ID in which the SQL statement was run.

opname
: The name of the operator.

phase
: The phase in which the operator was used. For example OPEN, NEXT, CLOSE.

rows
: The number of rows that are processed by the operator.

time
: The amount of time to process the operator. The time is displayed to the millisecond. A time of 01:20.10 is 1 minute, 20 seconds, and 10 milliseconds.

in1
: The first (outer) operator in the join.

in2
: The second (inner) operator in the join.

stmt-type
: The type of SQL statement, such as SELECT, UPDATE, DELETE.

onstat -g prc command: Print sessions using UDR or SPL routines

Use the **onstat -g prc** command to display the number of sessions that are currently using the UDR or SPL routine.

Syntax:

```
>>-onstat-- -g--prc-----------------------------------------><
```

Example output

Figure 1. onstat -g prc command output

```
UDR Cache:
    Number of lists          : 31
    PC_POOLSIZE              : 127

UDR Cache Entries:

list id   ref  drop hits       last_access          heap_ptr         udr_name
--------------------------------------------------------------------------------

0    79   0    0    1          2020-05-12 10:36:34  464a1c38         stores_demo@myserver:.proc1
0    46   0    0    10         2020-05-12 10:36:34  4dcba038         stores_demo@myserver:.proc2
0    376  0    0    1          2020-05-12 10:36:32  46303038         stores_demo@myserver:.proc3
0    363  0    0    1          2020-05-12 10:36:32  462f2c38         stores_demo@myserver:.proc4

    Total number of udr entries : 254
    Number of entries in use    : 9
```

Output description

Number of lists

Number of lists in the UDR cache

PC_POOLSIZE
: Number of entries that can be cached at one time

list
: UDR cache hash chain ID (bucket number)

id
: Unique ID of the routine

ref
: Number of sessions that are currently accessing the UDR or SPL routine from the cache

drop
: Whether the routine is marked to be dropped

hits
: The number of times the cache entry is accessed.

last_access
: The time at which the cache entry was last accessed.

heap_ptr
: Heap address that is used to store this entry

udr_name
: The name of the UDR or SPL routine in the cache

Total number of udr entries
: Number of entries in the cache

Number of entries in use
: Number of entries that are being used

onstat -g proxy command: Print proxy distributor information

Use the **onstat -g proxy** command to display information about proxy distributors. The output of the **onstat -g proxy** command differs slightly depending on whether the command is run on a primary server or on a secondary server.

Syntax:

```
>>-onstat-- -g--proxy--+-all-------------------------------------------------+-><
                       '-proxy_id--+-----------------------------------------+-'
                                   '-proxy_transaction_id--+-----------------+-'
                                                           '-sequence_number-'
```

Invocation	Explanation
onstat -g proxy	Displays proxy distributor information
onstat -g proxy all	When run on the primary server, displays information about proxy distributors and proxy agent threads. When run on the secondary server, displays information about all sessions currently performing updates to secondary servers.
onstat -g proxy *proxy_id proxy_transaction_id sequence_number*	This option is valid only on secondary servers. Displays detailed information about the current work bring performed by a given proxy distributor. The *proxy_transaction_id* and *sequence_number* are optional parameters. When supplied, the first number is considered the *proxy_transaction_id*, and the second is interpreted as the *sequence_number*. If the supplied *proxy_transaction_id* or *sequence_number* do not exist, the command output is the same as the output for **onstat -**

Example output using the `onstat -g proxy` command on a primary server

Figure 1. **onstat -g proxy** command output (run from primary server)

```
Secondary           Proxy       Reference Transaction  Hot Row
Node                ID          Count     Count        Total
nagpur_sdc1         2619        0         2            0
nagpur_c2           2632        0         1            0
nagpur_sec          2633        0         1            0    I
```

Output description

Secondary Node
: Name of the secondary server as it is known by the primary server.

Proxy ID
: ID of the proxy distributor. Proxy IDs are unique within a high-availability cluster.

Reference Count
: Indicates the number of threads that are using the information for the current transaction. When the count becomes 0, the transaction processing is complete (either successfully or unsuccessfully).

Transaction Count
: The number of transactions currently being processed by the proxy distributor.

Hot Row Total
: Total number of hot rows ever handled by the proxy distributor.

Example output using the onstat -g proxy command on a secondary server

Figure 2. **onstat -g proxy** command output (run from secondary server)

```
Primary            Proxy      Reference Transaction  Hot Row
Node               ID         Count     Count        Total
nagpur             2619       5         2            0
```

Output description

Primary Node
Name of the primary server.
Proxy ID
ID of the proxy distributor. Proxy IDs are unique within a high-availability cluster.
Reference Count
Indicates the number of threads that are using the information for the current transaction. When the count becomes 0, the transaction processing is complete (either successfully or unsuccessfully).
Transaction Count
The number of transactions currently being processed by the proxy distributor.
Hot Row Total
Total number of hot rows ever handled by the proxy distributor.

Example output using the onstat -g proxy all command on a primary server

Figure 3. **onstat -g proxy all** command output (run from primary server)

```
Secondary          Proxy      Reference Transaction  Hot Row
Node               ID         Count     Count        Total
nagpur_sdc1        2619       0         2            0
nagpur_c2          2632       0         1            0
nagpur_sec         2633       0         1            0

TID      Flags      Proxy  Source   Proxy    Current  sqlerrno iserrno
                    ID     SessID   TxnID    Seq
94       0x00000224 2619   21       1        29       0        0
95       0x00000224 2619   22       2        68       0        0
93       0x00000224 2632   21       2        2        0        0
91       0x00000224 2633   25       1        6        0        0
```

Output description

Secondary Node
Name of the secondary server as it is known by the primary server.
Proxy ID
ID of the proxy distributor. Proxy IDs are unique within a high-availability cluster.
Reference Count
Indicates the number of threads that are using the information for the current transaction. When the count becomes 0, the transaction processing is complete (either successfully or unsuccessfully).
Transaction Count
The number of transactions currently being processed by the proxy distributor.
Hot Row Total
Total number of hot rows ever handled by the proxy distributor.
TID
ID of the proxy agent thread running on the primary server. This ID is created by the proxy distributor to handle work from the session on the secondary server.
Flags
Flags of the proxy agent thread.
Proxy ID
The ID of the proxy distributor on behalf of which the proxy agent thread (TID) is running.
Source SessID
The ID of the user's session on the secondary server.
Proxy TxnID
The number of the current transaction. These numbers are unique to the proxy distributor.
Current® Seq
The sequence number of the current operation in the current transaction.
sqlerrno
The error number of any SQL error (or 0 if no errors).
iserrno
The error number of any ISAM or RSAM error (or 0 if no errors).

Example output using the onstat -g proxy all command on a secondary server

Figure 4. **onstat -g proxy all** command output (run from secondary server)

```
Primary            Proxy      Reference Transaction  Hot Row
Node               ID         Count     Count        Total
nagpur             2619       5         2            0

Session  Session  Proxy    Proxy    Proxy    Current  Pending  Reference
         Ref      Proxy_id TID      TxnID    Seq      Ops      Count
21       2        2619     94       1        29       1        1
22       2        2619     95       2        68       1        1
```

Output description

Primary Node
Name of the primary server.
Proxy ID
ID of the proxy distributor. Proxy IDs are unique within a high-availability cluster.
Reference Count
Indicates the number of threads that are using the information for the current transaction. When the count becomes 0, the transaction processing is complete (either successfully or unsuccessfully).
Transaction Count
The number of transactions currently being processed by the proxy distributor.
Hot Row Total
Total number of hot rows ever handled by the proxy distributor. A hot row is a row on a secondary server that is updated multiple times by more than one client. When a row is updated multiple times, the secondary server reads the before image from the primary server by placing an update lock on the row if the most recent update operation from a different session is not replayed on the secondary server.
Session
The session ID
Proxy ID
The ID of the proxy distributor on behalf of which the proxy agent thread (TID) is running.
Proxy TID
Transaction ID of the proxy agent thread running on the primary server. This ID is created by the proxy distributor to handle work from the secondary server session.
Proxy TxnID
The number of the current transaction. These numbers are unique to the proxy distributor.
Current Seq
The sequence number of the current operation in the current transaction.
Pending Ops
The number of operations buffered on the secondary server that have not yet been sent to the primary server.
Reference Count
Indicates the number of threads that are using the information for the current transaction. When the count becomes 0, the transaction processing is complete (either successfully or unsuccessfully).

Example output using the *proxy_id* option on a secondary server

This command returns information only on a secondary server.

Figure 5. **onstat -g proxy proxy_id** command output (run from secondary server)

```
Proxy     Reference Pending   ProxySID
TxnID     Count     Ops
1         1         1         3
2         1         1         4
```

Output description

Proxy TxnID
The number of the current transaction. These numbers are unique to the proxy distributor.
Reference Count
Indicates the number of threads that are using the information for the current transaction. When the count becomes 0, the transaction processing is complete (either successfully or unsuccessfully).
Pending Ops
The number of operations buffered on the secondary server that have not yet been sent to the primary server.
Proxy SID
Proxy session ID.

Example output using the *proxy_id proxy_transaction_id* options on a secondary server

This command returns information only on a secondary server.

Figure 6. **onstat -g proxy_id proxy_transaction_id** command output (run from secondary server)

```
Sequence  Operation rowid    Table                          sqlerrno
Number    Type              Name
28        *Update   526      stores_demo:nilesho.customer   0
```

Output description

Sequence Number
The number of the operation.
Operation Type
The type of operation to be performed. One of: Insert, Update, Delete, Other.
rowid
The row ID of the row in which to apply the operation.
Table Name
The full table name, trimmed to fit a reasonable length. Format: database.owner.tablename
sqlerrno
The error number of any SQL error (or 0 if no errors).

Example output using the *proxy_id proxy_transaction_id sequence_number* options on a secondary server

This command returns information only on a secondary server.

The output fields are the same as the output fields displayed for the **onstat -g proxy_id proxy_transaction_id** command. While the **onstat -g proxy_id proxy_transaction_id** command displays details for a transaction, the **onstat -g proxy_id proxy_transaction_id sequence_number** displays details for all transaction operations.

Figure 7. **onstat -g proxy_id proxy_transaction_id sequence_number** command output (run from secondary server)

```
s
Proxy    Reference Pending  ProxySID
TxnID    Count     Ops
61       0         3        22

onstat -g proxy 2788 61


Sequence Operation rowid    Table                          sqlerrno
Number   Type              Name
960      Update    264      stores_demo:nilesho.customer      0
961      Update    265      stores_demo:nilesho.orders        0
962      Update    266      stores_demo:nilesho.items         0

onstat -g proxy 2788 61 962


Sequence Operation rowid    Table                          sqlerrno
Number   Type              Name
962      Update    266      stores_demo:nilesho.items         0
```


onstat -g qst command: Print wait options for mutex and condition queues

Use the **onstat -g qst** command to display the wait statistics for mutex queues and condition queues (queues of waiters for a mutex or a condition).

The QSTATS configuration parameter must be set to `1` to enable the collection of statistics. For more information, see QSTATS configuration parameter.

Syntax:

```
>>-onstat-- -g--qst-----------------------------------------><
```

Example output

Figure 1. **onstat -g qst** command output

```
Mutex Queue Statistics
name      nwaits    avg_time max_time avgq maxq nservs    avg_time

ddh chai 1         1354863  1354863  1    1    56        1690

Condition Queue Statistics
name      nwaits    avg_time max_time avgq maxq nservs    avg_time

arrived  1         110008   110008   1    1    0         0
logbf0   21        642      4431     1    2    0         0
logbf1   15        475      2519     1    2    0         0
logbf2   19        596      3274     1    2    0         0
bp_cond  1         0        0        1    1    0         0
```

Output description

name (string)
: Name of the mutex or condition resource being waited for

nwaits (decimal)
: Number of times this resource was waited for

avg_time (decimal)
: Average time spent waiting (in microseconds)

max_time (decimal)
: Maximum time spent waiting (in microseconds)

avgq (decimal)
: Average length of the queue

maxq (decimal)
: Maximum length of the queue

nservs (decimal)
: Number of times this resource was acquired

avg_time (decimal, microsecond)
: Average time the resource was held per acquisition (in microseconds)

Related reference:
QSTATS configuration parameter

onstat -g rah command: Print read-ahead request statistics

Use the **onstat -g rah** command to display information about read-ahead requests.

Syntax:

```
>>-onstat-- -g--rah-------------------------------------------><
```

Example output

Figure 1. onstat -g rah command output

```
Read Ahead

# Qs                  1
# threads             2
# Requests            58690
# Continued                   0
# Memory Failures     0
Last Thread Add       04/06/2013.14:34
Way behind            0

Partition ReadAhead Statistics

           Buffer    Disk    Hit     Data         Index          Idx/Dat      Log/PageList Last Committed
Partnum    Reads     Reads   Ratio   # Reqs Eff # Reqs   Eff   # Reqs   Eff # Pages Eff   # Reqs   Eff    # Resch
0x200003   4312677   110     99      0      0   0        0     0        0     0      0      12906    100    0
0x300002   23740584  1427    99      0      0   0        0     0        0     0      0      6681     100    7
0x400002   17818942  966     99      0      0   0        0     0        0     0      0      25849    100    57
```

Output description

Qs
: Number of queues for read-ahead requests

threads
: Number of read-ahead threads

Requests
: Number of read-ahead requests

Continued
: Number of times a read-ahead request continued to occur

Memory Failures
: Number of failed requests because of insufficient memory

Last Thread Add
: Date and time when the last read-ahead thread was added

Way behind
: How many page list requests were dropped because the read-ahead daemon is too far behind

Partnum
: Partition number

Buffer reads
: Number of bufferpool and disk pages that were read

Disk Reads
: Number of pages that were read from disk

Hit Ratio
: Cache hit ratio for the partition

Reqs
: Number of read ahead requests. (There are 5 instances of this output field: for data, the index, index data, log pages, and last committed rows.)

Eff
: Efficiency of the read-ahead requests. This is the ratio been the number of pages requested by read-ahead operations to the number of pages that were already cached and for which a read-ahead operations was not needed. Values are between 0 and 100. A higher number means that read ahead is beneficial. (There are 5 instances of this output field: for data, the index, index data, log pages, and last committed rows.)

Resch
: The number of requests for last committed rows that are rescheduled because the updates to a multi-piece row are not complete.

onstat -g rbm command: Print a block map of shared memory

Use the **onstat -g rbm** command to display a hexadecimal bitmap of the free and used blocks within the resident segment of shared memory.

Syntax:

```
>>-onstat-- -g--rbm-------------------------------------------><
```

Example output

Figure 1. **onstat -g rbm** command output

```
Block bitmap for resident segment address 0x44000000:
 address = 0x440003bc, size(bits) = 3035
 used = 3031, largest_free = 4

      0:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
    256:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
    512:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
    768:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   1024:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   1280:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   1536:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   1792:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   2048:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   2304:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   2560:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   2816:ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffe00
```

Output description

Header

address (hex)
: In-memory starting address of the used/free blocks in the segment

size (bits)
: Number of bits in the block bitmap; each bit represents one block

used (blocks)
: Used blocks in the bitmap

largest_free (blocks)
: Largest run of free blocks

Data

Bit number (decimal): data (hex)
: Bit number followed by 32 bytes of data (hex)

onstat -g rea command: Print ready threads

Use the **onstat -g rea** command to display information about the virtual processor threads whose current status is ready.

Syntax:

```
>>-onstat-- -g--rea-------------------------------------------><
```

Example output

Following is sample output from the **onstat -g rea** command. For a description of the output, see onstat -g ath command: Print information about all threads.

Figure 1. **onstat -g rea** command output

```
Ready threads:
tid      tcb      rstcb    prty     status    vp-class    name
6        536a38   406464   4        ready        3cpu     main_loop()
28       60cfe8   40a124   4        ready        1cpu     onmode_mon
33       672a20   409dc4   2        ready        3cpu     sqlexec
```


onstat -g rss command: Print RS secondary server information

Use the **onstat -g rss** commands to display information about remote standalone secondary servers.

Syntax:

```
>>-onstat-- -g--rss--+-------------+--------------------------><
                    +-verbose-----+
                    +-log---------+
                    '-server_name-'
```

The output of the **onstat -g rss** command differs slightly depending on whether the command is run on the primary server or on the RS secondary server.

Invocation	Explanation
onstat -g rss	Displays brief RS secondary server information
onstat -g rss verbose	Displays detailed RS secondary server information
onstat -g rss log	Displays log information. This command is only applicable when run on the primary server.
onstat -g rss *server_name*	Displays information about a specific RS secondary server. This command is only applicable when run on the primary server.

Example output (primary server)

Figure 1. **onstat -g rss verbose** command output, when the command is run on the primary server.

```
Local server type: Primary
Index page logging status: Enabled
Index page logging was enabled at: 2020/05/23 06:12:06
Number of RSS servers: 2

RSS Server information:

RSS Server control block: 0x64f64758
RSS server name: rahulb_3
RSS server status: Active
RSS connection status: Connected
RSS flow control:384/352
Log transmission status: Active
Next log page to send(log id,page): 6,36
Last log page acked(log id,page): 6,35
Last log page applied(log id,page): 6,35
Time of Last Acknowledgement: 2020-05-23.06:13:29
Pending Log Pages to be ACKed: 0
Approximate Log Page Backlog:0
Sequence number of next buffer to send: 231
Sequence number of last buffer acked: 230
Supports Proxy Writes: N
Total number of delay(s): 8
Time of last delay: 2020-05-23.06:13:02
```

Output description (primary server)

Local server type
: Primary or RSS (remote standalone secondary) server type

Index page logging status
: Displays whether index page logging is enabled or disabled between primary server and secondary server

Index page logging was enabled at
: Date and time that index page logging was enabled

Number of RSS servers
: Number of RS secondary servers connected to the primary server

RSS Server control block
: RS secondary server control block

RSS Server name
: Name of RS secondary server

RSS Server status
: Displays whether RS secondary server is active or not

RSS flow control
: Values, in number of logical log pages, determining when flow control is enabled or disabled, respectively.

RSS Connection status
: Connection status of RS secondary server

Log transmission status
: Displays whether log transmission is active or inactive

Next log page to send (log id, page)
: The log ID and page number of the next log page that will be sent

Last log page acked (log id, page)
: The log ID and page number of the last acknowledged log

Last log page applied (log id, page)
: The log ID and page number of the last applied log

Time of Last Acknowledgment
: The time at which the last log was acknowledged

Pending Log pages to be ACKed
: The number of logs sent but not yet acknowledged

Approximate Log Page Backlog
: The difference between the number of logs that were sent and the end of the logical log

Sequence number of next buffer to send
: The sequence number of the next buffer to be sent

Sequence number of last buffer acked
: The sequence number of the last acknowledged buffer

Supports Proxy Writes
: Displays whether the server is currently configured to allow updates to secondary servers. **Y** = supports updates to secondary servers, **N** = does not support updates to secondary servers.

Total number of delay(s)

The total number of times the flow delay occurred.

Time of last delay
: The time of last delay in flow control.

Example output with log option (primary server)

Figure 2. **onstat -g rss log** command output, when the command is run on the primary server.

```
Log Pages Snooped:
RSS Srv          From      From      Tossed
name              Cache     Disk     (LBC full)

cdr_ol_nag_1_c1    1368       1331       0
cdr_ol_nag_1_c2    1357       1342       0
cdr_ol_nag_1_c3    1356       1343       0
```

Output description with log option (primary server)

Log Pages Snooped
: Statistics for each RS secondary server

RSS Srv name
: RS secondary server name

From Cache
: From cache number

From Disk
: Log from disk

Tossed (LBC full)
: Number of log pages that were discarded as a result of the LBC becoming full

Example output (RS secondary server)

Figure 3. **onstat -g rss** command output, when the command is run on the RS secondary server.

```
Local server type: RSS
Server Status: Active
Source server name: cdr_ol_nag_1
Connection status: Connected
Last log page received(log id,page): 7,877
```

Output description (RS secondary server)

Local server type
: Primary or RSS (remote standalone secondary) server type

Server Status
: Displays whether RS secondary server is active

Source server name
: Name of the primary server

Connection status
: Connection status of RS secondary server

Last log page received (log id,page)
: Most recent log ID and page received

Example output with verbose option (RS secondary server)

Figure 4. **onstat -g rss verbose** command output, when the command is run on the RS secondary server.

```
RSS Server control block: 0x45a3fe58
Local server type: RSS
Server Status: Active
Source server name: my_server
Connection status: Connected
Last log page received(log id,page): 10,1364
Sequence number of last buffer received: 489
Sequence number of last buffer acked: 489
Delay Apply: Configured (3)
Stop Apply: Not configured.
Delay or Stop Apply control block: 0x45a40ba8
    Pending pages: 7
    Last page written: (10:1372).
    Next page to read: (10:1366).
    Delay or Stop Apply thread: Running.
```

Output description with verbose option (RS secondary server)

RSS Server control block
: The server control block.

Local server type
: The local server's type.

Server Status
: The status of the RS secondary server.

Source server name
: The name of the primary server in the RS secondary server's high-availability cluster.

Connection status
: The status of the connection between the RS secondary server and the cluster's primary server.

Last log page received (log id,page)
: The log ID and page number of the last log acknowledged by the RS secondary server.

Sequence number of last buffer received
: The sequence number of the last buffer that was received by the RS secondary server.

Sequence number of last buffer acked
: The sequence number of the last buffer acknowledged by the RS secondary server.

Delay Apply
: Whether delay apply is configured or not. The delay value, in seconds, is included in parentheses.

Stop Apply
: Whether stop apply is configured or not. The stop value, which is enclosed in parentheses, is either 1 or a Unix time.

Delay or Stop Apply control block
: The control block of the delay or the stop apply.

Pending pages
: The number of pages that are waiting to be written to the log-staging directory.

Last page written
: The log id and page number of the log that was most recently written to the log-staging directory.

Next page to read
: The log id and page number of the next log to write to the log-staging directory.

Delay or Stop Apply thread
: The status of the delay-apply or stop-apply thread.

onstat -g rwm command: Print read and write mutexes

Use the **onstat -g rwm** command to display information about read, write, and waiting mutex threads, and to list the addresses of the tickets that these threads have acquired.

Syntax:

```
>>-onstat-- -g--rwm-----------------------------------------><
```

Example output

Figure 1. **onstat -g rwm** command output

```
MUTEX   NAME     write/read/wait    tcb list
<address> <name>     first mutex
       Writer     ticket = <ticket address>   tcb=<thread address> <thread name>
       Readers    ticket = <ticket address>   tcb=<thread address> <thread name>
       Waiters    ticket = <ticket address>   tcb=<thread address> <thread name>
<address> <name>        second mutex
       Writer     ticket = <ticket address>   tcb=<thread address> <thread name>
       Readers     ticket = <ticket address>  tcb=<thread address> <thread name>
       Waiters     ticket = <ticket address>  tcb=<thread address> <thread name>
....
....
....
<address> <name>       last mutex
       Writer     ticket = <ticket address>   tcb=<thread address> <thread name>
       Readers    ticket = <ticket address>   tcb=<thread address> <thread name>
       Waiters    ticket = <ticket address>   tcb=<thread address> <thread name>
```

Output description

tcb
: List of thread addresses

Writer
: List of write threads

Readers
: List of read threads

Waiters
: List of waiting threads

ticket
: Address of ticket acquired by the thread

onstat -g sch command: Print VP information

Use the **onstat -g sch** command to display information about thread migration and the number of semaphore operations, spins, and busy waits for each virtual processor.

Syntax:

```
>>-onstat-- -g--sch---------------------------------------------><
```

Example Output

Figure 1. **onstat -g sch** command output

```
VP Scheduler Statistics:
 vp    pid       class       semops    busy waits  spins/wait
 1     3284      cpu         23997     0           0
 2     1340      adm         0         0           0
 3     4624      lio         2         0           0
 4     3320      pio         2         0           0
 5     6076      aio         7710      0           0
 6     4580      msc         46        0           0
 7     3428      soc         7         0           0
 8     2308      soc         1         0           0

Thread Migration Statistics:
 vp  pid      class steal-at steal-sc idlvp-at idlvp-sc inl-polls Q-ln
 1   3284     cpu   0        0        0        0        0         0
 2   1340     adm   0        0        0        0        0         0
 3   4624     lio   0        0        0        0        0         0
 4   3320     pio   0        0        0        0        0         0
 5   6076     aio   0        0        0        0        0         0
 6   4580     msc   0        0        0        0        0         0
 7   3428     soc   0        0        0        0        0         0
 8   2308     soc   0        0        0        0        0         0
```


onstat -g scn command: Print scan information

Use the **onstat -g scn** command to display the status of a current scan and information about the scan.

If you have a long-running scan, you might want to use this command to check the progress of the scan, to determine how long the scan will take before it completes, and to view information about the scan. For tables, the **onstat -g scn** command output identifies whether a scan is a light or bufferpool scan.

Syntax:

```
>>-onstat-- -g -scn---------------------------------------------><
```

Example Output

Figure 1. **onstat -g scn** output showing table information

```
Light Scan Info
descriptor  address           next_lpage  next_ppage        ppage_left  bufcnt  look_aside

RSAM batch sequential scan info

SesID Thread Partnum  Rowid    Rows Scan'd Scan Type Lock Mode  Notes
48    68     10016e   12bb09   43146       Light     Table      Look aside,
40    47     100106   101      0           Buffpool  +Test      Must copy
```

Information about an index scan is valid when a scan is running.

Figure 2. **onstat -g scn** output showing index scan information

```
RSAM batch index scan info

SesID Thread Partnum  Scan Type Lock Mode  Notes
136   156    100197             SLock+Test
       Start Key    GT    :-2147483648:
       Stop Key     EQ    :1500:
       Current key        :170:
       Current position: buffp 0x10a4bc0c8 pagenum 2 slot 17 rowoff 4 flags 0
```

Output Description

descriptor (decimal)
: Light scan ID

address (hex)
: Memory address of the light scan descriptor

next_lpage (hex)
: Next logical page address to scan

next_ppage (hex)
: Next physical page address to scan

ppage_left (decimal)
: Number of physical pages left to scan in the current extent

bufcnt
: Number of light scan buffers used for this light scan

look_aside
: Whether look aside is needed for this light scan (`Y` = yes, `N` = no). Look asides occur when a thread needs to examine the buffer pool for existing pages to obtain the latest image of a page being light scanned.

SesID
: Session ID

Thread
: Thread ID

Partnum
: Partition number

Rowid
: Current® row ID

Rows Scan'd
: Number of rows that have been scanned

Scan Type
: For tables, either:

  - `Bufferpool`
  - `Light` (light scan)

  For indexes, either:

  - `key only`
  - No value if the scan is not a key-only scan

Lock Mode
: The type of acquired lock or no lock:

  - `Table` (table-level lock acquired)
  - `Slock` (share locks acquired)
  - `Ulock` (update locks acquired)
  - blank (no locks acquired)

  This column can also show one of the following values:

  - `+Test` (The scan tested for a conflict with the specified lock type; the lock was not acquired.)
  - `+Keep` (The acquired locks will be held until end of session instead of the end of the transaction.)

Notes
: This column can show one of the following values:

  - `Look aside`
    The light scan is performing look aside.

    The light scan reads blocks of pages directly from disk into large buffers, rather than getting each page from the buffer manager. In some cases, this process requires the light scan to check the buffer pool for the presence of each data page that it processes from one of its large buffers; this process is called *look aside*. If the page is currently in the buffer pool, the light scan will use that copy instead of the one in the light scan large buffer. If the page is not in the buffer pool, the light scan will use the copy that the light scan read from disk into its large buffer. If the light scan is performing look aside, the performance of the scan is slightly reduced.

    In many cases, the light scan can detect that it is impossible for the buffer pool to have a newer version of the page. In these situations, the light scan will not check the buffer pool, and the look aside note will be absent.

  - `Forward row lookup`
    The server is performing a light scan on a table that has rows that span pages. The light scan must access and use the buffer pool to get the remainder pieces of any rows that are not completely on the home page.

Start key
: Start key of the scan

Stop key
: End key of the scan

Current key
: The current key in the scan

Current position
: The current location of the scan in the index, for example, the page, slot, and offset

---

onstat -g sds command: Print SD secondary server information

Use the **onstat -g sds** command to display information about shared-disk secondary servers.

```
Syntax:

>>-onstat-- -g -sds -+-----------------+----------------------><
                    '-+-server_name-+-'
```

```
'-verbose-----'
```

The output of the **onstat -g sds** command differs slightly depending on whether the command is issued on the primary server or on the SD secondary server.

Invocation	Explanation
onstat -g sds	Displays brief SD secondary server information
onstat -g sds verbose	Displays detailed SD secondary server information
onstat -g sds *server_name*	Displays information about a specific SD secondary server. When *server_name* is specified, the command must be issued from the primary server.

Example output (primary server)

Figure 1. **onstat -g sds** command output when you run the command from primary server.

```
Local server type: Primary
Number of SDS servers:1

SDS server information

SDS srv      SDS srv      Connection        Last LPG sent  Supports
name         status       status            (log id,page)  Proxy Writes
C_151162     Active       Connected            554,4998    Y
```

Output description (primary server)

Local server type
: Primary or SDS (shared disk secondary) server type

Number of SDS servers
: Number of SD secondary servers connected to the primary server

SDS Srv name
: Name of SD secondary server

SDS Srv status
: Displays whether SD secondary server is active

Connection status
: Displays whether SD secondary server is connected

Last LPG sent (log id, page)
: Most recent LPG log ID and page

Supports Proxy Writes
: Displays whether the server is currently configured to allow updates to secondary servers. **Y** = supports updates to secondary servers, **N** = does not support updates to secondary servers.

Example output with verbose option (primary server)

Figure 2. **onstat -g sds *server_name*** command output when you run the command from primary server.

```
Number of SDS servers:2
Updater node alias name :server_1

SDS server control block: 0x46217640
server name: rahulb_4
server type: SDS
server status: Active
connection status: Connected
Last log page sent(log id,page):6,44
Last log page flushed(log id,page):6,44
Last log page acked (log id, page):6,44
Last LSN acked (log id,pos):6,180664
Last log page applied(log id,page): 6,44
Approximate Log Page Backlog:0
Current SDS Cycle:19
Acked SDS Cycle:19
Sequence number of next buffer to send: 447
Sequence number of last buffer acked: 444
Time of last ack:2020/05/23 06:16:28
Supports Proxy Writes: N
Time of last received message: 2020/05/23 06:16:49
Time of last alternate write: N/A
Time of last alternate read : N/A
Total number of delay(s): 11
Time of last delay: 2020/05/23 06:13:04
```

Output description with verbose option (primary server)

Number of SDS servers
: The number of SD secondary servers that share disk space with the primary server

Updater node alias name
: The name of the primary server

SDS server control block
: SD secondary server control block

server name
: The name of the server

server type
The type of server
server status
Displays whether the server is active or inactive
connection status
Status of connection between primary and secondary server
Last log page sent (log id, page)
Log ID and page of most recent log page sent
Last log page flushed (log id, page)
Log ID and page of the most recent log page flushed
Last log page acked (log id, pos))
Most recent log page acknowledged
Last LSN acked (log id, pos)
Most recent log sequence number that was acknowledged
Last log page applied(log id,page)
The log ID and page number of the last applied log
Approximate Log Page Backlog
The number of logs waiting to be sent
Current® SDS Cycle
Used internally by support to monitor coordination of the primary server with the SDS server
Acked SDS Cycle
Used internally by support to monitor coordination of the primary server with the SDS server
Sequence number of next buffer to send
Sequence number of next buffer to send
Sequence number of last buffer acked
Sequence number of next buffer acknowledged
Time of last ack
Date and time of last log acknowledgment
Supports Proxy Writes
Displays whether the server is currently configured to allow updates to secondary servers. **Y** = supports updates to secondary servers, **N** = does not support updates to secondary servers.
Time of last received message:
The timestamp of the current server's most recently received from another server.
Time of last alternate write
The timestamp of the current server's most recent write to the blobspace specified by the SDS_ALTERNATE configuration parameter.
Time of last alternate read
The timestamp of the current server's most recent read from the blobspace specified by the SDS_ALTERNATE configuration parameter.
Total number of delay(s)
The total number of times the flow delay occurred.
Time of last delay
The time of last delay in flow control.

Example output with verbose option (SD secondary server)

Figure 3. **onstat -g sds verbose** command output when you run the command from the SD secondary server.

```
SDS server control block: 0xb299880
Local server type: SDS
Server Status : Active
Source server name: my_source_server
Connection status: Connected
Last log page received(log id,page): 7,884
Next log page to read(log id,page):7,885
Last LSN acked (log id,pos):7,3621272
Sequence number of last buffer received: 0
Sequence number of last buffer acked: 0
Current paging file:/dbspaces/page_my_source_server_sdc1_
Current paging file size:2048
Old paging file:/dbspaces/page_my_source_server_sdc1_
Old paging file size:10240
```

Output description with verbose option (SD secondary server)

SDS server control block
SD secondary server control block
Local server type
Primary or SDS (shared disk secondary) server type
Server status
Displays whether SD secondary server is active
Source server name
Displays name of primary server
Connection status
Displays whether SD secondary server is connected
Last log page received (log id, page)
Most recent log page received
Next log page to read (log id,page)
Next log page in sequence to read
Last LSN acked (log id,pos)
Most recent LSN acknowledged
Sequence number of last buffer received
Sequence number of last buffer received

Sequence number of last buffer acked
: Sequence number of last buffer acknowledged

Current paging file
: Name of current paging file

Current paging file size
: Size of current paging file

Old paging file
: Name of previous paging file

Old paging file size
: Size of previous paging file

onstat -g seg command: Print shared memory segment statistics

Use the **onstat -g seg** command to show the statistics for shared memory segments.

This command shows how many segments are attached and their sizes. You can run the **onstat -g seg** command on a dump file that was created without the buffer pool.

```
Syntax:

>>-onstat-- -g--seg-------------------------------------------><
```

Example output

Figure 1. onstat -g seg command output

```
Segment Summary:
id          key         addr     size       ovhd     class blkused  blkfree
720914    52e44801   44000000 4390912    248812  R    1072     0
753683    52e44802   44430000 131072000  769136  V    22573    9427
819221    52e44803   4c130000 66027520   1       B    16120    0
851990    52e44804   50028000 83648512   1       B    20422    0
Total:    -           -       285138944 -       -    60187    9427
Virtual segment low memory reserve (bytes):4194304
Low memory reserve used 0 times and used maximum block size 0 bytes
```

Output description

id
: The ID of the shared memory segment

key
: The shared memory key that is associated with the shared memory segment ID

addr
: The address of the shared memory segment

size
: The size of the shared memory segment in bytes

ovhd
: The size of the shared memory segment control information (overhead) in bytes

class
: The class of the shared memory segment (B is for Buffferpool, R is for Resident, V is for Virtual, VX is for Virtual Extended, and M is for Message.)

blkused
: The number of blocks of used memory

blkfree
: The number of blocks of free memory

Virtual segment low memory reserve (bytes)
: The size of reserved memory for use when critical activities are needed and the server has limited free memory, specified in bytes (You specify reserved memory in the LOW_MEMORY_RESERVE configuration parameter.)

Low memory reserve used 0 times and used maximum block size 0 bytes)
: The number times that the server used the reserved memory and the maximum memory needed

Related reference:
SHMADD configuration parameter
SHMBASE configuration parameter
SHMVIRTSIZE configuration parameter
LOW_MEMORY_RESERVE configuration parameter
EXTSHMADD configuration parameter
Running onstat Commands on a Shared Memory Dump File

onstat -g ses command: Print session-related information

Use the **onstat -g ses** command to display information about the session.

By default, only the DBSA can view **onstat -g ses** information. However, when the UNSECURE_ONSTAT configuration parameter is set to `1`, all users can view this information.

Syntax:

```
>>-onstat-- -g--ses--+------------+--------------------------><
                     '-session_id-'
```

You can specify one of the following invocations.

onstat -g ses
Displays a one-line summary for each session

onstat -g ses *session_id*
Displays information for a specific session

Example output for all sessions

Figure 1. **onstat -g ses** *command output*

```
session                                          #RSAM    total      used       dynamic
id       user     tty       pid      hostname   threads  memory     memory     explain
24       informix -         0        -          0        12288      7936       off
23       informix -         17602    carson     1        57344      48968      off
3        informix -         0        -          0        12288      9168       off
2        informix -         0        -          0        12288      7936       off

Last 20 Sessions Terminated

Ses ID  Username  Hostname  PID    Time               Reason
46      user_1    host_1    21220  01/19/2015.15:20   session limit txn time (60s)
43      user_1    host_1    21340  01/19/2015.15:14   session limit memory (5124 KB)
61      user_1    host_1    21404  01/19/2015.15:04   session limit logspace (10242 KB)
64      user_1    host_1    21458  01/19/2015.15:02   session limit txn time (39548 KB)
```

Output description: session section

Session id
The session ID

user
The user who started the session

tty
The tty that is associated with the front end for this session

pid
The process ID associated with the front end for this session

hostname
The hostname from which this session connected

#RSAM threads
The number of RSAM thread that is allocated for this session

total memory
The amount of memory that is allocated for this session

used memory
The amount of memory that is actually used by this session

dynamic explain
Generate explain output of the SQL statements of the session (`on` or `off`)

Output description: Last 20 Sessions Terminated section

Ses ID
The session ID

Username
The user who started the session

Hostname
The hostname from which this session connected

PID
The process ID associated with the front end for this session

Time
The time at which the session was terminated.

Reason
The limit that was exceeded, followed by the limit value in parentheses.

Example output for a specific session

Figure 2. **onstat -g ses** *session_id* command output for a completed SQL statement

```
session          effective                      #RSAM    total    used     dynamic
id      user     user       tty   pid    hostname threads  memory   memory   explain
53      informix -          36    18638  apollo11 1        73728    63048    off

Program :
/usr/informix/bin/dbaccess
```

```
tid      name     rstcb            flags    curstk   status
77       sqlexec  4636ba20         Y--P---  4240     cond wait  sm_read    -

Memory pools    count 1
name          class addr              totalsize  freesize   #allocfrag #freefrag
53            V     4841d040          73728      10680      84          6

name            free       used            name            free       used
overhead        0          3288            scb             0          144
opentable       0          2904            filetable       0          592
log             0          16536           temprec         0          2208
gentcb          0          1656            ostcb           0          2920
sqscb           0          21296           sql             0          72
hashfiletab     0          552             osenv           0          2848
sqtcb           0          7640            fragman         0          392

sqscb info
scb               sqscb             optofc   pdqpriority optcompind  directives
481b70a0          483e2028          0        0           0           1

Sess       SQL             Current            Iso Lock       SQL  ISAM F.E.
Id         Stmt type       Database           Lvl Mode       ERR  ERR  Vers  Explain
53         -               sysmaster          CR  Not Wait   0    0    9.24  Off

Last parsed SQL statement :
  Database 'sysmaster@lx1'

Xadatasources participated in this session :
Xadatasource name                                        RMID       Active
xabasicdb@atmol10:sitaramv.xads_t3_i1                       6        YES
xabasicdb@atmol10:sitaramv.xads_t2_i1                       4        YES
xabasicdb@atmol10:sitaramv.xads_t1_i3                       3        YES
xabasicdb@atmol10:sitaramv.xads_t1_i2                       2        YES
xabasicdb@atmol10:sitaramv.xads_t1_i1                       1        YES
xabasicdb@atmol10:sitaramv.xads_t2_i2                       5        NO

DRDA client info
           Userid:
           Wrkstnname:   nemea
           Applname:     db2jcc_application
           Acctng:       JCC03510nemea
           Programid:
           Autocommit:
           Packagepath:

Session Limits
                Limit     Current
Locks           10000     1
Memory(KB)      5120      72
Temp Space(KB)  30720     0
Log Space(KB)   10240     0
Txn Time(s)     120       0
```

Output description: program section

Displays the full path of the client program that is used in your session. Use the client program information to monitor or stop access to the database.

Output description: threads section

Although this section has no title, the following output displays information about threads.

tid
: The thread ID

name
: The name of the thread

rstcb
: RSAM control block

flags
: Describes the status of the thread using the following codes:
  Position 1

  B
  : Waiting on a buffer

  C
  : Waiting on a checkpoint

  G
  : Waiting on a logical-log buffer write

  L
  : Waiting on a lock

  S
  : Waiting on a mutex

  T
  : Waiting on a transaction

  X
  : Waiting on a transaction cleanup

  Y
  : Waiting on a condition

Position 2

*
: An asterisk in this position means that the thread encountered an I/O failure in the middle of a transaction

Position 3

A
: Archive thread

B
: Begin work

P
: Begin Prepare or Prepared work

X
: XA prepared

C
: Committing or committed

R
: Aborting or aborted

H
: Heuristically aborted or heuristically rolling back

Position 4

P
: Primary thread

Position 5

R
: Reading

X
: Critical section

Position 6

R
: Recovery thread

Position 7

M
: Monitor thread

D
: Daemon thread

C
: Cleaner

F
: Flusher

B
: B-tree scanner

curstk
: Current® stack size

status
: Current thread status

Output description: memory pools header section

The information is repeated for each session pool.

name
: Name of pool

class
: Class of the memory where the pool is allocated from. R is for Resident, V is for Virtual, and M is for Message

addr
: Address of the pool structure

totalsize
: Total size of the memory that is acquired by the pool (in bytes)

freesize
: Number of bytes free in the pool

#allocfrag
: Number of allocated memory fragments in the pool

#freefrag
: Number of free fragments in the pool

Output description: Memory pools section

name
: Name of a component which allocated memory from the pool

free
: Number of bytes freed

used
: Number of bytes allocated

Output description: sqscb info section

scb
: The session control block. This is the address of the main session structure in shared memory

sqscb
: SQL level control block of the session

optofc
: The current value of the **OPTOFC** environment variable or ONCONFIG configuration file setting

pdqpriority
: The current value of the **PDQPRIORITY** environment variable or ONCONFIG configuration file setting

optcompind
: The current value of the **OPTCOMPIND** environment variable or ONCONFIG configuration file setting

directives
: The current value of the **DIRECTIVES** environment variable or ONCONFIG configuration file setting

Output description: SQL section

Displays SQL information for the specified session. This section contains the same information that is output from the **onstat -g sql** command. See onstat -g sql command: Print SQL-related session information.

Output description: Last parsed SQL statement section

The Last parsed SQL statement section contains the same information that is output from the **onstat -g sql** command. See onstat -g sql command: Print SQL-related session information.

Output description: Xadatasources participated in this session section

The Xadatasources participated in this session section shows information about the XA data sources that are available during the session, their resource manager identifiers, and whether they are currently active.

Xdatasource name
: The XA data source that participated in the session

RMID
: The identifier of the resource manager for the corresponding XA data source

Active
: Whether the XA data source is still active

Output description: DRDA client info section

The **DRDA client info** section shows information about Distributed Relational Database Architecture™ (DRDA) connections to clients.

Userid
: User ID of the client user

Wrkstnname
: Name of the client workstation

Applname
: Name of the client application, for example `db2jcc_application`

Acctng
: Accounting string from the client, for example `JCC03510nemea`

Programid
: Client program identifier (not used by Informix®)

Autocommit
: Default transaction autocommit mode for Informix data sources

Packagepath
: Client package path (not used by Informix)

Output description: Session limits section

Locks
: The session's number of locks.

Memory(KB)
: The session's memory.

Temp Space(KB)
: The session's temporary table space.

Log Space(KB)
: Log space for single transactions.

Txn Time(s)
: Duration of single transactions.

Figure 3. **onstat -g ses** *session_id* command output for a running SQL statement

```
session           effective                      #RSAM    total    used     dynamic
id       user     user      tty  pid   hostname   threads  memory   memory   explain
37       informix -         9    13965 apollo8    1        327680   308200   off

Program :
/usr/informix/bin/dbaccess

tid      name     rstcb            flags    curstk   status
44       sqlexec  44e5b350         ---P---  4320     running-

Memory pools    count 2
name         class addr              totalsize  freesize   #allocfrag #freefrag
37           V     8c64c040          323584     18736      199        21
37*00        V     8c756040          4096       744        1          1

name              free       used          name              free       used
overhead          0          6704          scb               0          144
opentable         0          5968          filetable         0          768
log               0          16536         temprec           0          22688
keys              0          216           ralloc            0          194672
gentcb            0          1592          ostcb             0          2992
sqscb             0          27880         sql               0          13384
hashfiletab       0          552           osenv             0          2672
sqtcb             0          9664          fragman           0          728
sapi              0          240           udr               0          272
rsam_seqscan      0          528

sqscb info
scb               sqscb             optofc   pdqpriority optcompind  directives
44ef4200          8ac90028          0        0           2           1
Sess       SQL            Current            Iso Lock       SQL  ISAM F.E.
Id         Stmt type      Database           Lvl Mode       ERR  ERR  Vers  Explain
37         SELECT         sysadmin           CR  Not Wait   0    0    9.24  Off

Current statement name : unlcur

Current SQL statement (3) :
  select * from systables, sysindexes, syscolumns

  QUERY_TIMEOUT setting: 00:00:25
  Clock time elapsed   : 00:00:13

Last parsed SQL statement :
  select * from systables, sysindexes, syscolumns
```

The QUERY_TIMEOUT setting and clock time are displayed only for running queries, not for DML or DDL statements or administration operations.

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)

onstat -g shard command: Print information about the shard definition

Use the **onstat -g shard** command to display information about the sharding definition.

Syntax:

```
>>-onstat-- -g--shard-----------------------------------------><
```

Output description

The output of the **onstat -g shard** command shows the following information without field labels.

Sharding definition name
: The name of the sharding definition.

Database name
: The name of the database that contains the table or collection that is distributed across multiple shards.

Table owner name
: The owner of the table or collection that is distributed across multiple shards.

Table name
: The name of the table or collection that is distributed across multiple shards.

Shard key
: The shard key that is used for distributing rows or documents. Value can be a table column, document field, or an expression.

Sharding strategy
: The method for determining which database server a new row or document is applied on. Values can be HASH (hash algorithm), CONSISTENT HASH (consistent hash algorithm), or EXPRESSION (expression).

Sharding type
: Specifies source-server action after a row or document is replicated to a target server. Values can be DELETE, KEEP, or INFORMATIONAL.

Shard optimization
: Specifies if queries can skip shard servers that do not contain relevant data. Values can be ENABLED or NOT ENABLED.

Version column
: Specifies the column or key that is used when Enterprise Replication attempts to verify that a source row or document was not updated. The value is a column or document field.

Sharding rule
: The rule for replicating data to a specific database server.

Example: Output for a sharding definition that uses consistent hash-based sharding

For this example, you have a sharding definition that was created by the following command:

```
cdr define shardCollection collection_1 database_1:john.customers_1
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1
   g_shard_server_2
   g_shard_server_3
```

The following example shows output when the **onstat -g shard** command is run on **g_shard_server_1**, **g_shard_server_2**, or **g_shard_server_3**.

Figure 1. **onstat -g shard** command output for a sharding definition that uses a consistent hash algorithm to distribute data across multiple shard servers.

```
collection_1 database_1:john.customers_1 key:b CONSISTENT HASH:DELETE SHARD OPTIMIZATION:NOT ENABLED
Matching for delete:column_3
   g_shard_server_1 (65542) (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 4019 and 5469)
                            or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5719 and 6123)
                            or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2113 and 2652)
   g_shard_server_2 (65543) (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 6124 and 7415)
                            or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5470 and 5718)
                            or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 7416 and 7873)
   g_shard_server_3 (65544) (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2653 and 3950)
                            or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) >= 7874
                            or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) < 2113
                            or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 3951 and 4018)
```

Each shard server has three hashing partitions.

Example: Output for a sharding definition that uses hash-based sharding

For this example, you have a sharding definition that was created by the following command:

```
cdr define shardCollection collection_1 database_1:josh.customers_1
   --type=delete --key=column_2 --strategy=hash --versionCol=column_3
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
   g_shard_server_D
```

The following example shows output when the **onstat -g shard** command is run on **g_shard_server_A**, **g_shard_server_B**, **g_shard_server_C**, or **g_shard_server_D**.

Figure 2. **onstat -g shard** command output for a sharding definition that uses a hash algorithm to distribute data across multiple shard servers.

```
collection_1 database_1:josh.customers_1 key:column_2 HASH:DELETE SHARD OPTIMIZATION:ENABLED
Matching for delete:column_3
g_shard_server_A (65545) mod(ifx_checksum(column_2::LVARCHAR, 0), 4) = 0
g_shard_server_B (65546) mod(ifx_checksum(column_2::LVARCHAR, 0), 4) in (1, -1)
g_shard_server_C (65547) mod(ifx_checksum(column_2::LVARCHAR, 0), 4) in (2, -2)
g_shard_server_D (65548) mod(ifx_checksum(column_2::LVARCHAR, 0), 4) in (3, -3)
```

Example: Output for a sharding definition that uses expression-based sharding

For this example, you have a sharding definition that was created by the following command:

```
cdr define shardCollection collection_2 database_2:john.customers_2
   --type=keep --key=state --strategy=expression --versionCol=version_column
   g_shard_server_F "IN ('AL','MS','GA')"
   g_shard_server_G "IN ('TX','OK','NM')"
   g_shard_server_H "IN ('NY','NJ')"
   g_shard_server_I REMAINDER
```

The following example shows output when the **onstat -g shard** command is run on **g_shard_server_F**, **g_shard_server_G**, **g_shard_server_H**, or **g_shard_server_I**.

Figure 3. **onstat -g shard** command output for a sharding definition that uses an expression to distribute data across multiple database servers.

```
collection_2 database_2:john.customers_2 key:state EXPRESSION:KEEP SHARD OPTIMIZATION:ENABLED
Matching for delete:version_column
g_shard_server_F (65564) state IN ('AL','MS','GA')
g_shard_server_G (65565) state IN ('TX','OK','NM')
g_shard_server_H (65566) state IN ('NY','NJ')
g_shard_server_I (65567) not ((state IN ('AL','MS','GA')) or (state IN('TX','OK','NM'))
or (state IN ('NY','NJ')))
```

Example: Output for a sharding definition that uses a BSON shard key and expression-based sharding

For this example, you have a sharding definition that was created by the following command:

```
cdr define shardCollection collection_3 database_3:susan.customers_3
   -t delete -k bson_value_lvarchar(data,'age') -s expression -v version
```

```
   g_shard_server_J "BETWEEN 0 and 20"
   g_shard_server_K "BETWEEN 21 and 62"
   g_shard_server_L "BETWEEN 63 and 100"
   g_shard_server_M  REMAINDER
```

The following example shows output when the **onstat -g shard** command is run on **shard_server_J**, **shard_server_K**, **shard_server_L**, or **shard_server_M**.

Figure 4. **onstat -g shard** command output for a sharding definition that uses a BSON shard key and an expression to distribute data across multiple database servers.

```
collection_3 database_3:susan.customers_3 key:bson_value_lvarchar(data,'age')
EXPRESSION:DELETE SHARD OPTIMIZATION:ENABLED
Matching for delete:version
g_shard_server_J (65568) bson_value_lvarchar(data,'age') BETWEEN 0 and 20"
g_shard_server_K (65569) bson_value_lvarchar(data,'age') BETWEEN 21 and 62"
g_shard_server_L (65570) bson_value_lvarchar(data,'age')BETWEEN 63 and 100"
g_shard_server_M (65571) not((bson_value_lvarchar(data,'age') BETWEEN 0 and 20)
or (bson_value_lvarchar(data,'age') BETWEEN 21 and 62) or (bson_value_lvarchar
(data,'age') BETWEEN 63 and 100))
```

Related information:
cdr define shardCollection
cdr change shardCollection
cdr delete shardCollection
cdr list shardCollection
CDR_AUTO_DISCOVER configuration parameter

onstat -g sle command: Print all sleeping threads

Use the **onstat -g sle** command to print all sleeping threads.

Syntax:

```
>>-onstat-- -g--sle-------------------------------------------><
```

Example output

Figure 1. **onstat -g sle** command output

```
Current Admin VP sleep period:  10  millisecs
Sleeping threads with timeouts: 21  threads
   tid v_proc            rstcb              name      time
    49      1         b3b13a8       onmode_mon      0.02
     5      1               0  Cosvr Avail Mgr      0.05
    42      1         b3ad028      main_loop()      0.08
     9      3         b3ad6e8         xtm_svcc      0.64
    14      5               0       mgmt_thd_5      0.65
    13      4               0       mgmt_thd_4      0.65
     4      1               0       mgmt_thd_1      0.65
     6      3               0          dfm_svc      0.98
    33     13               0      mgmt_thd_13      1.54
    27     10               0      mgmt_thd_10      1.54
    21      7               0       mgmt_thd_7      1.54
    12      3               0       mgmt_thd_3      1.76
    29     11               0      mgmt_thd_11      1.76
    23      8               0       mgmt_thd_8      2.08
    31     12               0      mgmt_thd_12      2.08
    35     14               0      mgmt_thd_14      2.98
    19      6               0       mgmt_thd_6      3.00
    25      9               0       mgmt_thd_9      3.00
    37      3               0          sch_rgm      3.48
    44      5         b3af8a8      btscanner 0      7.31
    46      3         b3b0628        bum_sched     41.26
```


onstat -g smb command: Print sbspaces information

Use the **onstat -g smb** command to display detailed information about sbspaces.

Syntax:

```
>>-onstat-- -g--smb--+------------+---------------------------><
                    +-c----------+
                    +-fdd--------+
                    +-lod--------+
                    +-s----------+
                    +-e----------+
                    '-h--+-----+-'
                         +-cad-+
```

```
                +-fdd-+
                '-lod-'
```

Command	Explanation
onstat -g smb c	Lists all the chunks in the sbspace.
onstat -g smb e	Lists the entries of all smart-large-object table types.
onstat -g smb e cad	Lists the entries for the smart-large-object chunk adjunct table.
onstat -g smb e fdd	Lists the entries for the smart-large-object file descriptor table.
onstat -g smb e lod	Lists the entries in the smart-large-object header table.
onstat -g smb fdd	Lists the smart-large-object file descriptors.
onstat -g smb h	Lists the headers of all smart-large-object table types.
onstat -g smb h cad	Lists the header for the smart-large-object chunk adjunct table.
onstat -g smb h fdd	Lists the header for the smart-large-object file descriptor table.
onstat -g smb h lod	Lists the header for the smart-large-object header table.
onstat -g smb lod	Lists the header and entries in the smart-large-object header table.
onstat -g smb s	Lists the sbspace attributes (owner, name, page size, **-Df** flag settings). Fields with a value of 0 or -1 were not initialized during sbspace creation.

Example output for the onstat -g smb c command

Use the **onstat -g smb c** command to monitor the amount of free space in each sbspace chunk, and the size in pages of the user data and metadata. The **onstat -g smb c** command displays the following information for each sbspace chunk:

- Chunk number and sbspace name
- Chunk size and pathname
- Total user data pages and free user data pages
- Location and number of pages in each user-data and metadata areas

In the following example, chunk 2 of sbspace1 has 2253 original free pages (`orig fr`), 2253 user pages (`usr pgs`), and 2245 free pages (`free pg`). For the first user-data area (`Ud1`), the starting page offset is 53 and the number of pages is 1126. For the metadata area (`Md`), the starting page offset is 1179 and the number of pages is 194. For the second user data area (`Ud2`), the starting page offset is 1373 and the number of pages is 1127.

```
Chunk Summary:

sbnum 2  chunk 2
chunk:   address   flags    offset   size   orig fr   usr pgs   free pg
         303cf2a8  F------- 0        2500   2253      2253      2245
         path: /usr11/myname/sbspace1

         start pg   npages
Ud1   :  53         1126
Md    :  1179       194
Ud2   :  1373       1127
```

Output for the onstat -g smb s command

The **onstat -g smb s** command displays the storage attributes for all sbspaces in the system:

- sbspace name, flags, owner
- logging status
- average smart-large-object size
- first extent size, next extent size, and minimum extent size
- maximum I/O access time
- lock mode

For more information on the **onstat -g smb** command, see the *IBM® Informix® Performance Guide*.

onstat -g smx command: Print multiplexer group information

Use the **onstat -g smx** command to display information about the server multiplexer group for servers using SMX.

Syntax:

```
>>-onstat-- -g--smx--+-----+-------------------------------------><
                    '-ses-'
```

Command	Explanation

Command	Explanation
onstat -g smx	Displays SMX connection statistics
onstat -g smx ses	Displays SMX session statistics

Example output

Figure 1. **onstat -g smx** command output

```
SMX connection statistics:
SMX control block: 0x47d5e028

  Peer server name: lx1
  SMX connection address: 0x47d60d10
  Encryption status: Disabled  Total bytes sent: 27055
  Total bytes received: 2006989
  Total buffers sent: 782
  Total buffers received: 7090
  Total write calls: 782
  Total read calls: 7090
  Total retries for write call: 0
  Data compression level: 1
  Data sent: compressed 40760 bytes by 33%
  Data received: compressed 12579324 bytes by 84%
```

Output description

SMX control block
: SMX control block

Peer server name
: Displays the name of the peer server

SMX connection address
: Displays the address of the SMX connection

Encryption status
: Displays whether encryption is enabled or disabled

Total bytes sent
: Displays the total number of bytes sent

Total bytes received
: Displays the total number of bytes received

Total buffers sent
: Displays the total number of buffers sent

Total buffers received
: Displays the total number of buffers received

Total write calls
: Displays the total number of write calls

Total read calls
: Displays the total number of read calls

Total retries for write call
: Displays the total number of retries for write call

Data compression level
: Displays the SMX compression level as set by the SMX_COMPRESS configuration parameter

Data sent: compressed x bytes by y%
: Displays the uncompressed number of bytes and the compression ratio of the data sent

Data received: compressed x bytes by y%
: Displays the uncompressed number of bytes and the compression ratio of the data received

Example Output

Figure 2. onstat -g smx ses Output

```
SMX session statistics:
SMX control block: 0x17c69028

Peer         SMX session        client        reads         writes
name         address            type
delhi_sec      19022050    smx Clone Send            6           183
```

Output Description

SMX control block
: SMX control block

Peer name
: Displays the name of the peer server

SMX session address
: SMX session address

Client type
: Displays type of secondary server

reads
: Displays the total number of session reads

writes
Displays the total number of session writes

onstat -g spi command: Print spin locks with long spins

Use the **onstat -g spi** command to display information about spin locks with long spins.

Syntax:

```
>>-onstat-- -g--spi-----------------------------------------><
```

Many resources in the server are accessed by two or more threads. In some of these accesses (such as updating a shared value), the server must guarantee that only one thread is accessing the resource at a time. A *spin lock* is the mechanism used to provide this mutually exclusive access for some resources. With this type of lock, a thread that did not succeed in acquiring the lock on the first try (because another thread was holding it) repeatedly attempts to acquire the lock until it succeeds.

The overhead cost of a spin lock is small, and spin locks are normally used for resources that require mutual exclusion for short periods of time. However, if a spin lock becomes highly contended, the loop-and-retry mechanism can become expensive.

The **onstat -g spi** command is helpful for identifying performance bottlenecks that are caused by highly contended spin locks. This option lists spin locks with waits, those spin locks for which a thread was not successful in acquiring the lock on its first attempt and thus had to loop and re-attempt.

Example output

Figure 1. **onstat -g spi** command output

```
Spin locks with waits:

Num Waits   Num Loops    Avg Loop/Wait     Name
114         117675        1032.24          lockfr3
 87         256461        2947.83          fast mutex, lockhash[832]
  1             11          11.00          fast mutex, 1:bhash[16668]
  4          51831       12957.75          fast mutex, 1:lru-4
  1            490         490.00          fast mutex, 1:bf[994850] 0xe00002 0x14eb32000
```

Output description

Num Waits (decimal)
Total number of times a thread waited for this spin lock.
Num Loops (decimal)
Total number of attempts before a thread successfully acquired the spin lock.
Avg Loop/Wait (floating point)
Average number of attempts needed to acquire the spin lock. Computed as `Num Loops / Num Waits`.
Name (string)
Uses the following codes to name the spin lock

lockfr
The lock free list. The number after **lockfr** is the index into the lock free list array.

lockhash[]
The lock hash bucket. The field inside the brackets is the index into the lock hash bucket array.

:bhash []
The buffer hash bucket. The field before the colon is the buffer pool index; the field inside the brackets after **bhash** is the index into the buffer hash bucket array.

:lru-
The LRU latch. The field before the colon is the buffer pool index; the field after **lru-** identifies the buffer chain pairs that are being used.

:bf[]
The buffer latch. The field before the colon is the buffer pool index; the field inside the brackets after **bf** is the position of buffer in the buffer array. The next two fields are the partition number and the page header address in memory for the buffer in hex form.

onstat -g sql command: Print SQL-related session information

Use the **onstat -g sql** command to display SQL-related information about a session.

By default, only the DBSA can view **onstat -g sql** syssqltrace information. However, when the UNSECURE_ONSTAT configuration parameter is set to `1`, all users can view this information.

Syntax:

```
>>-onstat-- -g--sql--sessionid---------------------------------><
```

You can specify one of the following invocations.

Invocation
: **Explanation**

onstat -g sql
: Displays a one line summary for each session

onstat -g sql *sessionid*
: Displays SQL information for a specific session

Note: Encrypted passwords and password hint parameters in encryption functions are not shown. The following figure displays an encrypted password in the `Last parsed SQL statement` field.

Figure 1. **onstat -g sql** *command output for a completed SQL statement*

```
onstat -g sql 22

Sess  SQL             Current            Iso Lock        SQL  ISAM F.E.           Current
Id    Stmt type       Database           Lvl Mode        ERR  ERR  Vers Explain    Role
22     -              test               CR  Not Wait     0    0   9.03 Off          hr
Last parsed SQL statement :
  select id, name, decrypt_char(ssn, 'XXXXXXXXXX') from emp
```

Output description

Sess id
: The session identifier

SQL Stmt type
: The type of SQL statement

Current® Database
: Name of the current database of the session

ISO Lvl
: Isolation level

  DR
  : Dirty Read

  CR
  : Committed Read

  CS
  : Cursor Stability

  DRU
  : Dirty Read, Retain Update Locks

  CRU
  : Committed Read, Retain Update Locks

  CSU
  : Cursor Stability, Retain Update Locks

  LC
  : Committed Read, Last Committed

  LCU
  : Committed Read Last Committed with Retain Update Locks

  RR
  : Repeatable Read

  NL
  : Database Without Transactions

Lock mode
: Lock mode of the current session

SQL Error
: SQL error number encountered by the current statement

ISAM Error
: ISAM error number encountered by the current statement

F.E. Version
: The version of the SQLI protocol used by the client program

Explain
: SET EXPLAIN setting

Current Role
: Role of the current user

Figure 2. **onstat -g sql** *command output for a running SQL statement*

```
onstat -g sql 28

Sess  SQL         Current     Iso Lock     SQL  ISAM F.E.
Id    Stmt type   Database    Lvl Mode     ERR  ERR  Vers  Explain
28    SELECT      sysmaster   CR  Not Wait 0    0    9.24  Off

Current statement name : unlcur

Current SQL statement (8) :
  select * from systables, syscolumns, sysindexes

  QUERY_TIMEOUT setting:     0 (No Timeout)
  Clock time elapsed   : 00:00:12
```

```
Last parsed SQL statement :
  select * from systables, syscolumns, sysindexes
```

The QUERY_TIMEOUT setting and clock time are displayed only for running queries, not for DML or DDL statements or administration operations.

onstat -g spf: Print prepared statement profiles

Use the **onstat -g spf** command to display current statistics about SQL queries.

You can use the statistics to determine the cost of each statement.

Syntax:

```
>>-onstat-- -g--spf---------------------------------------------><
```

If SQL tracing is enabled, the information that is shown is a snapshot of the work that is completed by the statement and might change as the statement continues to run. For example, to monitor the growth rate of buffer reads or writes in an active statement, you can issue three **onstat -g spf** runs at 2-second intervals.

If SQL tracing is disabled, a warning message is issued: "Statistics disabled".

Example output

Figure 1. **onstat -g spf** command output

```
Statement profiles
sid   sdb      tottm  execs runtm  pdq scans sorts bfrd  pgrd  bfwrt  pgwrt  lkrqs  lkwts
35    4de84028 0.01   0     0.01   0   0     0     301   352   0      512    2998   0
25    4dc0b028 0.00   0     0.00   0   0     0     0     0     0      0      0      0
...
```

Output description

sid
: The session ID.

sdb
: The last 8 digits of the statement pointer.

tottm
: The current total run time, in seconds, of all statements.

execs
: The current number of completed statement runs. This value does not include statements that are running.

runtm
: The current run time of the statement, in seconds.

pdq
: The current parallel database queries (PDQ) priority level. The PDQ priority value can be any integer from 0 through 100. For more information, see Managing PDQ queries.

scans
: The current number of PDQ scans that are allocated.

sorts
: The current number of completed sorts.

bfrd
: The current number of buffer reads.

pgrd
: The current number of page reads.

bfwrt
: The current number of buffer writes.

pgwrt
: The current number of page writes.

lkrqs
: The current number of lock requests.

lkwts
: The current number of lock waits.

Related reference:
set sql tracing argument: Set global SQL tracing (SQL administration API)

onstat -g src command: Patterns in shared memory

Use the **onstat -g src** command to search for patterns in shared memory.

Syntax:

```
>>-onstat-- -g--src--pattern--mask-----------------------------><
```

Example output

The following example shows output for the **onstat -g src*pattern mask*** command where *pattern* = `0x123` and *mask* = `0xffff`.
Figure 1. **onstat -g src** command output

```
Search Summary:
addr               contents
000000000ad17a50: 01090000 00000000 00000000 00000123   ........ .......#
000000000ad7dec0: 00000001 014e3a0c 00000000 0ade0123   .....N:. .......#
```

Output description

addr (hexadecimal)
: Address in shared memory where search pattern is found

contents (hexadecimal)
: Contents of memory at given address

---

onstat -g ssc command: Print SQL statement occurrences

Use the **onstat -g ssc** command to monitor the number of times that the database server reads the SQL statement in the cache.

By default, only the DBSA can view **onstat -g ssc** syssqltrace information. However, when the UNSECURE_ONSTAT configuration parameter is set to `1`, all users can view this information.

Syntax:

```
>>-onstat-- -g--ssc--+----------+-------------------------------><
                     '-+-all--+-'
                       '-pool-'
```

The **all** option reports the *key-only* cache entries as well as the fully cached statements. If the value in the **hits** column is less than the STMT_CACHE_HITS value, that entry is a *key-only* cache entry. For more information, see memory utilization in the *IBM® Informix® Performance Guide*.

The **pool** option reports usage of all memory pools for the SQL statement cache. The output displays information on the name, class, address, and total size of the memory pools. For more information, see improving query performance in the *IBM Informix Performance Guide*.

Example output

Figure 1. **onstat -g ssc** command output

```
Statement Cache Summary:
#lrus   currsize  maxsize   Poolsize  #hits   nolimit
4       117640    524288    139264    0       1
Statement Cache Entries:
lru hash ref_cnt hits flag heap_ptr     database           user
---------------------------------------------------------------
  0  262       0    7   -F  aad8038      sscsi007           admin
   INSERT INTO ssc1 ( t1_char , t1_short , t1_key , t1_float , t1_smallfloat
    , t1_decimal , t1_serial ) VALUES ( ? , ? , ? , ? , ? , ? , ? )
  0  127       0    9   -F  b321438      sscsi007           admin
   INSERT INTO ssc2 ( t2_char , t2_key , t2_short ) VALUES ( ? , ? , ? )
  1  134       0   15   -F  aae0c38      sscsi007           admin
   SELECT t1_char , t1_short , t1_key , t1_float , t1_smallfloat ,
    t1_decimal , t1_serial FROM ssc1 WHERE t1_key = ?
  1  143       0    3   -F  b322c38      sscsi007           admin
   INSERT INTO ssc1 ( t1_char , t1_key , t1_short ) SELECT t2_char , t2_key
    + ? , t2_short FROM ssc2
  2   93       0    7   -F  aae9838      sscsi007           admin
   DELETE FROM ssc1 WHERE t1_key = ?
  2  276       0    7   -F  aaefc38      sscsi007           admin
   SELECT count ( * ) FROM ssc1
  2  240       1    7   -F  b332838      sscsi007           admin
   SELECT COUNT ( * ) FROM ssc1 WHERE t1_char = ? AND t1_key = ? AND
    t1_short = ?
  3   31       0    7   -F  aaec038      sscsi007           admin
   SELECT count ( * ) FROM ssc1 WHERE t1_key = ?
  3   45       0    1   -F  b31e438      sscsi007           admin
   DELETE FROM ssc1
  3  116       0    0   -F  b362038      sscsi007           admin
   SELECT COUNT ( * ) FROM ssc1
    Total number of entries: 10.
```

Output description - Statement Cache Summary section

#lrus
Number of least recently used queues (LRUS)
currsize
Current® cache size
maxsize
Limit on total cache memory
Poolsize
Total pool size
#hits
The number of hits before insertion. This number equals the value of the STMT_CACHE_HITS configuration parameter
nolimit
The value of the STMT_CACHE_NOLIMIT configuration parameter

Output description - Statement Cache Entries section

The Statement Cache Entries section shows the entries that are fully inserted into the cache.

lru
The index of lru queue to which the cache entry belongs
hash
Hash values of cached entry
ref_count
Number of threads referencing the statement
hits
Number of times a statement matches a statement in the cache. The match can be for a key-only or fully cached entry.
flag
Cache entry flag `-D` indicates that the statement is dropped, `-F` indicates that the statement is fully cached, and `-I` indicates that the statement is in the process of being moved to a fully cached state
heap_ptr
Address of memory heap for cache entry

Related reference:
STMT_CACHE_HITS configuration parameter
STMT_CACHE_NOLIMIT configuration parameter
STMT_CACHE_NUMPOOL configuration parameter

onstat -g stk command: Print thread stack

Use the **onstat -g stk** *tid* command to display the stack of the thread specified by thread ID.

This option is not supported on all platforms and is not always accurate.

Syntax:

```
>>-onstat-- -g--stk -tid----------------------------------------><
```

Example output

Figure 1. **onstat -g stk** *tid* command output

```
Stack for thread: 2 adminthd
 base: 0x000000010aad5028
  len:   33280
   pc: 0x00000001002821e8
  tos: 0x000000010aadc621
state: running
   vp: 2

0x1002821e8 oninit :: yield_processor + 0x260 sp=0x10aadce20(0x10ac834d0, 0x0, 0x1,
   0x100000000, 0xc8a000, 0x100c8a000)
0x100274e38 oninit :: wake_periodic + 0xdc sp=0x10aadced0 delta_sp=176(0x41b0, 0xc7a024bc,
   0x0, 0x41c4, 0x10aacf598, 0x90)
0x100274fcc oninit :: admin_thread + 0x108 sp=0x10aadcf80 delta_sp=176(0x0, 0x2328,
   0xd26c00, 0x5, 0xc8a000, 0x156c)
0x1002484ec oninit :: startup + 0xd8 sp=0x10aadd050 delta_sp=208(0xa, 0x10aad47d0,
   0x10aad47d0, 0x100db1988, 0xd1dc00, 0x1)
```


onstat -g stm command: Print SQL statement memory usage

Use the **onstat -g stm** command to display the memory that each prepared SQL statement uses.

By default, only the DBSA can view **onstat -g stm** syssqltrace information. However, when the UNSECURE_ONSTAT configuration parameter is set to 1, all users can view this information.

Syntax:

```
>>-onstat-- -g--stm-------------------------------------------><
```

To display the memory for only one session, specify the session ID in the **onstat -g stm** command.

Example output

Figure 1. **onstat -g stm** command output

```
session   65 ------------------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 aad8028   16544   SELECT COUNT ( * ) FROM ssc1 WHERE t1_char = ?
 AND t1_key = ? AND t1_short = ?
```

Output description

sdblock
: Address of the statement descriptor block

heapsz
: Size of the statement memory heap

statement
: Query text

onstat -g stq command: Print queue information

Use the **onstat -g stq** command to display information about the queue.

Syntax:

```
>>-onstat-- -g--stq--session-----------------------------------><
```

To view queue information for a particular session specify the *session* option. Omit the *session* option to view queue information for all sessions.

Example output

Figure 1. **onstat -g stq** command output

```
Stream Queue: (session 25 cnt 4)  0:db12400  1:db18400  2:dcf0400  3:dcf6400
 Full Queue:  (cnt 2  waiters 0)  0:0  1:db12400
 Empty Queue: (cnt 0  waiters 0)
```

Output description

session
: Session id

cnt
: Number of stream queue buffers

waiters
: Number of threads waiting for the stream queue buffer

onstat -g sts command: Print stack usage for each thread

Use the **onstat -g sts** command to display information about the maximum and current stack use for each thread.

Syntax:

```
>>-onstat-- -g--sts-------------------------------------------><
```

Example output

Figure 1. **onstat -g sts** command output

```
Stack usage:

TID     Total   Max             Current         Thread Name
                bytes   %       bytes   %
2       32768   3124    9       3079    9       adminthd
3       32768   2870    8       2871    8       childthd
5       32768   14871   45      2871    8       Cosvr Avail Mgr
6       32768   2870    8       2871    8       dfm_svc
7       131072  3190    2       3191    2       xmf_svc
9       32768   3126    9       3127    9       xtm_svcc
10      32768   3580    10      3335    10      xtm_svcp
11      32768   3238    9       3239    9       cfgmgr_svc
12      32768   6484    19      2871    8       lio vp 0
14      32768   6484    19      2871    8       pio vp 0
16      32768   6484    19      2871    8       aio vp 0
18      131072  10391   7       2871    2       msc vp 0
20      32768   4964    15      2871    8       fifo vp 0
22      32768   4964    15      2871    8       fifo vp 1
24      32768   6028    18      2871    8       aio vp 1
26      32768   5444    16      2951    9       dfmxpl_svc
27      32768   2886    8       2887    8       sch_svc
28      32768   7812    23      5015    15      rqm_svc
29      32768   7140    21      3079    9       sm_poll
30      32768   11828   36      6439    19      sm_listen
31      32768   2870    8       2871    8       sm_discon
32      32768   14487   44      4055    12      main_loop()
33      32768   4272    13      2903    8       flush_sub(0)
34      32768   2902    8       2903    8       flush_sub(1)
35      32768   2870    8       2871    8       btscanner 0
36      32768   3238    9       3239    9       aslogflush
37      32768   3055    9       2887    8       bum_local
38      32768   3238    9       3239    9       bum_rcv
39      32768   4902    14      4903    14      onmode_mon
42      32768   4964    15      2871    8       lio vp 1
44      32768   5136    15      2871    8       pio vp 1
```


onstat -g sym command: Print symbol table information for the oninit utility

Use the **onstat -g sym** command to display symbol table information for the **oninit** utility.

Syntax:

```
>>-onstat-- -g--sym-----------------------------------------><
```

Example output

Figure 1. **onstat -g sym** command output
The following example shows the first few lines from the output:

```
Table for oninit has 23378 entries
    Initial value for -base-: 0x0
        0x3451e0 _start
        0x345300 .ld_int
        0x345348 .ld_llong
        0x3453dc .ld_float
        0x345428 .ld_double
        0x3454c4 .st_int
        0x3454fc .st_llong
        0x34556c .st_float
        0x3455c0 .st_double
        0x34565c .st_float_foreff
        0x345694 .st_double_foreff
        0x345718 main
        0x34c2ac get_cfgfile
        0x34c2fc is_server_alias
```

Output description

The **onstat -g sym** command displays the relative in-memory address and name of symbols (functions and variables) in the **oninit** utility.

onstat -g top command: Print top consumers of resources

Use the **onstat -g top** command to display information about top consumers of various resources such as CPU time, I/O operations, and memory growth.

You can specify the maximum number of consumers to display, along with the sample time interval. The onstat -g top command may be followed by three optional numeric arguments:

```
Syntax:

>>onstat -g top [<keyword pair>] [<max consumers> [<interval> [<repeats>]]]
```

Onstat -g top can take an optional pair of keyword arguments:

- **Top Threads**
  - onstat -g top thread cpu: Threads currently using the most CPU time.
  - onstat -g top thread drd: Threads currently performing the most disk reads.
  - onstat -g top thread bfr: Threads currently performing the most buffer reads.
  - onstat -g top thread bfw: Threads currently performing the most buffer writes.
  - onstat -g top thread plg: Threads currently performing the most physical logging.
  - onstat -g top thread llg: Threads currently performing the most logical logging.
- **Top Sessions**
  - onstat -g top session cpu: Sessions currently using the most CPU time.
  - onstat -g top session drd: Sessions currently performing the most disk reads.
  - onstat -g top session bfr: Sessions currently performing the most buffer reads.
  - onstat -g top session bfw: : Sessions currently performing the most buffer writes.
  - onstat -g top session plg: Sessions currently performing the most physical logging.
  - onstat -g top session llg: Sessions currently performing the most logical logging.
- **Top Chunks**
  - onstat -g top chunk ios: Chunks currently performing the most reads and writes.
  - onstat -g top chunk art: Chunks with the highest average read time.
  - onstat -g top chunk awt: Chunks with the highest average write time.
- **Top Spaces**
  - onstat -g top space ios: Spaces currently performing the most reads and writes.
  - onstat -g top space art: Spaces with the highest average read time.
  - onstat -g top space awt: Spaces with the highest average write time.
- **Top Virtual Memory Pools**
  - onstat -g top mempool gro: Memory pools currently growing the fastest.
- **Top Session Memory**
  - onstat -g top sessmem gro: Sessions currently allocating the most memory.
- **Top Partitions**
  - onstat -g top partition drd: Partitions currently performing the most disk reads.
- **Top Tables**
  - onstat -g top table drd: Tables currently performing the most disk reads.

Example

Figure 1. **onstat -g top** command examples

```
Top 3 spaces for average write times
onstat -g top space awt 3
top 10 tables for disk reads over a 1 minute interval
onstat -g top partition drd 10 60
Top 15 threads for CPU time, sampling every 20 seconds, and repeating 5 times
 onstat -g top thread cpu 15 20 5
Memory pools that are growing, updating every 30 seconds, repeating indefinitely
onstat -g top mempool gro 0 30 0
The least active session in terms of buffer reads, repeating 10 times, with 5 seconds between each update
onstat -g top session BFR 1 5 10
```

Note: To reverse the order of the consumer list, capitalize the second keyword. For example, the following command will display the top *least* active chunks:

```
onstat -g top chunk IOS
```


onstat -g tpf command: Print thread profiles

Use the **onstat -g tpf** command to display thread profiles.

```
Syntax:

>>-onstat-- -g--tpf--tid----------------------------------------><
```

Specify the *tid* thread ID to print the profile for a specific thread. Set *tid* to 0 to display the profiles for all of the threads.

Example output

Figure 1. **onstat -g tpf** command output

```
onstat -g tpf 945

Thread profiles
tid lkreqs lkw dl to lgrs isrd iswr isrw isdl isct isrb lx bfr bfw lsus lsmx seq
945 1969   0   0  0  6181 1782 2069 13   0    0    0    0  16183 7348 743580 0    6
```

Output description

tid
: Thread ID

lkreqs
: Lock requests

lkw
: Lock waits

dl
: Deadlocks

to
: Remote deadlock timeout

lgrs
: Log records

isrd
: Number of reads

iswr
: Number of writes

isrw
: Number of rewrites

isdl
: Number of deletes

isct
: Number of commits

isrb
: Number of rollbacks

lx
: Long transactions

bfr
: Buffer reads

bfw
: Buffer writes

lsus
: Log space currently used

lsmx
: Max log space used

seq
: Sequence scans

onstat -g ufr command: Print memory pool fragments

Use the **onstat -g ufr** command to display a list of the fragments that are currently in use in the specified memory pool.

This command requires an additional argument to specify either a pool name or session ID whose memory pool information is to be displayed. Each session is allocated a memory pool with the same name as the session ID. Use the **onstat -g mem** command to identify the pool name and the **onstat -g ses** command to identify the session ID.

Syntax:

```
>>-onstat-- -g--ufr--+-pool name-+-----------------------------><
                     '-sessionid-'
```

Memory pools are broken into fragments for various uses. With the **onstat -g ufr** command it is possible to see a list of these fragments showing their respective sizes in bytes and the type of information they contain. The information provided is generally used by Technical Support to assist in the analysis of a reported problem.

Example output for a specified pool name

Figure 1. **onstat -g ufr global** command output for a specified pool name

```
Memory usage for pool name global:
size       memid
1736       overhead
23544      mcbmsg
72         messages
33112      osenv
25432      rsam
88         shmblklist
5170664    net
```

Example output for a specified session ID

The following example shows the output for session ID 6.

Figure 2. **onstat -g ufr** command output for a specified session ID

```
Memory usage for pool name 6:
size       memid
3256       overhead
```

```
144         scb
2968        ostcb
18896       sqscb
3312        opentable
72          sql
808         filetable
352         fragman
552         hashfiletab
1584        gentcb
12096       log
2960        sqtcb
2928        osenv
720         keys
224         rdahead
16248       temprec
```

Output description

size (decimal)
: Size, in bytes, of the pool fragment.

memid (string)
: Name of the pool fragment.

onstat -g vpcache command: Print CPU virtual processor and tenant virtual processor private memory cache statistics

Run the **onstat -g vpcache** command to display statistics about CPU virtual processor and tenant virtual processor private memory caches.

Syntax:

```
>>-onstat-- -g--vpcache-----------------------------------------><
```

Example output

The output for each CPU or tenant virtual processor has the same format. The following example shows the output for one CPU virtual processor.

Figure 1. **onstat -g vpcache** command output

```
CPU virtual processor memory block cache statistics - 4096 byte blocks

Number of 4096 byte memory blocks requested for each CPU virtual processor:262144
CPU virtual processor memory block cache mode : Dynamic

vpid    pid          Blocks held  Hit percentage   Free cache
1       2557540      4667202       99.2 %           100.0 %

Current total virtual processor allocations from cache: 59466799, Total frees: 60209953

  size  cur blks   tgt blks   alloc      miss     free      drain  draintime
  1     1662023    9661       49167485   0        49816526  0      Thu Apr 11 09:43:35 2013
  2     130        52428      7609556    297043   7609612   0      Thu Jan  1 00:00:00 1970
  3     329160     9          905094     0        943256    0      Thu Apr 11 09:43:36 2013
  4     424        9          306637     16192    306506    0      Thu Apr 11 09:43:33 2013
  5     10         9          119313     122607   119315    0      Thu Apr 11 09:43:36 2013
  6     20790      9          55305      0        57700     0      Thu Apr 11 09:43:23 2013
  7     9877       9          31164      0        31942     0      Thu Apr 11 09:43:14 2013
  8     2816       5242       6500       0        6537      0      Thu Jan  1 00:00:00 1970
  9     234        9          606575     8323     605525    0      Thu Apr 11 09:43:36 2013
  10    1130       9          5597       0        5679      0      Thu Apr 11 09:43:18 2013
  11    231        5242       1808       0        1753      0      Thu Jan  1 00:00:00 1970
  12    1068       9          5667       0        5666      0      Thu Apr 11 09:43:28 2013
  13    65         5242       7114       175      7110      0      Thu Jan  1 00:00:00 1970
  14    28         5242       26200      172      26185     0      Thu Jan  1 00:00:00 1970
  15    30         5242       13562      553      13547     0      Thu Jan  1 00:00:00 1970
  16    2627136    34         349124     0        408425    0      Thu Apr 11 09:43:35 2013
  17    1309       9          59         0        107       0      Thu Apr 11 09:27:33 2013
  18    198        5242       7          0        6         0      Thu Jan  1 00:00:00 1970
  19    190        5242       5          0        1         0      Thu Jan  1 00:00:00 1970
  20    60         5242       30         19       19        0      Thu Jan  1 00:00:00 1970
  21    462        5242       38         0        43        0      Thu Jan  1 00:00:00 1970
  22    22         5242       3          0        1         0      Thu Jan  1 00:00:00 1970
  23    69         5242       141        15       135       0      Thu Jan  1 00:00:00 1970
  24    4944       35         189509     2078     185347    0      Thu Apr 11 09:43:35 2013
  25    75         5242       1          0        1         0      Thu Jan  1 00:00:00 1970
  26    0          9          364        220      361       0      Thu Apr 11 09:39:17 2013
  27    27         5242       1          0        2         0      Thu Jan  1 00:00:00 1970
  28    56         5242       415        33       410       0      Thu Jan  1 00:00:00 1970
  29    319        5242       7101       735      7088      0      Thu Jan  1 00:00:00 1970
  30    3240       5242       174        0        223       0      Thu Jan  1 00:00:00 1970
  31    279        11         51994      2515     50682     0      Thu Apr 11 09:43:36 2013
  32    800        5242       256        0        243       0      Thu Jan  1 00:00:00 1970
```

Output description

vpid
: The ID of the virtual processor

pid
: The process ID for the virtual processor that is assigned by the operating system

Blocks held
: The number of 4096 byte blocks that are available in the private memory cache

Hit percentage
: The percentage of time that a block was available when requested

Free cache
: The percentage of time that blocks were freed for reuse without being drained

Current VP total allocations from cache
: The number of times a block or group of blocks was taken from the cache

Total frees
: The number of times a block or group of blocks was added to the cache

size
: The size of the memory blocks, in 4096-byte blocks

cur blks
: The current number of 4096-byte blocks that are allocated (a multiple of `size`)

tgt blks
: The target number of blocks for the cache entry before the cache is drained

alloc
: The number of times a requestor received a block of this size

miss
: The number of times a block was requested but none were available

free
: The number of times a memory block was placed into the cache

drain
: The number of times an aged block was forced out to make room for another block

draintime
: The last time the bin of memory blocks was drained

Related reference:
VP_MEMORY_CACHE_KB configuration parameter
Related information:
Private memory caches

onstat -g wai command: Print wait queue thread list

Use the **onstat -g wai** command to display a list of the threads in the system that are currently in the wait queue and not currently executing. The output is sorted by thread ID.

Syntax:

```
>>-onstat-- -g--wai-----------------------------------------><
```

Example output

Figure 1. **onstat -g wai** command output

```
Waiting threads:
 tid      tcb              rstcb            prty status                vp-      name
 2        46b1ea40         0                1    IO Idle               5lio        lio vp 0
 3        46b3dc58         0                1    IO Idle               6pio        pio vp 0
 4        46b5dc58         0                1    IO Idle               7aio        aio vp 0
 5        46b7cc58         0                1    IO Idle               8msc        msc vp 0
 6        46b1ed10         460f5028         1    sleeping secs: 1      3cpu        main_loop()
 9        46d0d6e0         0                1    sleeping forever      1cpu        soctcplst
 10       46d70b48         0                1    sleeping forever      3cpu        sm_listen
 11       46e5d9a0         0                1    sleeping secs: 1      3cpu        sm_discon
 12       46e5dc70         460f5820         1    sleeping secs: 1      3cpu        flush_sub(0)
 13       46e8a5a8         460f6018         1    sleeping secs: 1      3cpu        aslogflush
 14       46fe8148         460f6810         1    sleeping secs: 41     3cpu        btscanner_0
 15       46fe84a8         0                1    IO Idle              10aio        aio vp 1
 16       46fe8778         460f7008         1    sleeping secs: 1      1cpu        onmode_mon
 36       47531960         460f7ff8         1    sleeping secs: 253    3cpu        dbScheduler
 37       47531c30         460f87f0         1    sleeping forever      4cpu        dbWorker1
 38       47491028         460f7800         1    sleeping forever      4cpu        dbWorker2
```

Output description

tid (decimal)
: Thread ID

tcb (hex)
: In-memory address of the thread control block

rstcb (hex)
: In-memory address of the RSAM thread control block

prty (decimal)
: Thread priority. Higher numbers represent higher priorities

status (string)
: Current® status of the thread

vp- (decimal and string)
: Virtual processor integer ID of the VP on which the thread last ran, concatenated with the name of the VP upon which the thread runs

name (string)
: Name of the thread

onstat -g wmx command: Print all mutexes with waiters

Use the **onstat -g wmx** command to display all of the mutexes with waiters.

Syntax:

```
>>-onstat-- -g--wmx-----------------------------------------><
```

Example output

Figure 1. **onstat -g wmx** command output

```
Mutexes with waiters:
mid      addr              name              holder   lkcnt  waiter   waittime
134825   7000002043a9148   free_lock         11009    0      200      22921
                                                             11010    22918
```

Output description

mid
: Internal mutex identifier

addr
: Address of locked mutex

name
: Name of the mutex

holder
: Thread ID of the thread that is holding the mutex
: 0 = The read/write mutex is held in shared mode

lkcnt
: For a read/write mutex, the current number of threads that are locking the mutex in shared mode. For a relockable mutex, the number of times the mutex was locked or relocked by the thread that is holding the mutex.

waiter
: List of IDs of the threads that are waiting for this mutex

waittime
: Amount of time in seconds that the thread is waiting

onstat -g wst command: Print wait statistics for threads

Use the **onstat -g wst** command to show the wait statistics for the threads within the system.

The WSTATS configuration parameter must be set to `1` to enable wait statistics collection. For more information, see WSTATS configuration parameter .

Syntax:

```
>>-onstat-- -g--wst-----------------------------------------><
```

Example output

```
Version 11.70.F  -- On-Line -- Up 18:52:59 -- 78856 Kbytes
name    tid    state            n          avg(us)     max(us)
msc vp 0 5     ready            6                9          17
msc vp 0 5     run              6             1107        2215
msc vp 0 5     IO Idle          5           2985.9s    1496.1s

main_loo 7     IO Wait          55             6496      16725
main_loo 7     yield time       44929          1.2s      343.1s
main_loo 7     ready            44998        206085      343.1s
main_loo 7     run              44985             5         436
```

```
...

sqlexec  63   IO Wait          2            1118      2165
sqlexec  63   other cond       6           34237    204142
sqlexec  63   ready            9               7        16
sqlexec  63   run              7             1.1s      7.7s
```

Output description

name (string)
: Thread name

tid (decimal)
: Thread ID

state (string)
: State the thread waited in for this line of output. A single thread can have multiple lines of output if it waited in more than one state. Values that can appear in the `state` field include:

- `chkpt cond`: The thread waited for a checkpoint condition.
- `cp mutex`: The thread waited for checkpoint mutex to become available.
- `deadlock mutex`: The thread waited for a deadlock mutex to become available.
- `empty Q`: The thread waited for an empty buffer on a queue.
- `fork`: The thread waited for a child thread to run.
- `full Q`: The thread waited for a full buffer on a queue.
- `IO Idle`: The I/O thread was idle.
- `IO Wait`: The thread yielded while it waited for I/O completion.
- `join wait`: The thread waited for another thread to exit.
- `lock mutex`: The thread waited for lock mutex to become available.
- `lockfree mutex`: The thread waited for a lock-free mutex to become available.
- `logflush`: Logical log flushing occurred.
- `log mutex`: The thread waited for logical log mutex to become available.
- `logcopy cond`: The thread waited for logical log copy condition.
- `logio cond`: The thread waited for a logical log condition.
- `lrus mutex`: The thread waited for a buffer LRU mutex to become available.
- `misc`: The thread waited for a miscellaneous reason.
- `other cond`: The thread waited for an internal condition.
- `other mutex`: The thread waited for an internal system mutex to become available.
- `other yield`: The thread yielded for an internal reason.
- `OS read`: The thread waited for an operating system read call to complete.
- `OS write`: The thread waited for an operating system write call to complete.
- `ready`: The thread was ready to run.
- `run`: The thread ran.
- `sort io`: The thread waited for sort I/O completion.
- `vp mem sync`: The thread waited for synchronization of virtual processor memory.
- `yield bufwait`: The thread yielded while it waited for a buffer to become available.
- `yield 0`: The thread yielded with an immediate timeout.
- `yield time`: The thread yielded with a timeout.
- `yield forever`: The thread yielded and stays that way until it wakes up.

n (decimal)
: Number of times the thread waited in this state

avg(us) (floating point)
: Average user time the thread spent waiting in this state per wait occurrence. Time is in microseconds; an `s` after the value indicates user time in seconds.

max(us) (floating point)
: Maximum user time the thread spent waiting in this state for a single wait occurrence. Time is in microseconds; an `s` after the value indicates user time in seconds.

Related reference:
onstat -g ath command: Print information about all threads
Related information:
WSTATS configuration parameter

onstat -G command: Print TP/XA transaction information

Use the **onstat -G** command to display information about global transactions generated through the TP/XA library.

Syntax:

```
>>-onstat-- -G-------------------------------------------------><
```

Example output

Figure 1. **onstat -G** command output

```
Global Transaction Identifiers
address    flags  isol    timeout  fID       gtl  bql  data      dbpartnum
```

```
45cb0318   -LH-G  COMMIT  0        4478019   2    2    30323032  100163
```

For a tightly coupled transaction, all branches share the same transaction address shown in the address column.

Output description

address
: Transaction address

flags
: **Flag codes for position 1 (current transaction state):**

  A
  : User thread attached to the transaction

  S
  : TP/XA suspended transaction

  C
  : TP/XA waiting for rollback

  Flag codes for position 2 (transaction mode):

  T
  : Tightly-coupled mode (MTS)

  L
  : Loosely-coupled mode (default mode)

  Flag codes for position 3 (transaction stage):

  B
  : Begin work

  P
  : Distributed query prepared for commit

  X
  : TP/XA prepared for commit

  C
  : Committing or committed

  R
  : Rolling back or rolled back

  H
  : Heuristically rolling back or rolled back

  Flag code for position 4:

  X
  : XA data source global transaction

  Flag codes for position 5 (type of transaction):

  G
  : Global transaction

  C
  : Distributed query coordinator

  S
  : Distributed query subordinate

  B
  : Both distributed query coordinator and subordinate

  M
  : Redirected global transaction

isol
: Transaction isolation level

timeout
: Transaction lock timeout

fID
: Format ID

gtl
: Global transaction ID length

bql
: Branch qualifier length

data
: Transaction-specific data

dbpartnum
: Database identifier of where the transaction starts

Related reference:
IFX_XA_UNIQUEXID_IN_DATABASE configuration parameter

onstat -h command: Print buffer header hash chain information

Use the **onstat -h** command to display information about the buffer header hash chains (sometimes called "hash buckets") that are used to access pages in each buffer pool.

Syntax:

```
>>-onstat-- -h-------------------------------------------------><
```

Example output

The output is displayed in the form of a numeric histogram of chain lengths, with summary information for each buffer pool. All numeric values in the output are decimal. Shorter hash chains enable requested buffers to be located more quickly by the server, because on average it will need to check fewer buffer headers on a target chain to find the target buffer.

The page size of the buffer pool in bytes is shown as a header to the output for each buffer pool. The histogram and summary information are then presented for that buffer pool.

Figure 1. **onstat -h** command output

```
Buffer pool page size: 2048

buffer hash chain length histogram
   # of chains          of len
          3423               0
          4546               1
           223               2
          8192   total chains
          4992   hashed buffs
          5000    total buffs

Buffer pool page size: 4096

buffer hash chain length histogram
   # of chains          of len
           707               0
           315               1
             2               2
          1024   total chains
           319   hashed buffs
          1000    total buffs
```

Output description

Histogram Information on Hash Chains

The histogram information has a row for each buffer hash chain length that presently exists in the system. Each row has two columns:

of chains
: Number of hash chains of the given length

of len
: Length of these chains

Summary Information Per Buffer Pool

total chains
: Number of hash chains that exist for this buffer pool

hashed buffs
: Number of buffer headers currently hashed into the hash chains for this buffer pool

total buffs
: Total number of buffers in this buffer pool

onstat -i command: Initiate interactive mode

Use the **onstat -i** command to put the **onstat** utility in the interactive mode.

Syntax:

```
>>-onstat-- -i--+-------------+--------------------------------><
               +-r--seconds--+
               '-rz--seconds-'
```

In interactive mode, you can enter multiple **onstat** options per session, but only one at a time. An **onstat** prompt appears and allows you to enter an option.

Important: In interactive mode, do not precede the option with a dash.

Additional options

Two additional options, **onstat r *seconds*** and **onstat rz *seconds***, are available in interactive mode. The **onstat r *seconds*** option is similar to the current **onstat -r *seconds*** option, which repeatedly generates a display. If an administrator executes **onstat r *seconds*** at the interactive-mode prompt, the prompt changes to reflect the specified interval in seconds and reappears, waiting for the next command. In the following example, the display generated by the next command repeats every three seconds:

```
onstat> r 3
onstat[3]>
```

The **onstat rz *seconds*** option enables you to repeat the next command as specified and set all profile counters to 0 between each execution.

Terminating interactive mode or repeating sequence

To terminate the interactive mode, press `CTRL-d`.

To terminate a repeating sequence, press `CTRL-c`.

onstat -j command: Provide onpload status information

Use the **onstat -j** command to provide information about the status of an **onpload** job.

The **onstat -j** command provides an interactive mode that is analogous to the **onstat -i** command.

Syntax:

```
>>-onstat-- -j-------------------------------------------------><
```

When **onpload** starts, it writes a series of messages to **stdout** or to a log file. The following lines show a typical **onpload** log file:

```
Mon Jul 23 16:11:30 2010

SHMBASE         0x4400000
CLIENTNUM       0x49010000
Session ID 1

Load Database   -> cnv001
Load Table      -> cnv001a
Load File       -> testrec.dat
Record Mapping  -> cnv001a

Database Load Completed -- Processed 50 Records
Records Inserted-> 50
Detected Errors--> 0
Engine Rejected--> 0

Mon Jul 23 16:11:37 2010
```

Output description

The two lines that start with SHMBASE and CLIENTNUM provide the information that you need to locate shared memory for an instance of **onpload**. The **oninit** process has similar values stored in the $ONCONFIG file. When you use the **onstat** utility to gather information about the **oninit** process, the **onstat** utility uses information from $INFORMIXDIR/etc/$ONCONFIG file to locate shared memory. When you use **onstat** to gather information about **onpload**, you must give the **onstat** utility the name of a file that contains SHMBASE and CLIENTNUM information.

Typically the file that contains the SHMBASE and CLIENTNUM information is the log file. For example, if the **onpload** log file is **/tmp/cnv001a.log**, you can enter the following command:

```
onstat -j /tmp/cnv001a.log
```

The previous command causes the **onstat** utility to attach to **onpload** shared memory and to enter interactive mode. You can then enter a question mark (?) or any other pseudo request to see a usage message displayed. An example follows:

```
onstat> ?
Interactive Mode: One command per line, and - are optional.
    -rz    repeat option every n seconds (default: 5) and
           zero profile counts
  MT COMMANDS:
    all    Print all MT information
    ath    Print all threads
    wai    Print waiting threads
    act    Print active threads
    rea    Print ready threads
    sle    Print all sleeping threads
    spi    print spin locks with long spins
    sch    print VP scheduler statistics
    lmx    Print all locked mutexes
    wmx    Print all mutexes with waiters
    con    Print conditions with waiters
    stk <tid>   Dump the stack of a specified thread
    glo    Print MT global information
    mem <pool name|session id>   print pool statistics.
    seg    Print memory segment statistics.
    rbm    print block map for resident segment
    nbm    print block map for non-resident segments
```

```
    afr <pool name|session id> Print allocated poolfragments.
    ffr <pool name|session id> Print free pool fragments.
    ufr <pool name|session id> Print pool usage breakdown
    iov   Print disk IO statistics by vp
    iof   Print disk IO statistics by chunk/file
    ioq   Print disk IO statistics by queue
    iog   Print AIO global information
    iob   Print big buffer usage by IO VP
    sts   Print max and current stack sizes
    qst   print queue statistics
    wst   print thread wait statistics
    jal   Print all Pload information
    jct   Print Pload control table
    jpa   Print Pload program arguments
    jta   Print Pload thread array
    jmq   Print Pload message queues, jms for summary only
onstat>
```

Most of the options are the same as those that you use to gather information about Informix®, with the following exceptions:

```
jal   Print all Pload information
jct   Print Pload control table
jpa   Print Pload program arguments
jta   Print Pload thread array
jmq   Print Pload message queues, jms for summary only
```

These options apply only to **onpload**. You can use the **onstat -j** command to check the status of a thread, locate the VP and its PID, and then attach a debugger to a particular thread. The options for the **onstat** utility that do not apply to **onpload** are not available (for example, **onstat -g ses**).

onstat -k command: Print active lock information

Use the **onstat -k** command to print information about active locks, including the address of the lock in the lock table.

Syntax:

```
>>-onstat-- -k-------------------------------------------------><
```

Example output

The maximum number of locks available is specified by the value of the LOCKS configuration parameter in the onconfig file.

Figure 1. onstat -k command output

```
Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
a095f78  0        a4d9e68  0        HDR+S    100002   203         0
 1 active, 2000 total, 2048 hash buckets, 0 lock table overflows
```

In the following output, the number 2 in the last row shows an Enterprise Replication pseudo lock:

```
Locks
address  wtlist    owner    lklist   type    tblsnum  rowid    key#/bsiz
a1993e8  0         5c2f03d0 a19be30     S         2   1c05a            0
```

Output description

address
: Is the address of the lock in the lock table
If a user thread is waiting for this lock, the address of the lock shows in the **wait** field of the **onstat -u** (users) output.

wtlist
: Is the first entry in the list of user threads that is waiting for the lock, if there is one

owner
: Is the shared-memory address of the thread that is holding the lock
This address corresponds to the address in the **address** field of **onstat -u** (users) output. When the **owner** value is displayed in parentheses, it represents the shared memory address of a transaction structure. This scenario is possible only when a lock is allocated for a global transaction. This address corresponds to the address field of the output for **onstat -G**.

lklist
: Is the next lock in a linked list of locks that are held by the owner listed

type
: Uses the following codes to indicate the type of lock:

  HDR
  : Header

  B
  : Bytes

  S
  : Shared

  X

Exclusive

I
: Intent

U
: Update

IX
: Intent-exclusive

IS
: Intent-shared

SIX
: Shared, intent-exclusive

tblsnum
: Is the tblspace number of the locked resource. If the number is less than 10000, it indicates Enterprise Replication pseudo locks.

rowid
: Is the row identification number
The rowid provides the following lock information:

- If the rowid equals zero, the lock is a table lock.
- If the rowid ends in two zeros, the lock is a page lock.
- If the rowid is six digits or fewer and does not end in zero, the lock is probably a row lock.
- If the rowid is more than six digits, the lock is probably an index key-value lock.

key#/bsiz
: Is the index key number, or the number of bytes locked for a VARCHAR lock
If this field contains 'K-' followed by a value, it is a key lock. The value identifies which index is being locked. For example, K-1 indicates a lock on the first index that is defined for the table.

Related reference:
LOCKS configuration parameter

onstat -l command: Print physical and logical log information

Use the **onstat -l** command to display information about the physical logs, logical logs, and temporary logical logs.

Syntax:

```
>>-onstat-- -l-----------------------------------------------><
```

Example Output

Figure 1. **onstat -l** command output

```
Physical Logging
Buffer bufused  bufsize  numpages numwrits pages/io
  P-1  0        16       716      55       13.02
      phybegin          physize  phypos   phyused  %used
      1:263             500      270      0        0.00

Logical Logging
Buffer bufused  bufsize  numrecs  numpages numwrits recs/pages pages/io
  L-3  0        16       42169    2872     1043     14.7       2.8
       Subsystem    numrecs  Log Space used
       OLDRSAM      42169    4436496

address  number   flags    uniqid   begin                size     used    %used
a517f70  1        U-B----  1        1:763                500      500   100.00
a517fb0  2        U-B----  2        1:1263               500      500   100.00
a40daf0  3        U-B----  3        1:1763               500      500   100.00
a40db30  4        U-B----  4        1:2263               500      500   100.00
a40db70  5        U-B----  5        1:2763               500      500   100.00
a40dbb0  6        U---C-L  6        1:3263               500      372    74.40
a40dbf0  7        A------  0        1:3763               500        0     0.00
a40dc30  8        A------  0        1:4263               500        0     0.00
 8 active, 8 total
```

Output description for the physical log files

The first section of the display describes the physical-log configuration:

buffer
: Is the number of the physical-log buffer

bufused
: Is the number of pages of the physical-log buffer that are used

bufsize
: Is the size of each physical-log buffer in pages

numpages
: Is the number of pages written to the physical log

numwrits
Is the number of writes to disk
pages/io
Is calculated as `numpages/numwrits`
This value indicates how effectively physical-log writes are being buffered.

phybegin
Is the physical page number of the beginning of the log
physize
Is the size of the physical log in pages
phypos
Is the current position in the log where the next log-record write is to occur
phyused
Is the number of pages used in the log
%used
Is the percent of pages used

The second section of the **onstat -l** command output describes the logical-log configuration:

buffer
Is the number of the logical-log buffer
bufused
Is the number of pages used in the logical-log buffer
bufsize
Is the size of each logical-log buffer in pages
numrecs
Is the number of records written
numpages
Is the number of pages written
numwrits
Is the number of writes to the logical log
recs/pages
Is calculated as `numrecs/numpages`
You cannot affect this value. Different types of operations generate different types (and sizes) of records.

pages/io
is calculated as `numpages/numwrits`
You can affect this value by changing the size of the logical-log buffer (specified as LOGBUFF in the ONCONFIG file) or by changing the logging mode of the database (from buffered to unbuffered, or vice versa).

The following fields are repeated for each logical-log file:

address
Is the address of the log-file descriptor
number
Is logid number for the logical-log file
The logid numbers might be out of sequence because either the database server or administrator can insert a log file in-line.

flags
Provides the status of each log as follows:

A
Newly added (and ready to use)
B
Backed up
C
Current® logical-log file
D
Marked for deletion
To drop the log file and free its space for reuse, you must perform a level-0 backup of all storage spaces

F
Free, available for use
L
The most recent checkpoint record
U
Used

uniqid
Is the unique ID number of the log
begin
Is the beginning page of the log file
size
Is the size of the log in pages
used
Is the number of pages used
%used
Is the percent of pages used
active
Is the number of active logical logs
total
Is the total number of logical logs

Output description for temporary logical log files

The database server uses *temporary logical logs* during a warm restore because the permanent logs are not available then. The following fields are repeated for each temporary logical-log file:

address
Is the address of the log-file descriptor
number
Is logid number for the logical-log file
flags
Provides the status of each log as follows:

B
Backed up
C
Current logical-log file
F
Free, available for use
U
Used

uniqid
Is the unique ID number of the log
begin
Is the beginning page of the log file
size
Is the size of the log in pages
used
Is the number of pages used
%used
Is the percent of pages used
active
Is the number of active temporary logical logs

Related reference:
LOGBUFF configuration parameter
PHYSBUFF configuration parameter

onstat -L command: Print the number of free locks

Use the **onstat -L** command to print the number of free locks on a lock-free list.

```
Syntax:

>>-onstat-- -L-------------------------------------------------><
```

Example output

Figure 1. **onstat -L** output

```
num   list head     available locks
0     10a143b70     19996
1     101010101     200
3     020202020     300
```

Output description

num
The list number
list head
The starting address of the list
available locks
The number of locks on this list

onstat -m command: Print recent system message log information

Use the **onstat -m** command to display the 20 most recent lines of the system message log.

You can use the **onstat -m** command option with the database server in any mode, including offline.

Syntax:

```
>>-onstat-- -m-----------------------------------------------><
```

Example output

Output from this command lists the full pathname of the message log file and the 20 file entries. A date-and-time header separates the entries for each day. A time stamp prefaces single entries within each day. The name of the message log is specified as MSGPATH in the **ONCONFIG** file.

Figure 1. **onstat -m** command output

```
Message Log File: /work/11.50/dbspaces/star3.log
11:26:33  Checkpoint Completed:  duration was 0 seconds.
11:26:33  Checkpoint loguniq 1, logpos 0x23c408, timestamp: 0x2cc2 Interval: 9
```


onstat -o command: Output shared memory contents to a file

Use the **onstat -o** command to write the contents of shared memory to a specified file for later analysis. If you do not specify an output file, a file named **onstat.out** is created in the current directory.

Syntax:

```
>>-onstat-- -o--+---------+----+---------+---------------------><
               +-nobuffs-+    '-outfile-'
               '-full----'
```

Use the **nobuffs** option to exclude the buffer pool in the resident segment of shared memory from the output file. This results in a smaller output file.

Use the **full** option to create an output file that is the same size as the shared memory segments for the IBM® Informix® instance. You must have enough room in the file system to handle the output.

If you do not specify either the **nobuffs** or the **full** option, the output is controlled by the database server DUMPSHMEM configuration parameter setting:

- If DUMPSHMEM is set to 0 or to 1, **onstat -o** command writes a full shared-memory dump file.
- If DUMPSHMEM is set to 2, **onstat -o** command writes a **nobuffs** shared-memory dump file that excludes the buffer pool in the resident segment.

By running additional **onstat** commands against the file, you can gather information from a previously saved shared memory dump. The *outfile* that you create with the **onstat -o** command is the *infile* that you can use as a source file to run the additional **onstat** commands. For more information, see Running onstat Commands on a Shared Memory Dump File.

Related reference:
DUMPSHMEM configuration parameter (UNIX)

onstat -p command: Print profile counts

Use the **onstat -p** command to display information about profile counts either since you started the database server or since you ran the **onstat -z** command.

Syntax:

```
>>-onstat-- -p-----------------------------------------------><
```

Example output

Figure 1. **onstat -p** command output

```
Profile
dskreads  pagreads   bufreads   %cached dskwrits   pagwrits   bufwrits   %cached
16934     47321      203600361  99.99   103113     158697     950932     89.16

isamtot   open     start    read     write   rewrite  delete  commit  rollbk
139214865 9195777  12257208 94191268 362691  55696    38134   128294  24

gp_read   gp_write   gp_rewrt   gp_del     gp_alloc   gp_free   gp_curs
39        2          27         51         0          0         16

ovlock    ovuserthread   ovbuff   usercpu  syscpu   numckpts   flushes
0         0              0        1551.59  144.82   1822       1822

bufwaits  lokwaits  lockreqs   deadlks  dltouts  ckpwaits  compress  seqscans
176       1         195872383  0        0        1         39331     1259170
```

```
ixda-RA   idx-RA     da-RA    logrec-RA  RA-pgsused lchwaits
0         7594       2124     0          2002       18848
```

Output description

The first portion of the output describes reads and writes.

Reads and writes are tabulated in three categories: from disk, from buffers, and number of pages (read or written).

The first **%cached** field is a measure of the number of reads from buffers compared to reads from disk. The second **%cached** field is a measure of the number of writes to buffers compared to writes to disk.

The database server buffers the information and writes the information to the disk in pages. For this reason, the number of disk writes displayed as `dskwrits` is usually less than the number of writes that an individual user runs:

dskreads
The number of actual reads from disk
pagreads
The number of pages read
bufreads
Is the number of reads from shared memory
%cached
The percent of reads cached in the buffer pool.
If `bufreads` exceeds the maximum integer (or long) value, its internal representation becomes a negative number, but the value appears as `0.0`.

dskwrits
The actual number of physical writes to disk
This number includes the writes for the physical and logical logs reported in **onstat -l** .

pagwrits
The number of pages written
bufwrits
The number of writes to shared memory
%cached
The percent of writes cached in the buffer pool.

The next portion of the **-p** display tabulates the number of times different ISAM calls were executed. The calls occur at the lowest level of operation and do not necessarily correspond one-to-one with SQL statement execution. A single query might generate multiple ISAM calls. These statistics are gathered across the database server and cannot be used to monitor activity on a single database unless only one database is active or only one database exists:

isamtot
The total number of calls
open
Increments when a tblspace is opened
start
Increments the pointer within an index
read
Increments when the read function is called
write
Increments with each write call
rewrite
Increments when an update occurs
delete
Increments when a row is deleted
commit
Increments each time that an **iscommit()** call is made
No one-to-one correspondence exists between this value and the number of explicit COMMIT WORK statements that are executed.

rollbk
Increments when a transaction is rolled back

The next portion of the **onstat -p** command output displays information about generic pages. The Generic Page Manager provides an API for Informix® to manage nonstandard pages in the database server buffer pool. The following table describes the Generic Page Manager fields in the **onstat -p** command output.

gp_read
The number of generic page reads
gp_write
The number of generic page writes
gp_rewrt
The number of generic page updates
gp_del
The number of generic page deletes
gp_alloc
The number of generic page allocations
gp_free
The number of generic pages freed and returned to tblspaces
gp_curs
The number of cursors used against generic pages

The next portion of the **onstat -p** command output displays the number of times that a resource was requested when none was available:

ovlock
Number of times that sessions attempted to exceed the maximum number of locks

For more information, see “LOCKS” on page 1-56.

ovuserthread
The number of times that a user attempted to exceed the maximum number of user threads
ovbuff
The number of times that the database server did not find a free shared-memory buffer
When no buffers are free, the database server writes a dirty buffer to disk and then tries to find a free buffer.

usercpu
Is the total user CPU time that all user threads use, expressed in seconds
This entry is updated every 15 seconds.

syscpu
The total system CPU time that all user threads use, expressed in seconds
This entry is updated every 15 seconds.

numckpts
The number of checkpoints since the boot time
flushes
The number of times that the buffer pool wasflushed to the disk

The next portion of the **onstat -p** command output contains miscellaneous information, as follows:

bufwaits
Increments each time that a user thread must wait for a buffer
lokwaits
Increments each time that a user thread must wait for a lock
lockreqs
Increments each time that a lock is requested
deadlks
Increments each time that a potential deadlock is detected and prevented
dltouts
Increments each time that the distributed deadlock time-out value is exceeded while a user thread is waiting for a lock
ckpwaits
Is the number of checkpoint waits
compress
Increments each time that a data page is compressed
seqscans
Increments for each sequential scan

*
The last portion of the **onstat -p** command output contains the following information:

ixda-RA
The count of read-aheads that go from index leaves to data pages
idx-RA
The count of read-aheads that traverse index leaves
da-RA
The count of data-path-only scans
logrec-RA
The log records that the database server read ahead
RA-pgsused
The number of pages used that the database server read ahead
lchwaits
Stores the number of times that a thread was required to wait for a shared-memory latch
Many latch waits typically results from a high volume of processing activity in which the database server is logging most of the transactions.

Related reference:
DEADLOCK_TIMEOUT configuration parameter

onstat -P command: Print partition information

Use the **onstat -P** command to display the partition number and the pages in the buffer pool for all of the partitions.

```
Syntax:

>>-onstat-- -P-------------------------------------------------><
```

For information about running **onstat -P** on a dump file created without the buffer pool, see Running onstat Commands on a Shared Memory Dump File.

Example output

Figure 1. **onstat -P** command output

```
Buffer pool page size: 2048
partnum  total    btree    data     other    dirty
0        36       1        8        27       0
```

```
1048577  2        0        0        2        0
1048578  4        1        1        2        0
1048579  23       10       12       1        0
1048580  68       31       36       1        0
4194309  3        0        1        2        0

Totals:  3000     786      1779     435      0
Percentages:
Data  59.30
Btree 26.20
Other 14.50

Buffer pool page size: 8192
partnum  total    btree    data     other    dirty
0        999      0        0        999      0
5242881  1        0        0        1        0

Totals:  1000     0        0        1000     0
Percentages:
Data  0.00
Btree 0.00
Other 100.00
```

Output description

Buffer pool page size
: The size, in bytes, of the buffer pool pages.

partnum
: The partition number.

total
: The total number of partitions.

btree
: The number of B-tree pages in the partition.

data
: The number of data pages in the partition.

other
: The number of other pages in the partition.

dirty
: The number of dirty pages in the partition.

---

onstat -r command: Repeatedly print selected statistics

Use the **onstat -r** command to repeatedly print the statistics for other options specified in the command at specified intervals.

Syntax:

```
>>-onstat-- -r--+-------------------------+--------------------><
               +-seconds--other_options-+
               '-other_options----------'
```

Use the **onstat -r *seconds other_options*** command to specify the seconds to repeat the other option.

Use **onstat -r *other_options*** command to have the option repeat every five seconds, which allows the other options to be concatenated with the **-r** option, as in this example: `onstat -rFh`.

The **onstat -r** command can be used in both command mode and interactive mode, and can be useful for repeating command output to monitor system resource utilization.

Example output running the onstat -r command every five seconds

Figure 1. command output

```
onstat -r

IBM Informix Dynamic Server Version 11.70.F       -- On-Line -- Up 20:05:25 -- 1067288 Kbytes

IBM Informix Dynamic Server Version 11.70.F       -- On-Line -- Up 20:05:30 -- 1067288 Kbytes

IBM Informix Dynamic Server Version 11.70.F       -- On-Line -- Up 20:05:35 -- 1067288 Kbytes
```

Example output running the onstat -r command every ten seconds

Figure 2. command output

```
onstat -r 10

IBM Informix Dynamic Server Version 11.50.F       -- On-Line -- Up 20:06:58 -- 1067288 Kbytes
```

```
IBM Informix Dynamic Server Version 11.50.F       -- On-Line -- Up 20:07:08 -- 1067288 Kbytes

IBM Informix Dynamic Server Version 11.50.F       -- On-Line -- Up 20:07:18 -- 1067288 Kbytes
```

Example output running the onstat -r every one second, with the -h option

Figure 3. **onstat -r 1 -h** command output

```
onstat -r 1 -h

Buffer pool page size: 2048

buffer hash chain length histogram
   # of chains          of len
          3841               0
          3767               1
           522               2
            62               3
          8192   total chains
          4351   hashed buffs
          5000    total buffs


Buffer pool page size: 2048

buffer hash chain length histogram
   # of chains          of len
          4020               0
          3392               1
           735               2
            43               3
             2               4
          8192   total chains
          4172   hashed buffs
          5000    total buffs
```

Example output running the onstat -r command every five seconds, with the -Fh options

Figure 4. **onstat -rFh** command output

```
onstat -rFh


Fg Writes     LRU Writes    Chunk Writes
0             0             21

address           flusher  state    data     # LRU    Chunk    Wakeups  Idle Tim
460e6820          0        I        0        0        2        5        9.820
      states: Exit Idle Chunk Lru

Buffer pool page size: 2048

buffer hash chain length histogram
   # of chains          of len
          6342               0
          1850               1
          8192   total chains
          1850   hashed buffs
          5000    total buffs


Fg Writes     LRU Writes    Chunk Writes
0             0             21

address           flusher  state    data     # LRU    Chunk    Wakeups  Idle Tim
460e6820          0        I        0        0        2        10       22.755
      states: Exit Idle Chunk Lru

Buffer pool page size: 2048

buffer hash chain length histogram
   # of chains          of len
          4396               0
          3796               1
          8192   total chains
          3796   hashed buffs
          5000    total buffs
```


onstat -R command: Print LRU, FLRU, and MLRU queue information

Use the **onstat -R** command to display detailed information about the LRU queues, FLRU queues, and MLRU queues. For each queue, the **onstat -R** command displays the number of buffers in the queue and the number and percentage of buffers that have been modified.

For an in-depth discussion of the three types of queues, see LRU queues in the shared-memory chapter of the *IBM® Informix® Administrator's Guide*.

Syntax:

```
>>-onstat-- -R-------------------------------------------------><
```

Example output

Figure 1. **onstat -R** command output

```
        Buffer pool page size: 2048
        8 buffer LRU queue pairs              priority levels
    # f/m   pair total     % of    length       LOW      HIGH
     0 f         375     100.0%      375        375         0
     1 m                   0.0%        0          0         0
     2 f         375     100.0%      375        375         0
     3 m                   0.0%        0          0         0
     4 f         375     100.0%      375        375         0
     5 m                   0.0%        0          0         0
     6 F         375     100.0%      375        375         0
     7 m                   0.0%        0          0         0
     8 f         375     100.0%      375        375         0
     9 m                   0.0%        0          0         0
    10 f         375     100.0%      375        375         0
    11 m                   0.0%        0          0         0
    12 f         375     100.0%      375        375         0
    13 m                   0.0%        0          0         0
    14 f         375     100.0%      375        375         0
    15 m                   0.0%        0          0         0
    0 dirty, 3000 queued, 3000 total, 4096 hash buckets, 2048 buffer size
    start clean at  60.000% (of pair total) dirty, or 226 buffs dirty, stop at
      50.000%
    Buffer pool page size: 8192
        4 buffer LRU queue pairs              priority levels
    # f/m   pair total     % of    length       LOW      HIGH
     0 F         250     100.0%      250        250         0
     1 m                   0.0%        0          0         0
     2 f         250     100.0%      250        250         0
     3 m                   0.0%        0          0         0
     4 f         250     100.0%      250        250         0
     5 m                   0.0%        0          0         0
     6 f         250     100.0%      250        250         0
     7 m                   0.0%        0          0         0
    0 dirty, 1000 queued, 1000 total, 1024 hash buckets, 8192 buffer size
    start clean at  60.000% (of pair total) dirty, or 150 buffs dirty, stop at
      50.000%
```

Output description

Buffer pool page size
: Is the page size of the buffer pool in bytes

#
: Shows the queue number
Each LRU queue is composed of two subqueues: an FLRU queue and a MLRU queue. (For a definition of FLRU and MLRU queues, see LRU queues in the shared-memory chapter of the *IBM Informix Administrator's Guide*.) Thus, queues 0 and 1 belong to the first LRU queue, queues 2 and 3 belong to the second LRU queue, and so on.

f/m
: Identifies queue type
This field has four possible values:

  f
  : Free LRU queue
  In this context, free means not modified. Although nearly all the buffers in an LRU queue are available for use, the database server attempts to use buffers from the FLRU queue rather than the MLRU queue. (A modified buffer must be written to disk before the database server can use the buffer.)

  F
  : Free LRU with fewest elements
  The database server uses this estimate to determine where to put unmodified (free) buffers next.

  m
  : MLRU queue

  M
  : MLRU queue that a flusher is cleaning

length
: Tracks the length of the queue measured in buffers

% of
: Shows the percent of LRU queue that this subqueue composes
For example, suppose that an LRU queue has 50 buffers, with 30 of those buffers in the MLRU queue and 20 in the FLRU queue. The **% of** column would list percents of 60.00 and 40.00, respectively.

pair total
: Provides the total number of buffers in this LRU queue

priority levels

Displays the priority levels: `LOW, MED_LOW, MED_HIGH, HIGH`

The **onstat -R** command also lists the priority levels.

Summary information follows the individual LRU queue information. You can interpret the summary information as follows:

dirty
Is the total number of buffers that have been modified in all LRU queues
queued
Is the total number of buffers in LRU queues
total
Is the total number of buffers
hash buckets
Is the number of hash buckets
buffer size
Is the size of each buffer
start clean
Is the value specified in the **lru_max_dirty** field of the BUFFERPOOL configuration parameter
stop at
Is the value specified in the **lru_min_dirty** field of the BUFFERPOOL configuration parameter
priority downgrades
Is the number of LRU queues downgraded to a lower priority.
priority upgrades
Is the number of LRU queues upgraded to a higher priority.

---

onstat -s command: Print latch information

Use the **onstat -s** command to display general latch information, including the resource that the latch controls.

Syntax:

```
>>-onstat-- -s-------------------------------------------------><
```

Example output

Figure 1. **onstat -s** command output

```
IBM Informix Dynamic Server Version 14.10.F         -- On-Line (CKPT REQ)
-- Up 00:19:21 -- 167608 Kbytes
Blocked:CKPT

Latches with lock or userthread set
name     address           lock wait userthread
physlog  4410d5a8          0    0    45a616b8
```

Output description

name
Identifies the resource that the latch controls with the following abbreviations:

archive
Storage-space backup
bf
Buffers
bh
Hash buffers
chunks
Chunk table
ckpt
Checkpoints
dbspace
Dbspace table
flushctl
Page-flusher control
flushr
Page cleaners
locks
Lock table
loglog
Logical log
LRU
LRU queues
physb1
First physical-log buffer
physb2

Second physical-log buffer

physlog
: Physical log

pt
: Tblspace tblspace

tblsps
: Tblspace table

users
: User table

address
: Is the address of the latch
This address appears in the **onstat -u** (users) command output wait field if a thread is waiting for the latch.

lock
: Indicates if the latch is locked and set
The codes that indicate the lock status (`1` or `0`) are computer dependent.

wait
: Is the shared-memory address of the user thread that is waiting for a latch, or is blank when no user threads are waiting

userthread
: Is the shared-memory address of the user thread that holds the latch
This address corresponds to the value in the **tcb** column of the **onstat -g ath** output. You can compare this address with the user addresses in the **onstat -u** output to obtain the user-process identification number.

---

onstat -t and onstat -T commands: Print tblspace information

Use the **onstat -t** command to display tblspace information for active tblspaces. Use the **onstat -T** command to display tblspace information for all tblspaces.

The **onstat -t** command also lists the number of active tblspaces and the total number of tblspaces.

Syntax:

```
>>-onstat--+- -t-+-----------------------------------------><
           '- -T-'
```

Example output

Figure 1. **onstat -t** command output

```
Tblspaces
 n address  flgs ucnt tblnum   physaddr         npages nused  npdata nrows  nextns
62 a40dc70  0    1    100001   1:14             250    250    0      0      1
195 ac843e0  0    1    1000df   1:236             16     9      4      53     2
 2 active, 221 total
```

Output description

n
: Is a counter of open tblspaces

address
: Is the address of the tblspace in the shared-memory tblspace table

flgs
: Uses the following flag bits to describe the flag:

0x00000002
: Flush the partition info at the next checkpoint.

0x00000004
: Drop partition is in progress.

0x00000008
: Partition is a pseudo partition (sysmaster).

0x00000020
: ALTER TABLE is in progress.

0x00000040
: Partition has been dropped.

0x00000800
: Partition is the system temp table.

0x00001000
: Partition is the user temp table.

0x00008000
: Online index create or drop in progress.

0x00400000
: A single user access to the partition is requested.

0x00800000
: Drop partition is completed.

0x40000000
Flush the partition info. The partition flush can be delayed until later in the checkpoint process.

ucnt
Is the usage count, which indicates the number of user threads currently accessing the tblspace

tblnum
Is the tblspace number expressed as a hexadecimal value
The integer equivalent appears as the **partnum** value in the **systables** system catalog table.

physaddr
Is the physical address (on disk) of the tblspace

npages
Is the number of pages allocated to the tblspace

nused
Is the number of used pages in the tblspace

npdata
Is the number of data pages used

nrows
Is the number of data rows used

nextns
Is the number of noncontiguous extents allocated
This number is not the same as the number of times that a next extent has been allocated.

onstat -u command: Print user activity profile

Use the **onstat -u** command to display a profile of user activity.

Syntax:

```
>>-onstat-- -u-------------------------------------------------><
```

Example output

Figure 1. **onstat -u** command output

```
Userthreads
address  flags   sessid   user     tty      wait     tout locks nreads   nwrites
a4d8018  ---P--D 1        informix -        0        0    0     58       4595
a4d8628  ---P--F 0        informix -        0        0    0     0        2734
a4d8c38  ---P--- 5        informix -        0        0    0     0        1
a4d9248  ---P--B 6        informix -        0        0    0     40       0
a4d9858  ---P--D 7        informix -        0        0    0     0        0
a4d9e68  Y--P--- 21       niraj    -        a65e5a8  0    1     0        0
 6 active, 128 total, 7 maximum concurrent
```

Output description

The **-u** option displays the following output for each user thread.

address
The shared-memory address of the user thread in the user table.
Compare this address with the addresses displayed in the output from the **-s** option (latches); the output from the **-b**, **-B**, and **-X** options (buffers); and the output from the **-k** option (locks) to learn what resources this thread is holding or waiting for.

flags
Provides the status of the session.
The flag codes for position 1:

B
Waiting for a buffer
C
Waiting for a checkpoint
G
Waiting for a write of the logical-log buffer
L
Waiting for a lock
S
Waiting for mutex
T
Waiting for a transaction
Y
Waiting for condition
X
Waiting for a transaction cleanup (rollback)
DEFUNCT
The thread has incurred a serious assertion failure, and has been suspended to allow other threads to continue their work.

The flag code for position 2:

*
: Transaction active during an I/O failure

The flag code for position 3:

A
: A dbspace backup thread

For other values that appear here, see the third position of flag codes for the **-x** option.

The flag code for position 4:

P
: Primary thread for a session

The flag codes for position 5:

R
: Reading

X
: Thread in critical section

The flag codes for position 6:

R
: Thread used in recovery (for example, physical or logical recovery)

-
: Thread not used in recovery

The flag codes for position 7:

B
: A B-tree cleaner thread

C
: Terminated user thread waiting for cleanup

D
: A daemon thread

F
: A page-cleaner thread

sessid
: The session identification number.
During operations such as parallel sorting and parallel index building, a session might have many user threads associated with it. For this reason, the session ID identifies each unique session.

user
: The user login name, which is derived from the operating system

tty
: The name of the standard error (stderr) file that the user is using, which is derived from the operating system.
This field is blank on Windows.

wait
: If the user thread is waiting for a specific latch, lock, mutex, or condition, this field displays the address of the resource. Use this address to map to information provided in the output from the **-s** (latch) option or the **-k** (lock) option. If the wait is for a persistent condition, run a **grep** for the address in the output from the **onstat -a** command.

tout
: The number of seconds left in the current wait
If the value is `0`, the user thread is not waiting for a latch or lock. If the value is `-1`, the user thread is in an indefinite wait.

locks
: The number of locks that the user thread is holding
The **-k** output should include a listing for each lock held.)

nreads
: The number of disk reads that the user thread has executed

nwrites
: The number of write calls that the user thread has executed
All write calls are writes to the shared-memory buffer cache.

The last line of the **onstat -u** command output displays the maximum number of concurrent user threads that were allocated since you initialized the database server. For example, the last line of a sample **onstat -u** command output is as follows:

```
4 active, 128 total, 17 maximum concurrent
```

The last part of the line, `17 maximum concurrent`, indicates that the maximum number of user threads that were running concurrently since you initialized the database server is 17.

The output also indicates the number of active users and the maximum number of users allowed.

onstat -x command: Print database server transaction information

Use the **onstat -x** command to display transaction information on the database server.

Syntax:

```
>>-onstat-- -x-------------------------------------------------><
```

The transaction information is required only in the following situations:

- X/Open environment
- Database server participation in distributed queries
- Database server uses the Microsoft Transaction Server (MTS) transaction manager

Example output

Figure 1. **onstat -x** command output

```
Transactions

address   flags userthread  locks  begin    current     isol    est.     retrys coord
                             logpos   logpos              rb_time
a6d8028   A---- a695028     0      -        -           COMMIT  -        0
a6d8348   A---- a695878     0      -        -           COMMIT  -        0
a6d8668   A---- a6960c8     0      -        -           COMMIT  -        0
a6d8988   A---- a696918     0      -        -           COMMIT  -        0
a6d8fc8   A---- a698208     0      -        -           COMMIT  -        0
a6d92e8   A---- a6979b8     0      -        -           COMMIT  -        0
a6d9608   A---- a698a58     0      -        -           COMMIT  -        0
a6d9928   A---- a6992a8     1      -        -           DIRTY   -        0
a6d9c48   A---- a6992a8     0      -        -           NOTRANS -        0
a6d9f68   A---- a69a348     0      -        -           COMMIT  -        0
a6da288   A---- a69ab98     0      -        -           COMMIT  -        0
a6da5a8   A---- a69b3e8     0      -        -           COMMIT  -        0
a6da8c8   A---- a69bc38     0      -        -           COMMIT  -        0
a6dabe8   A---- a69c488     0      -        -           COMMIT  -        0
a6daf08   A---- a699af8     0      -        -           COMMIT  -        0
a6db228   A---- a6992a8     0      -        -           COMMIT  -        0
a6db548   A---- a69ccd8     1      -        -           DIRTY   -        0
a6db868   A---- a69d528     1      -        -           DIRTY   -        0
a6dbb88   A---- a69ccd8     0      -        -           COMMIT  -        0
a6dbea8   A---- a69dd78     0      -        -           COMMIT  -        0
a6dc1c8   A---- a69e5c8     0      -        -           COMMIT  -        0
a6dc4e8   A-B-- a69ee18   502    33:0x25018  34:0x486fc COMMIT  0:07     0
 22 active, 128 total, 23 maximum concurrent
```

Output description

You can interpret output from the **onstat -x** command as follows:

address
: The shared-memory address of the transaction structure

flags
: The flag codes for position 1 (current transaction state):

  A
  : User thread attached to the transaction

  S
  : TP/XA suspended transaction

  C
  : TP/XA waiting for rollback

  The flag codes for position 2 (transaction mode):

  T
  : Tightly-coupled mode (MTS)

  L
  : Loosely-coupled mode (default mode)

  The flag codes for position 3 (transaction stage):

  B
  : Begin work

  P
  : Distributed query prepared for commit

  X
  : TP/XA prepared for commit

  C
  : Committing or committed

  R
  : Rolling back or rolled back

  H

Heuristically rolling back or rolled back

The flag code for position 4:

X
XA transaction

The flag codes for position 5 (type of transaction):

G
Global transaction
C
Distributed query coordinator
S
Distributed query subordinate
B
Both distributed query coordinator and subordinate
M
Redirected global transaction

userthread
The thread that owns the transaction (**rstcb** address)
locks
The number of locks that the transaction holds
begin logpos
The position within the log when the BEGIN WORK record was logged.
current logpos
The current log position of the most recent record that the transaction is wrote too (As a transaction rolls back, the current log position will actually wind back until it gets to the beginning log position. When the beginning log position is reached, the rollback is complete.)
isol
The isolation level.
est. rb time
The estimated time the server needs to rollback the transaction. As a transaction goes forward, this time increases. If a transaction rolls back, the time decreases as the transaction unwinds.
retrys
Are the attempts to start a recovery thread for the distributed query
coord
The name of the transaction coordinator when the subordinate is executing the transaction
This field tells you which database server is coordinating the two-phase commit.

The last line of the **onstat -x** command output indicates that `8` is the maximum number of concurrent transactions since you initialized the database server.

```
8 active, 128 total, 8 maximum concurrent
```

- Determine the position of a logical-log record
  Use the **onstat -x** command to determine the position of a logical-log record.
- Determine the mode of a global transaction
  The **onstat -x** command is useful for determining whether a global transaction is executing in loosely-coupled or tightly-coupled mode.

Determine the position of a logical-log record

Use the **onstat -x** command to determine the position of a logical-log record.

The **curlog** and **logposit** fields provide the exact position of a logical-log record. If a transaction is not rolling back, **curlog** and **logposit** describe the position of the most recently written log record. When a transaction is rolling back, these fields describe the position of the most recently “undone” log record. As the transaction rolls back, the **curlog** and **logposit** values decrease. In a long transaction, the rate at which the **logposit** and **beginlg** values converge can help you estimate how much longer the rollback is going to take.

For an **onstat -x** command example, see monitoring a global transaction in the chapter on multiphase commit protocols in the *IBM® Informix® Administrator's Guide*.

Determine the mode of a global transaction

The **onstat -x** command is useful for determining whether a global transaction is executing in loosely-coupled or tightly-coupled mode.

The second position of the flags column in the output from the **onstat -x** command displays the flags for global transactions. The `T` flag indicates tightly-coupled mode and the `L` flag indicates loosely-coupled mode.

- *Loosely-coupled mode* means that the different database servers coordinate transactions but do not share locks. Each branch in a global transaction has a separate transaction XID. The records from all branches display as separate transactions in the logical log.
- *Tightly-coupled mode* means that the different database servers coordinate transactions and share resources such as locking and logging. In a global transaction, all branches that access the same database share the same transaction XID. Log records for branches with the same XID appear under the same session ID. MTS uses tightly-coupled mode.

onstat -X command: Print thread information

Use the **onstat -X** command to obtain precise information about the threads that are waiting for buffers.

For each buffer in use, the **onstat -X** command displays general buffer information that is also available with either the **onstat -b** or **onstat -B** commands. For more information, refer to the **onstat -b** command in onstat -b command: Print buffer information for buffers in use.

Syntax:

```
>>-onstat-- -X-------------------------------------------------><
```

Example output

Figure 1. **onstat -X** command output

```
        Buffers (Access)
    address  owner    flags pagenum          memaddr  nslots pgflgs scount   waiter
        Buffer pool page size: 2048
     0 modified, 3000 total, 4096 hash buckets, 2048 buffer size
        Buffer pool page size: 8192
     0 modified, 1000 total, 1024 hash buckets, 8192 buffer size
```

Output description

The **onstat -X** command has a **waiter** field to list all user threads that are waiting for the buffer, whereas the **onstat -b** and **onstat -B** commands contain a **waitlist** field that displays the address of the first user thread that is waiting for the buffer. The maximum number of shared buffers is specified in the **buffers** field in the BUFFERPOOL configuration parameter in the ONCONFIG file.

Buffer pool page size
: The size of the buffer pool pages in bytes

address
: The address of the buffer header in the buffer table

flags
: Flags identifying the current status of the buffer page:

  0x01
  : Modified Data

  0x02
  : Data

  0x04
  : LRU

  0x08
  : Error

  0x10
  : Shared lock

  0x20
  : LRU AIO write in progress

  0x40
  : Chunk write in progress

  0x80
  : Exclusive lock

  0x100
  : Cleaner assigned to LRU

  0x200
  : Buffer should avoid bf_check calls

  0x400
  : Do log flush before writing page

  0x800
  : Buffer has been 'buff' -checked

  0x8000
  : Buffer has been pinned

pagenum
: The physical page number on the disk

memaddr
: The buffer memory address

nslots
: The number of slot-table entries in the page
  This field indicates the number of rows (or portions of a row) that are stored on the page.

pgflgs
: Uses the following values, alone or in combination, to describe the page type:

  1
  : Data page

  2

Tblspace page

4
Free-list page

8
Chunk free-list page

9
Remainder data page

b
Partition resident blobpage

c
Blobspace resident blobpage

d
Blob chunk free-list bit page

e
Blob chunk blob map page

10
B-tree node page

20
B-tree root-node page

40
B-tree branch-node page

80
B-tree leaf-node page

100
Logical-log page

200
Last page of logical log

400
Sync page of logical log

800
Physical log

1000
Reserved root page

2000
No physical log required

8000
B-tree leaf with default flags

scount
Displays the number of threads that are waiting for the buffer

waiter
Lists the addresses of all user threads that are waiting for the buffer

onstat -z command: Clear statistics

Use the **onstat -z** command to clear database server statistics, including statistics that relate to Enterprise Replication, and set the profile counts to 0.

If you use the **onstat -z** command to reset and monitor the count of some fields, be aware that profile counts are incremented for all activity that occurs in any database that the database server manages. Any user can reset the profile counts and thus interfere with monitoring that another user is conducting.

Syntax:

```
>>-onstat-- -z-------------------------------------------------><
```


Return codes on exiting the onstat utility

The **onstat** utility displays a set of return codes when you exit the utility.

Example

The following lines are an example of the messages and return codes that are displayed when you exit the **onstat** utility:

```
GLS failures: -1
Failed to attach shared memory: -1
Failed to attach shared memory when running 'onstat -': 255
All other errors detected by onstat: 1
No errors detected by onstat: 0
Administration mode: 7
```

Return code values

The following table lists the database server mode that corresponds to the return codes that are displayed when you exit the **onstat** utility.

Value	Explanation
-1	GLS locale initialization failed or failed to attach to shared memory
0	Initialization mode
1	Quiescent mode
2	Recovery mode
3	Backup mode
4	Shutdown mode
5	Online mode
6	Abort mode
7	User mode
255	Off-Line mode

SQL Administration API

- SQL Administration API Functions
  These topics describe the SQL administration API **admin()** and **task()** functions.

SQL Administration API Functions

These topics describe the SQL administration API **admin()** and **task()** functions.

- SQL Administration API Overview
  Use the SQL administration API to remotely administer through SQL statements.
- SQL administration API portal: Arguments by privilege groups
  You can view a list of **admin()** and **task()** function arguments, which are sorted by privilege groups, with links to information about the arguments.
- add bufferpool argument: Add a buffer pool (SQL administration API)
  Use the add bufferpool argument with the **admin()** or **task()** function to create a buffer pool.
- add chunk argument: Add a new chunk (SQL administration API)
  Use the add chunk argument with the **admin()** or **task()** function to add a chunk to a dbspace or blobspace.
- add log argument: Add a new logical log (SQL administration API)
  Use the add log argument with the **admin()** or **task()** function to add a logical log to a dbspace.
- add memory argument: Increase shared memory (SQL administration API)
  Use the add memory argument with the **admin()** or **task()** function to add to the virtual portion of shared memory.
- add mirror argument: Add a mirror chunk (SQL administration API)
  Use the add mirror argument with the **admin()** or **task()** function to add a mirror chunk to a dbspace.
- alter chunk argument: Change chunk status to online or offline (SQL administration API)
  Use the alter chunk argument with the **admin()** or **task()** function to bring a chunk online or take a chunk offline in a dbspace, blobspace, or sbspace.
- alter logmode argument: Change the database logging mode (SQL administration API)
  Use the alter logmode argument with the **admin()** or **task()** function to change the database logging mode to ANSI, buffered, non-logging, or unbuffered.
- alter plog argument: Change the physical log (SQL administration API)
  Use the alter plog argument with the **admin()** or **task()** function to change the location and size of the physical log.
- archive fake argument: Perform an unrecorded backup (SQL administration API)
  Use the archive fake argument with the **admin()** or **task()** function to perform a backup operation to clone the data in a server without creating a persistent backup that could be used to perform a restore.
- autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
  Use the autolocate database add argument with the **admin()** or **task()** function to add a dbspace to the list of available dbspaces for the automatic location and fragmentation of tables for the specified database.
- autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)
  Use the autolocate database anywhere argument with the **admin()** or **task()** function to specify that the database server can use any non-critical dbspace for the automatic location and fragmentation of tables for the specified database.
- autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)
  Use the autolocate database argument with the **admin()** or **task()** function to specify the list of available dbspaces for the automatic location and fragmentation of tables for the specified database.
- autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)
  Use the autolocate database off argument with the **admin()** or **task()** function to disable the automatic location and fragmentation of tables for a specified database.
- autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
  Use the autolocate database remove argument with the **admin()** or **task()** function to remove a dbspace from the list of available dbspaces into which the database server can automatically locate and fragment tables for the specified database.
- cdr argument: Administer Enterprise Replication (SQL administration API)
  Use the cdr argument with the **admin()** or **task()** function to administer Enterprise Replication.

- cdr add trustedhost argument: Add trusted hosts (SQL administration API)
  Use the cdr add trustedhost argument with the **admin()** or **task()** function to add trusted-host relationships for database servers in a high-availability cluster or Enterprise Replication domain. For a database to participate in a high-availability cluster or Enterprise Replication domain, its host must be listed in the trusted-host files of the other high-availability or replication servers.
- cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API)
  The cdr autoconfig serv argument with the **admin()** or **task()** function can autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can autoconfigure replication.
- cdr list trustedhost argument: List trusted hosts (SQL administration API)
  Use the cdr list trustedhost argument with the **admin()** or **task()** function to list trusted-host information from the file that is specified by the database server's REMOTE_SERVER_CFG configuration parameter.
- cdr remove trustedhost argument: Remove trusted hosts (SQL administration API)
  Use the cdr remove trustedhost argument with the **admin()** or **task()** function to remove entries from a database server's trusted-host file.
- check data argument: Check data consistency (SQL administration API)
  Use the check data argument with the **admin()** or **task()** function to check or repair all pages in the specified partition for consistency.
- check extents argument: Check extent consistency (SQL administration API)
  Use the check extents argument with the **admin()** or **task()** function to verify that the extents on disk correspond to the current control information.
- check partition argument: Check partition consistency (SQL administration API)
  Use the check partition argument with the **admin()** or **task()** function to print tblspace information for a table or fragment.
- checkpoint argument: Force a checkpoint (SQL administration API)
  Use the checkpoint argument with the **admin()** or **task()** function to force a checkpoint.
- clean sbspace argument: Release unreferenced smart large objects (SQL administration API)
  Use the clean sbspace argument with the **admin()** or **task()** function to release any unreferenced BLOB or CLOB objects from the sbspace.
- create blobspace argument: Create a blobspace (SQL administration API)
  Use the create blobspace argument with the **admin()** or **task()** function to create a blobspace.
- create blobspace from storagepool argument: Create a blobspace from the storage pool (SQL administration API)
  Use the create blobspace from storagepool argument with the **admin()** or **task()** function to create a blobspace from an entry from the storage pool.
- create chunk argument: Create a chunk (SQL administration API)
  Use the create chunk argument with the **admin()** or **task()** function to create a chunk in a dbspace or in a blobspace.
- create chunk from storagepool argument: Create a chunk from the storage pool (SQL administration API)
  Use the create chunk from storagepool argument with the **admin()** or **task()** function to manually create a chunk from an entry in the storage pool.
- create database argument: Create a database (SQL administration API)
  Use the create database argument with the **admin()** or **task()** function to create a database.
- create dbaccessdemo argument: Create the demonstration database (SQL administration API)
  Use the create dbaccessdemo argument with the **admin()** or **task()** function to create the **stores_demo** demonstration database.
- create dbspace argument: Create a dbspace (SQL administration API)
  Use the create dbspace argument with the **admin()** or **task()** function to create a dbspace.
- create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API)
  Use the create dbspace from storagepool argument with the **admin()** or **task()** function to create a permanent dbspace from an entry in the storage pool.
- create plogspace: Create a plogspace (SQL administration API)
  Use the create plogspace argument with the **admin()** or **task()** function to create a plogspace in which to store the physical log.
- create sbspace argument: Create an sbspace (SQL administration API)
  Use the create sbspace argument with the **admin()** or **task()** function to create an sbspace.
- create sbspace from storagepool argument: Create an sbspace from the storage pool (SQL administration API)
  Use the create sbspace from storagepool argument with the **admin()** or **task()** function to create an sbspace from an entry from the storage pool.
- create sbspace with accesstime argument: Create an sbspace that tracks access time (SQL administration API)
  Use the create sbspace with accesstime argument with the **admin()** or **task()** function to create an sbspace that tracks the time of access for all smart large objects stored in the sbspace.
- create sbspace with log argument: Create an sbspace with transaction logging (SQL administration API)
  Use the create sbspace with log argument with the **admin()** or **task()** function to create an sbspace with transaction logging turned on.
- create tempdbspace argument: Create a temporary dbspace (SQL administration API)
  Use the **create tempdbspace** argument with the **admin()** or **task()** function to create a temporary dbspace.
- create tempdbspace from storagepool argument: Create a temporary dbspace from the storage pool (SQL administration API)
  Use the create tempdbspace from storagepool argument with the **admin()** or **task()** function to create a temporary dbspace from an entry from the storage pool.
- create tempsbspace argument: Create a temporary sbspace (SQL administration API)
  Use the create sbspace argument with the **admin()** or **task()** function to create an sbspace.
- create tempsbspace from storagepool argument: Create a temporary sbspace from the storage pool (SQL administration API)
  Use the create tempsbspace from storagepool argument with the **admin()** or **task()** function to create a temporary sbspace from an entry from the storage pool.
- defragment argument: Dynamically defragment partition extents (SQL administration API)
  Use the defragment argument with the **admin()** or **task()** function to defragment tables or indexes to merge non-contiguous extents.
- drop blobspace argument: Drop a blobspace (SQL administration API)
  Use the drop blobspace argument with the **admin()** or **task()** function to drop the specified blobspace.
- drop blobspace to storagepool argument: Return space from an empty blobspace to the storage pool (SQL administration API)
  Use the drop blobspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty blobspace to the storage pool.
- drop chunk argument: Drop a chunk (SQL administration API)
  Use the drop chunk argument with the **admin()** or **task()** function to drop the specified chunk from a dbspace, blobspace, or sbspace.
- drop chunk to storagepool argument: Return space from an empty chunk to the storage pool (SQL administration API)
  Use the drop chunk to storagepool argument with the **admin()** or **task()** function to return the space from an empty chunk to the storage pool.
- drop database argument: Drop a database (SQL administration API)
  Use the drop database argument with the **admin()** or **task()** function to drop a database.
- drop dbspace argument: Drop a dbspace (SQL administration API)
  Use the drop dbspace argument with the **admin()** or **task()** function to drop the specified dbspace.
- drop dbspace to storagepool argument: Return space from an empty dbspace to the storage pool (SQL administration API)
  Use the drop dbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty dbspace to the storage pool.
- drop log argument: Drop a logical log (SQL administration API)
  Use the drop log argument with the **admin()** or **task()** function to drop the specified logical log.
- drop plogspace: Drop the plogspace (SQL administration API)
  Use the drop plogspace argument with the **admin()** or **task()** function to drop the plogspace.
- drop sbspace argument: Drop an sbspace (SQL administration API)
  Use the drop sbspace argument with the **admin()** or **task()** function to drop the specified sbspace.

- drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API)
  Use the drop sbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty sbspace to the storage pool.
- drop tempdbspace argument: Drop a temporary dbspace (SQL administration API)
  Use the drop tempdbspace argument with the **admin()** or **task()** function to drop the specified temporary dbspace.
- drop tempdbspace to storagepool argument: Return space from an empty temporary dbspace to the storage pool (SQL administration API)
  Use the drop tempdbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty temporary dbspace to the storage pool.
- drop tempsbspace to storagepool argument: Return space from an empty temporary sbspace to the storage pool (SQL administration API)
  Use the drop tempsbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty temporary sbspace to the storage pool.
- export config argument: Export configuration parameter values (SQL administration API)
  Use the export config argument with the **admin()** or **task()** function to export a file that contains all configuration parameters and their current values.
- file status argument: Display the status of a message log file (SQL administration API)
  Use the file status argument with the **admin()** or **task()** function to specify the status of an online, ON-Bar activity, or ON-Bar debug message log file.
- grant admin argument: Grant privileges to run SQL administration API commands
  Use the grant admin argument with the **admin()** or **task()** function to grant privileges to run SQL administration API commands.
- ha make primary argument: Change the mode of a secondary server (SQL administration API)
  Use the ha make primary argument with the **admin()** or **task()** function to change the specified secondary server to a primary or standard server.
- ha rss argument: Create an RS secondary server (SQL administration API)
  Use the ha rss argument with the **admin()** or **task()** function to create a remote standalone (RS) secondary server.
- ha rss add argument: Add an RS secondary server to a primary server (SQL administration API)
  Use the ha rss add argument with the **admin()** or **task()** function to associate a primary server with a remote standalone (RS) secondary server.
- ha rss change argument: Change the password of an RS secondary server (SQL administration API)
  Use the ha rss change argument with the **admin()** or **task()** function to change the connection password for the specified RS secondary server.
- ha rss delete argument: Delete an RS secondary server (SQL administration API)
  Use the ha rss delete argument with the **admin()** or **task()** function to stop replication and delete the RS secondary server.
- ha sds clear argument: Stop shared-disk replication (SQL administration API)
  Use the ha sds clear argument with the **admin()** or **task()** function to stop replication to shared disk (SD) secondary servers and convert the primary server to a standard server.
- ha sds primary argument: Convert an SD secondary server to a primary server (SQL administration API)
  Use the ha sds primary argument with the **admin()** or **task()** function to change a shared disk (SD) secondary server to a primary server.
- ha sds set argument: Create a shared-disk primary server (SQL administration API)
  Use the ha sds set argument with the **admin()** or **task()** function to define a primary server to replicate to shared disk (SD) secondary servers.
- ha set idxauto argument: Replicate indexes to secondary servers (SQL administration API)
  Use the ha set idxauto argument with the **admin()** or **task()** function to control whether indexes are automatically replicated to secondary servers.
- ha set ipl argument: Log index builds on the primary server (SQL administration API)
  Use the ha set ipl argument with the **admin()** or **task()** function to control whether to log index builds on the primary server.
- ha set primary argument: Define an HDR primary server (SQL administration API)
  Use the ha set primary argument with the **admin()** or **task()** function to define a High-Availability Data Replication (HDR) primary server and specify the secondary server.
- ha set secondary argument: Define an HDR secondary server (SQL administration API)
  Use the ha set secondary argument with the **admin()** or **task()** function to define a High-Availability Data Replication (HDR) secondary server and specify the primary server.
- ha set standard argument: Convert an HDR server into a standard server (SQL administration API)
  Use the **ha set standard** argument with the **admin()** or **task()** function to convert a High-Availability Data Replication (HDR) primary or secondary server to a standard server.
- ha set timeout argument: Change SD secondary server timeout (SQL administration API)
  Use the ha set timeout argument with the **admin()** or **task()** function to change the amount of time in seconds that the primary server waits for acknowledgments from shared disk (SD) secondary servers.
- import config argument: Import configuration parameter values (SQL administration API)
  Use the import config argument with the **admin()** or **task()** function to import a file that contains one or more dynamically updatable configuration parameters and apply the new values.
- index compress repack shrink arguments: Optimize the storage of B-tree indexes (SQL administration API)
  Use the index compress repack shrink argument with the **admin()** or **task()** function to compress detached B-tree indexes, consolidate free space (repack), and return free space (shrink) in partitions.
- index estimate_compression argument: Estimate index compression (SQL administration API)
  Use the index estimate_compression argument with the **admin()** or **task()** function to estimate if you can save disk space by compressing a B-tree index.
- master_key reset argument: Change the keystore password (SQL administration API)
  Use the master_key reset argument with the **admin()** or **task()** function to change the password for the storage space encryption keystore. The password is used to encrypt the keystore for storage space encryption.
- message log delete argument: Delete a message log file (SQL administration API)
  Use the message log delete argument or file delete argument with the **admin()** or **task()** function to specify the particular online, ON-Bar activity, or ON-Bar debug message log to delete.
- message log rotate argument: Rotate the message log file (SQL administration API)
  Use the message log rotate argument or the file rotate argument with the **admin()** or **task()** function to specify the particular online, ON-Bar activity, or ON-Bar debug message log file to rotate, and to indicate the maximum number of message logs to rotate.
- message log truncate argument: Delete the contents of a message log file (SQL administration API)
  Use the message log truncate argument or file truncate argument with the **admin()** or **task()** function to specify the particular online, ON-Bar activity, or ON-Bar debug message log file to truncate. When the database server truncates a message log file, it deletes the messages in the log file, but keeps the log file.
- modify chunk extend argument: Extend the size of a chunk (SQL administration API)
  Use the modify chunk extend argument with the **admin()** or **task()** function to extend the size of the chunk by a specified minimum amount. The chunk must be marked as extendable.
- modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)
  Use the modify chunk extendable argument with the **admin()** or **task()** function to specify that a particular chunk in an unmirrored dbspace or temporary dbspace can be extended..
- modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API)
  Use the modify chunk extendable off argument with the **admin()** or **task()** function to specify that a particular chunk cannot be extended.
- modify chunk swap_mirror argument: Switch primary and mirror chunk files without any downtime(SQL administration API)
  Use the modify chunk swap_mirror argument with the **admin()** or **task()** function to easily migrate data from old disk drives to new ones without downtime.
- modify space swap_mirror argument: Switch all primary and mirror chunk files for a space without any downtime (SQL administration API)
  Use the modify space swap_mirrors argument with the **admin()** or **task()** function to easily migrate data from old disk drives to new ones without downtime.

- modify config arguments: Modify configuration parameters (SQL administration API)
  Use the modify config argument with the **admin()** or **task()** function to change the value of a configuration parameter in memory until you restart the database server. Use the modify config persistent argument to change the value of a configuration parameter in memory and preserve the value in the onconfig file after you restart the server.
- modify space expand argument: Expand the size of a space (SQL administration API)
  Use the modify space expand argument with the **admin()** or **task()** function to immediately expand the size of a space, when you do not want to wait for Informix® to automatically expand the space.
- modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)
  Use the modify space sp_sizes argument with the **admin()** or **task()** function to modify the create, extend, and maximum sizes that are associated with expanding a storage space. Modify the sizes to control how Informix uses storage pool entries for a particular storage space.
- onbar argument: Backup the storage spaces (SQL administration API)
  Use the onbar argument with the **admin()** or **task()** function to backup the storage spaces.
- onmode and a arguments: Add a shared-memory segment (SQL administration API)
  Use the onmode and a arguments with the **admin()** or **task()** function to add a shared-memory segment.
- onmode and c arguments: Force a checkpoint (SQL administration API)
  Use the onmode and c arguments with the **admin()** or **task()** function to force a checkpoint.
- onmode and C arguments: Control the B-tree scanner (SQL administration API)
  Use the onmode and C arguments with the **admin()** or **task()** function to control the B-tree scanner for cleaning indexes of deleted items.
- onmode and d arguments: Set data-replication types (SQL administration API)
  Use the onmode and d arguments with the **admin()** or **task()** function to change the mode of a server participating in high-availability data replication (HDR).
- onmode and D arguments: Set PDQ priority (SQL administration API)
  Use the onmode and D arguments with the **admin()** or **task()** function to temporarily reset the PDQ resources that the database server can allocate to any one decision-support query.
- onmode and e arguments: Change usage of the SQL statement cache (SQL administration API)
  Use the onmode and e arguments with the **admin()** or **task()** function to temporarily change the mode of the SQL statement cache.
- onmode and F arguments: Free unused memory segments (SQL administration API)
  Use the onmode and F arguments with the **admin()** or **task()** function to free unused memory segments.
- onmode and h arguments: Update sqlhosts caches (SQL administration API)
  Use the onmode and h arguments with the **admin()** or **task()** function to Update sqlhosts caches.
- onmode and j arguments: Switch the database server to administration mode (SQL administration API)
  Use the onmode and j arguments with the **admin()** or **task()** function to change the database server to administration mode.
- onmode and l arguments: Switch to the next logical log (SQL administration API)
  Use the onmode and l arguments with the **admin()** or **task()** function to switch the current logical-log file to the next logical-log file.
- onmode and m arguments: Switch to multi-user mode (SQL administration API)
  Use the onmode and m arguments with the **admin()** or **task()** function to change the database server to multi-user mode.
- onmode and M arguments: Temporarily change decision-support memory (SQL administration API)
  Use the onmode and M arguments with the **admin()** or **task()** function to temporarily change the size of memory available for parallel queries.
- onmode and n arguments: Unlock resident memory (SQL administration API)
  Use the onmode and n arguments with the **admin()** or **task()** function to end forced residency of the resident portion of shared memory.
- onmode and O arguments: Mark a disabled dbspace as down (SQL administration API)
  Use the onmode and O arguments with the **admin()** or **task()** function to mark a disabled dbspace as down so that the checkpoint that is being blocked by the disabled dbspace can continue and any blocked threads are released.
- onmode and p arguments: Add or remove virtual processors (SQL administration API)
  Use the onmode and p arguments with the **admin()** or **task()** function to dynamically add or remove virtual processors for the current database server session. This function does not update the onconfig file.
- onmode and Q arguments: Set maximum number for decision-support queries (SQL administration API)
  Use the onmode and Q arguments with the **admin()** or **task()** function to change the maximum number of concurrently executing decision-support queries.
- onmode and r arguments: Force residency of shared memory (SQL administration API)
  Use the onmode and r arguments with the **admin()** or **task()** function to start forced residency of the resident portion of shared memory.
- onmode and S arguments: Set maximum number of decision-support scans (SQL administration API)
  Use the onmode and S arguments with the **admin()** or **task()** function to change the maximum number of concurrently executing decision-support scans for the current session.
- onmode and W arguments: Reset statement cache attributes (SQL administration API)
  Use the onmode and W arguments with the **admin()** or **task()** function to change whether and when a statement can be inserted into the SQL cache.
- onmode and wf arguments: Permanently update a configuration parameter (SQL administration API)
  Use the onmode and wf arguments with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in the **onconfig** file.
- onmode and wm arguments: Temporarily update a configuration parameter (SQL administration API)
  Use the onmode and wm arguments with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in memory.
- onmode, wm, and AUTO_LRU_TUNING arguments: Change LRU tuning status (SQL administration API)
  Use the onmode, wm, and AUTO_LRU_TUNING arguments with the **admin()** or **task()** function to change the LRU tuning status without updating the onconfig file. .
- onmode and Y arguments: Change query plan measurements for a session (SQL administration API)
  Use the onmode and Y arguments with the **admin()** or **task()** function to change the output of query plan measurements for an individual session.
- onmode and z arguments: Terminate a user session (SQL administration API)
  Use the onmode and z arguments with the **admin()** or **task()** function to terminate the specified user session.
- onmode and Z arguments: Terminate a distributed transaction (SQL administration API)
  Use the onmode and Z arguments with the **admin()** or **task()** function to terminate the specified distributed transaction. Use this function only if communication between the participating database servers has been lost. If applications are performing distributed transactions, terminating one of the distributed transactions can leave your client/server database system in an inconsistent state.
- onsmsync argument: Synchronize with the storage manager catalog (SQL administration API)
  Use the onsmsync argument with the **admin()** or **task()** function to synchronize the **sysutils** database and emergency boot file with the storage manager catalog.
- onstat argument: Monitor the database server (SQL administration API)
  Use the onstat argument with the **admin()** or **task()** function to monitor the database server.
- ontape archive argument: Backup the data on your database (SQL administration API)
  Use the ontape archive argument with the **admin()** or **task()** function to create a backup of your database data.
- print error argument: Print an error message (SQL administration API)
  Use the print error argument with the **admin()** or **task()** function to print the message associated with the specified error number.
- print file info argument: Display directory or file information (SQL administration API)
  Use the print file info argument with the **admin()** or **task()** function to display information about a directory or a file
- print partition argument: Print partition information (SQL administration API)
  Use the print partition argument with the **admin()** or **task()** function to print the headers of a specified partition.

- rename space argument: Rename a storage space (SQL administration API)
  Use the rename space argument with the **admin()** or **task()** function to rename a dbspace, blobspace, sbspace, or extspace.
- reset config argument: Revert configuration parameter value (SQL administration API)
  Use the reset config argument with the **admin()** or **task()** function to revert the value of a dynamically updatable configuration parameter to its value in the onconfig file. Dynamically updatable configuration parameters are those parameters that you can change for a session with an **onmode** or SQL administration API command.
- reset config all argument: Revert all dynamically updatable configuration parameter values (SQL administration API)
  Use the reset config all argument with the **admin()** or **task()** function to revert the values of all dynamically updatable configuration parameter to their values in the onconfig file. Dynamically updatable configuration parameters are those parameters that you can change for a session with an **onmode** or SQL administration API command.
- reset sysadmin argument: Move the sysadmin database (SQL administration API)
  Use the reset sysadmin argument with the **admin()** or **task()** function to move the **sysadmin** database to the specified dbspace. Moving the **sysadmin** database resets the database back to the original state when it was first created; all data, **command history**, and results tables are lost. Only built-in tasks, sensors, and thresholds remain in the **sysadmin** tables.
- restart listen argument: Stop and start a listen thread dynamically (SQL administration API)
  Use the restart listen argument with the **admin()** or **task()** function to stop and then start an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.
- revoke admin argument: Revoke privileges to run SQL administration API commands
  Use the revoke admin argument with the **admin()** or **task()** function to revoke privileges to run SQL administration API commands.
- scheduler argument: Stop or start the scheduler (SQL administration API)
  Use the scheduler argument with the **admin()** or **task()** function to start or stop the scheduler.
- scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
  Use the scheduler lmm enable argument with the **admin()** or **task()** function to start automatic low memory management and to update low memory threshold settings.
- scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)
  Use the scheduler lmm disable argument with the **admin()** or **task()** function to stop the current and subsequent invocations of automatic low memory management.
- set chunk argument: Change the status of a chunk (SQL administration API)
  Use the set chunk argument with the **admin()** or **task()** function to change the status of a blobspace, dbspace, or sbspace to online or offline.
- set dataskip argument: Start or stop skipping a dbspace (SQL administration API)
  Use the set dataskip argument with the **admin()** or **task()** function to specify whether the database server skips a dbspace that is unavailable during the processing of a transaction.
- set index compression argument: Change index page compression (SQL administration API)
  Use the set index compression argument with the **admin()** or **task()** function to modify the level at which two partially used index pages are merged.
- set onconfig memory argument: Temporarily change a configuration parameter (SQL administration API)
  Use the set onconfig memory argument with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in memory.
- set onconfig permanent argument: Permanently change a configuration parameter (SQL administration API)
  Use the set onconfig permanent argument with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in the **onconfig** file.
- set sbspace accesstime argument: Control access time tracking (SQL administration API)
  Use the set sbspace accesstime argument with the **admin()** or **task()** function to start or stop tracking the time of access for all smart large objects stored in the sbspace.
- set sbspace avg_lo_size argument: Set the average size of smart large objects (SQL administration API)
  Use the set sbspace avg_lo_size argument with the **admin()** or **task()** function to specify an expected average size of the smart large objects in the specified sbspace so that the database server can calculate the size of the metadata area.
- set sbspace logging argument: Change the logging of an sbspace (SQL administration API)
  Use the set sbspace logging argument with the **admin()** or **task()** function to specify whether the database server logs changes to the user data area of the sbspace.
- set sql tracing argument: Set global SQL tracing (SQL administration API)
  Use the set sql tracing argument with the **admin()** or **task()** function to set global SQL tracing.
- set sql tracing database argument: Change database tracing (SQL administration API)
  Use the set sql tracing database argument with the **admin()** or **task()** function to start or stop tracing for a database, or list which databases are being traced.
- set sql tracing session argument: Control tracing for a session (SQL administration API)
  Use the set sql tracing session argument with the **admin()** or **task()** function to change SQL tracing for the current session.
- set sql tracing user argument: Control tracing for users (SQL administration API)
  Use the set sql tracing user argument with the **admin()** or **task()** function to change SQL tracing for users.
- set sql user tracing argument: Set global SQL tracing for a user session (SQL administration API)
  Use the set sql user tracing argument with the **admin()** or **task()** function to set the mode of global SQL tracing for a specified user session.
- start json listener argument: Start the MongoDB API wire listener (deprecated)
  Use the start json listener argument with the **admin()** or **task()** function to start the MongoDB API wire listener.
- start listen argument: Start a listen thread dynamically (SQL administration API)
  Use the start listen argument with the **admin()** or **task()** function to start an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.
- start mirroring argument: Starts storage space mirroring (SQL administration API)
  Use the start mirroring argument with the **admin()** or **task()** function to start mirroring for a specified dbspace, blobspace, or sbspace.
- statement cache enable argument: Enable the SQL statement cache (SQL administration API)
  Use the statement cache enable argument with the **admin()** or **task()** function to enable the statement cache.
- statement cache flush argument: Flush the SQL statement cache (SQL administration API)
  Use the statement cache flush argument with the **admin()** or **task()** function to flush the statement cache.
- statement cache hits argument: Specify the number of hits in the SQL statement cache (SQL administration API)
  Use the statement cache hits argument with the **admin()** or **task()** function to specify the number of hits in the SQL statement cache.
- statement cache nolimit argument: Control whether to insert qualified statements into the SQL statement cache (SQL administration API)
  Use the statement cache nolimit argument with the **admin()** or **task()** function to control whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.
- statement cache off argument: Disable the SQL statement cache (SQL administration API)
  Use the statement cache off argument with the **admin()** or **task()** function to turn off the SQL statement cache.
- statement cache save argument: Save the SQL statement cache (SQL administration API)
  Use the statement cache save argument with the **admin()** or **task()** function to save the statement cache.
- statement cache restore argument: Restore the SQL statement cache (SQL administration API)
  Use the statement cache restore argument with the **admin()** or **task()** function to restore the statement cache.
- stop json listener: Stop the wire listener (deprecated)
  Use the stop json listener argument with the **admin()** or **task()** function to stop the wire listener.

- stop listen argument: Stop a listen thread dynamically (SQL administration API)
  Use the stop listen argument with the **admin()** or **task()** function to stop an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.
- stop mirroring argument: Stops storage space mirroring (SQL administration API)
  Use the stop mirroring argument with the **admin()** or **task()** function to stop mirroring for a specified dbspace, blobspace, or sbspace.
- storagepool add argument: Add a storage pool entry (SQL administration API)
  Use the storagepool add argument with the **admin()** or **task()** function to add an entry to the *storage pool* (a collection of available raw devices, cooked files, or directories that Informix can use to automatically add space to an existing storage space).
- storagepool delete argument: Delete one storage pool entry (SQL administration API)
  Use the storagepool delete argument with the **admin()** or **task()** function to delete an entry from the storage pool.
- storagepool modify argument: Modify a storage pool entry (SQL administration API)
  Use the storagepool modify argument with the **admin()** or **task()** function to modify an entry for a directory, cooked file, or raw device that Informix can use when additional storage space is required.
- storagepool purge argument: Delete storage pool entries (SQL administration API)
  Use the storagepool purge argument with the **admin()** or **task()** function to delete all storage pool entries, storage pool entries that have a status of `Full`, or storage pool entries that have a status of `Error`.
- Table and fragment pfsc_boost argument: Enable or disable a boosted partition free space cache (PFSC)
  You can enable and disable a boosted partition free space cache for a table or fragment using SQL administration API **admin()** or **task()** functions and arguments.
- Table and fragment compress and uncompress operations (SQL administration API)
  You can compress and uncompress the data in a table or in table fragments with SQL administration API **admin()** or **task()** functions and arguments. Compression operations apply only to the contents of data rows and the images of those data rows that appear in logical log records.
- tenant create argument: Create a tenant database (SQL Administration API)
  Use the **tenant create** argument with the **admin()** or **task()** function to create a tenant database.
- tenant drop argument: Drop a tenant database (SQL Administration API)
  Use the tenant drop argument with the **admin()** or **task()** function to drop a tenant database.
- tenant update argument: Modify tenant database properties (SQL Administration API)
  Use the tenant update argument with the **admin()** or **task()** function to modify the properties of a tenant database.

SQL Administration API Overview

Use the SQL administration API to remotely administer through SQL statements.

The SQL administration API consists of two functions: **admin()** and **task()**. These functions perform the same operations, but return results in different formats. These functions take one or more arguments that define the operation. Many of the operations are ones that you can also complete with command line utilities. The advantage of using the SQL administration API functions is that you can run them remotely from other database servers; whereas you must be directly connected to the database server on which to run command line utility commands.

You can invoke the **admin()** and **task()** functions within SQL statements that can include an expression, or you can use the EXECUTE FUNCTION statement to call them. Run the **admin()** or **task()** function within a transaction that does not include any other statements.
Note: When connected to a secondary node, the **admin()** function is disabled and will always return a value of -1. In addition, the **task()** function will return an error for commands that involve modifying disk structures, since these administrative actions are meant to be executed only on primary or standalone nodes.
The SQL administration API functions are defined in the **sysadmin** database. You must be connected to the **sysadmin** database, either directly or remotely, to run these functions.

The SQL administration API functions can be run only by the following users:

- The user **informix**
- The **root** user, if Connect privilege on the **sysadmin** database is granted to the user
- The DBSA group members, if Connect privilege on the **sysadmin** database is granted to the role
- Users granted privileges to SQL administration API commands by the **admin()** and **task()** functions with **grant admin** argument.

You can generate SQL administration API commands for reproducing the storage spaces, chunks, and logs that exist in a file. To do this, run the **dbschema** utility with the **-c** option.

- admin() and task() Function Syntax Behavior
  The **admin()** and **task()** functions take one or more arguments as quoted strings separated by commas.
- admin() and task() Argument Size Specifications
  By default, the units for arguments specifying sizes in **admin()** and **task()** functions are kilobytes. You can specify other units.
- admin() and task() Function Return Codes
  The **admin()** and **task()** functions perform equivalent tasks but produce different types of return codes. Use the **admin()** function if you want an integer return code, or the **task()** function if you want a textual return code.

Related information:
Storage space, chunk, and log creation

admin() and task() Function Syntax Behavior

The **admin()** and **task()** functions take one or more arguments as quoted strings separated by commas.

The syntax for the **admin()** and **task()** functions includes the following rules:

- Each argument must be delimited by a pair of single (') quotation marks or double (") quotation marks.

- Arguments must be separated by a comma.
- The maximum number of arguments is 28.
- Most arguments are not case-sensitive, with the following exceptions:
  - The argument that immediately follows the initial onmode argument is case-sensitive.
    For example:

    ```
    EXECUTE FUNCTION task("onmode","D","50");
    ```

  - The arguments included with the cdr argument are case-sensitive.
    For example:

    ```
    EXECUTE FUNCTION task("cdr define server",
                          "-c=g_amsterdam","--init g_amsterdam");
    ```

- If you are not directly connected to the **sysadmin** database, you must include the **sysadmin** database name and the server name, according to the standard Database Object Name syntax. For example, if your server name is **ids_server**, you could run the following statement:

  ```
  EXECUTE FUNCTION sysadmin@ids_server:admin("add bufferpool","2",
  "50000","8","60.0","50.0");
  ```

  For more information on the Database Object Name syntax, see the *IBM® Informix® Guide to SQL: Syntax*.

admin() and task() Argument Size Specifications

By default, the units for arguments specifying sizes in **admin()** and **task()** functions are kilobytes. You can specify other units.

You can use the following units in size arguments to the **admin()** and **task()** functions:

Notation	Corresponding Units
B	Bytes
K	Kilobytes (Default)
M	Megabytes
G	Gigabytes
T	Terabytes
P	Petabytes

The letter case of these characters is ignored.

Any white space that separates the size specification and the units abbreviation in the same argument is ignored. For example, the specifications `"128M"` and `"128 m"` are both interpreted as 128 megabytes.

When a size argument is omitted, the default size for that object applies, based either on the setting of a configuration parameter, or on the system default if no parameter is set. Storage spaces, for example, have a default size of 100 megabytes.

admin() and task() Function Return Codes

The **admin()** and **task()** functions perform equivalent tasks but produce different types of return codes. Use the **admin()** function if you want an integer return code, or the **task()** function if you want a textual return code.

When you run the **admin()** or **task()** function, it:

- Performs the specified operation.
- Returns a value that signifies whether the function succeeded or failed.
- Inserts a row into the **command_history** table of the **sysadmin** database.

The return codes for the **admin()** and **task()** functions indicate whether the function succeeded or failed in different formats:

- The **task()** function returns a textual message. The message is also inserted into the **cmd_ret_msg** column in the new row that the **task()** function inserts into the **command_history** table.
- The **admin()** function returns an integer. This number is also inserted into the **cmd_number** column in the new row that the **admin()** function inserts into the **command_history** table.
  - If this value is greater than zero, the function succeeded, and a new row was inserted into the **command_history** table.
  - If this value is zero, the function succeeded, but could not insert a new row into the **command_history** table.
  - If this value is less than zero, the function failed, but a new row was inserted into the **command_history** table.

The operation that the **admin()** or **task()** function specifies occurs in a separate transaction from the insertion of the new row into the **command_history** table. If the command executes successfully, but the insertion into the **command_history** table fails, the command takes effect, but an **online.log** error entry indicates the problem.

If the **command_history.cmd_number** serial counter is 200 when this function is called, and the command succeeds, then Informix® executes the command and returns the integer 201. If the command fails, this example returns the value –201.

Suppose the **task()** function had executed the same command:

```
EXECUTE FUNCTION task("check extents");
```

This command instructs the database server to check the extents, and returns a message indicating whether the command succeeded or failed.
If the **command_history.cmd_number** serial counter is 201 when this function is called, and the command fails, then the returned value is –202. Suppose that the next SQL administration API function that the DBSA invokes is this:

```
EXECUTE FUNCTION admin('create dbspace',
   'dbspace2', '/work/CHUNKS/dbspace2', "20M");
```

If in this case the command succeeds, the returned value is 203. The DBSA can use the following query to examine the two rows of the **command_history** table that these calls to the **admin()** function inserted:

```
SELECT * FROM command_history WHERE cmd_number IN (202,203);
```

This query returns two rows:

```
cmd_number       202
cmd_exec_time    2009-04-17 16:26:14
cmd_user         informix
cmd_hostname     olympia
cmd_executed     create dbspace
cmd_ret_status   -1
cmd_ret_msg      Unable to create file /work/dbspace2

cmd_number       203
cmd_exec_time    2009-04-17 16:26:15
cmd_user         informix
cmd_hostname     olympia
cmd_executed     create dbspace
cmd_ret_status   0
cmd_ret_msg      created dbspace number 2 named dbspace2
```

Note: When connected to a secondary node, the **admin()** function is disabled and will always return a value of -1. In addition, the **task()** function will return an error for commands that involve modifying disk structures, since these administrative actions are meant to be executed only on primary or standalone nodes.

SQL administration API portal: Arguments by privilege groups

You can view a list of **admin()** and **task()** function arguments, which are sorted by privilege groups, with links to information about the arguments.

Privilege groups identify what SQL administration API commands a user can run. Some function arguments are in multiple privilege groups. Privilege groups are granted to users so that they can run the commands that they need for their jobs. By default, only user **informix** or the DBSA can run SQL administration API commands.

Use the **grant admin** argument to grant privileges and the **revoke admin** argument to revoke privileges.

- ADMIN: The user can run all SQL administration API functions.
- BAR privilege group: The user can run backup and restore functions.
- FILE privilege group: The user can manage the message log and display file information.
- GRANT privilege group: The user has the privilege to grant and revoke privileges.
- HA privilege group: The user can run high-availability functions.
- MISC privilege group: The user can administer the database server.
- MONITOR privilege group: The user can run all SQL administration API functions that only display information.
- OPERATOR: The user can run all SQL administration API functions except functions in the GRANT privilege group.
- REPLICATION privilege group: The user can run Enterprise Replication **cdr** utility functions.
- SQL privilege group: The user can run functions that are related to SQL statements for managing databases.
- SQLTRACE privilege group: The user can run SQL tracing functions.
- STORAGE privilege group: The user can run space-related functions.
- TENANT privilege group: The user can run tenant database functions.
- WAREHOUSE: The user can run Informix® Warehouse Accelerator administration tools. See Permissions for administering Informix Warehouse Accelerator.

BAR privilege group

The BAR privilege group includes SQL administration API function arguments to back up your databases.

Table 1. **admin()** and **task()** Function Arguments for backup and restore

Argument	Version
archive fake argument: Perform an unrecorded backup (SQL administration API)	11.50.xC1
ontape archive argument: Backup the data on your database (SQL administration API)	11.70.xC2
onbar argument: Backup the storage spaces (SQL administration API)	11.70.xC2
onsmsync argument: Synchronize with the storage manager catalog (SQL administration API)	11.70.xC2

FILE privilege group

The FILE privilege group includes SQL administration API function arguments to manage message logs and display file information.

Table 2. **admin()** and **task()** Function Arguments for Message Log Commands

Argument	Version
file status argument: Display the status of a message log file (SQL administration API)	11.10.xC3
message log rotate argument: Rotate the message log file (SQL administration API)	11.10.xC3
message log delete argument: Delete a message log file (SQL administration API)	11.10.xC3
message log truncate argument: Delete the contents of a message log file (SQL administration API)	11.10.xC3
print file info argument: Display directory or file information (SQL administration API)	11.70.xC2

GRANT privilege group

The GRANT privilege group includes SQL administration API function arguments to grant or revoke privileges for running SQL administration API commands to other users.

Table 3. **admin()** and **task()** Function Arguments for granting and revoking privileges

Argument	Version
grant admin argument: Grant privileges to run SQL administration API commands	12.10.xC1
revoke admin argument: Revoke privileges to run SQL administration API commands	12.10.xC1

HA privilege group

The HA privilege group includes SQL administration API function arguments to manage high-availability clusters.

Table 4. **admin()** and **task()** Function Arguments for high-availability cluster Commands

Argument	Version
ha make primary argument: Change the mode of a secondary server (SQL administration API)	11.50.xC1
ha rss argument: Create an RS secondary server (SQL administration API)	11.50.xC1
ha rss add argument: Add an RS secondary server to a primary server (SQL administration API)	11.50.xC1
ha rss change argument: Change the password of an RS secondary server (SQL administration API)	11.50.xC1
ha rss delete argument: Delete an RS secondary server (SQL administration API)	11.50.xC1
ha sds clear argument: Stop shared-disk replication (SQL administration API)	11.50.xC1
ha sds primary argument: Convert an SD secondary server to a primary server (SQL administration API)	11.50.xC1
ha sds set argument: Create a shared-disk primary server (SQL administration API)	11.50.xC1
ha set idxauto argument: Replicate indexes to secondary servers (SQL administration API)	11.50.xC1
ha set ipl argument: Log index builds on the primary server (SQL administration API)	11.50.xC1
ha set primary argument: Define an HDR primary server (SQL administration API)	11.50.xC1
ha set secondary argument: Define an HDR secondary server (SQL administration API)	11.50.xC1
ha set standard argument: Convert an HDR server into a standard server (SQL administration API)	11.50.xC1
ha set timeout argument: Change SD secondary server timeout (SQL administration API)	11.50.xC1
onmode and d arguments: Set data-replication types (SQL administration API)	11.50.xC1

MISC privilege group

The MISC privilege group includes SQL administration function arguments to administer the database server:

- onstat
- Configuration parameters
- Data, partitions, and extents
- Listen threads
- Message log
- Memory
- PDQ
- Server mode
- SQL statement cache
- Other administrative tasks

onstat

SQL administration API function arguments to monitor the database server by running **onstat** commands.

Table 5. **admin()** and **task()** Function Arguments for onstat Commands

Argument	Version
onstat argument: Monitor the database server (SQL administration API)	12.10.xC1

Configuration parameters

SQL administration API function arguments to update configuration parameters.

Table 6. **admin()** and **task()** Function Arguments for Configuration Parameter Commands

Argument	Version
export config argument: Export configuration parameter values (SQL administration API)	12.10xC1
import config argument: Import configuration parameter values (SQL administration API)	12.10xC1
modify config arguments: Modify configuration parameters (SQL administration API)	12.10xC1
onmode and wf arguments: Permanently update a configuration parameter (SQL administration API)	11.10.xC1
onmode and wm arguments: Temporarily update a configuration parameter (SQL administration API)	11.10.xC1
onmode, wm, and AUTO_LRU_TUNING arguments: Change LRU tuning status (SQL administration API)	11.10.xC1
reset config argument: Revert configuration parameter value (SQL administration API)	12.10xC1
reset config all argument: Revert all dynamically updatable configuration parameter values (SQL administration API)	12.10xC1
set onconfig memory argument: Temporarily change a configuration parameter (SQL administration API)	11.50.xC3
set onconfig permanent argument: Permanently change a configuration parameter (SQL administration API)	11.50.xC3

Data, partitions, and extents

SQL administration API function arguments to manage data, partitions, and extents.

Table 7. **admin()** and **task()** Function Arguments for Data, Partition, and Extent Commands

Argument	Version
check data argument: Check data consistency (SQL administration API)	11.10.xC1
check extents argument: Check extent consistency (SQL administration API)	11.10.xC1
check partition argument: Check partition consistency (SQL administration API)	11.10.xC1
checkpoint argument: Force a checkpoint (SQL administration API)	11.10.xC1
create dbaccessdemo argument: Create the demonstration database (SQL administration API)	12.10.xC1
onmode and C arguments: Control the B-tree scanner (SQL administration API)	11.10.xC1
onmode and c arguments: Force a checkpoint (SQL administration API)	11.10.xC1
print partition argument: Print partition information (SQL administration API)	11.10.xC1
set dataskip argument: Start or stop skipping a dbspace (SQL administration API)	11.10.xC1
set index compression argument: Change index page compression (SQL administration API)	11.50.xC2

Listen threads

SQL administration API function arguments to control listen threads for a SOCTCP or TLITCP network protocol without interrupting existing connections.

Table 8. **admin()** and **task()** Function Arguments for Listen Thread Commands

Argument	Version
restart listen argument: Stop and start a listen thread dynamically (SQL administration API)	11.50.xC6
start listen argument: Start a listen thread dynamically (SQL administration API)	11.50.xC6
stop listen argument: Stop a listen thread dynamically (SQL administration API)	11.50.xC6

Message log

SQL administration API function arguments to manage message logs.

Table 9. **admin()** and **task()** Function Arguments for Message Log Commands

Argument	Version
file status argument: Display the status of a message log file (SQL administration API)	11.10.xC3
message log rotate argument: Rotate the message log file (SQL administration API)	11.10.xC3
message log delete argument: Delete a message log file (SQL administration API)	11.10.xC3
message log truncate argument: Delete the contents of a message log file (SQL administration API)	11.10.xC3

Memory

SQL administration API function arguments to manage memory.

Table 10. **admin()** and **task()** Function Arguments for Memory Commands

Argument	Version
add bufferpool argument: Add a buffer pool (SQL administration API)	11.10.xC1
add memory argument: Increase shared memory (SQL administration API)	11.10.xC1
onmode and a arguments: Add a shared-memory segment (SQL administration API)	11.10.xC1
onmode and F arguments: Free unused memory segments (SQL administration API)	11.10.xC1
onmode and n arguments: Unlock resident memory (SQL administration API)	11.10.xC1
onmode and r arguments: Force residency of shared memory (SQL administration API)	11.10.xC1
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)	11.70.xC3 11.50xC9
scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)	11.70.xC3 11.50xC9

PDQ

SQL administration API function arguments to manage PDQ.

Table 11. **admin()** and **task()** Function Arguments for PDQ Commands

Argument	Version
onmode and D arguments: Set PDQ priority (SQL administration API)	11.10.xC1
onmode and M arguments: Temporarily change decision-support memory (SQL administration API)	11.10.xC1
onmode and Q arguments: Set maximum number for decision-support queries (SQL administration API)	11.10.xC1
onmode and S arguments: Set maximum number of decision-support scans (SQL administration API)	11.10.xC1

Server mode

SQL administration API function arguments to change the server mode.

Table 12. **admin()** and **task()** Function Arguments for Server Mode Commands

Argument	Version
onmode and j arguments: Switch the database server to administration mode (SQL administration API)	11.10.xC1
onmode and m arguments: Switch to multi-user mode (SQL administration API)	11.10.xC1

SQL statement cache

SQL administration API function arguments to manage the SQL statement cache.

Table 13. **admin()** and **task()** Function Arguments for SQL Statement Cache Commands

Argument	Version
onmode and e arguments: Change usage of the SQL statement cache (SQL administration API)	11.10.xC1
onmode and W arguments: Reset statement cache attributes (SQL administration API)	11.10.xC1

Other administrative tasks

SQL administration API function arguments to manage other administrative tasks.

Table 14. **admin()** and **task()** Function Arguments for other administrative task Commands

Argument	Version
alter logmode argument: Change the database logging mode (SQL administration API)	11.10.xC1
create dbaccessdemo argument: Create the demonstration database (SQL administration API)	12.10.xC1
onmode and e arguments: Change usage of the SQL statement cache (SQL administration API)	11.10.xC1
onmode and l arguments: Switch to the next logical log (SQL administration API)	11.10.xC1
onmode and p arguments: Add or remove virtual processors (SQL administration API)	11.10.xC1
onmode and Y arguments: Change query plan measurements for a session (SQL administration API)	11.10.xC1
onmode and z arguments: Terminate a user session (SQL administration API)	11.10.xC1
onmode and Z arguments: Terminate a distributed transaction (SQL administration API)	11.10.xC1
print error argument: Print an error message (SQL administration API)	11.10.xC1
reset sysadmin argument: Move the sysadmin database (SQL administration API)	11.10.xC1
scheduler argument: Stop or start the scheduler (SQL administration API)	11.10.xC1

MONITOR privilege group

The MONITOR privilege group includes SQL administration function arguments to monitor the message log, Enterprise Replication, and compression estimates.

Table 15. **admin()** and **task()** Function Arguments for monitoring the message log, Enterprise Replication, or compression estimates

Argument	Version
cdr error, **cdr finderr**, **cdr list repair**, **cdr list replicate**, **cdr list replicateset**, **cdr list server**, **cdr list template**, **cdr stats recv**, and **cdr stats rqm** arguments cdr argument: Administer Enterprise Replication (SQL administration API)	12.10.xC1
file status argument: Display the status of a message log file (SQL administration API)	11.10.xC3
index estimate_compression argument: Estimate index compression (SQL administration API)	12.10.xC1
print error argument: Print an error message (SQL administration API)	11.10.xC1
onstat argument: Monitor the database server (SQL administration API)	12.10.xC1
table estimate_compression and **fragment estimate_compression** arguments table or fragment arguments: Compress data and optimize storage (SQL administration API)	11.50.xC4

REPLICATION privilege group

The REPLICATION privilege group includes SQL administration API function arguments to manage Enterprise Replication.

Table 16. **admin()** and **task()** Function Arguments for Enterprise Replication Commands

Argument	Version
cdr argument: Administer Enterprise Replication (SQL administration API)	11.50.xC3

SQL privilege group

The SQL privilege group includes SQL administration API function arguments to create and drop databases and view error messages.

Table 17. **admin()** and **task()** Function arguments for databases and error messages

Argument	Version
create database argument: Create a database (SQL administration API)	11.70.xC2
create dbaccessdemo argument: Create the demonstration database (SQL administration API)	12.10.xC1
drop database argument: Drop a database (SQL administration API)	11.70.xC2
print error argument: Print an error message (SQL administration API)	11.10.xC1

SQLTRACE privilege group

The SQLTRACE privilege group includes SQL administration API function arguments to manage SQL tracing.

Table 18. **admin()** and **task()** Function Arguments for SQL Tracing Commands

Argument	Version
set sql tracing argument: Set global SQL tracing (SQL administration API)	11.10.xC1
set sql tracing database argument: Change database tracing (SQL administration API)	11.50.xC3
set sql tracing session argument: Control tracing for a session (SQL administration API)	11.50.xC3
set sql tracing user argument: Control tracing for users (SQL administration API)	11.10.xC1
set sql user tracing argument: Set global SQL tracing for a user session (SQL administration API)	11.50.xC3

STORAGE privilege group

The STORAGE privilege group includes SQL administration API function arguments for managing the following aspects of storage:

- Storage space encryption argument
- Automatic table storage location arguments
- Compression
- Logical and physical logs
- Mirroring
- Storage spaces
- Storage provisioning

Storage space encryption argument

SQL administration API function argument to change the master encryption key for storage space encryption.

Table 19. **admin()** and **task()** Function Arguments for storage space encryption command

Argument	Version

Argument	Version
master_key reset argument: Change the keystore password (SQL administration API)	12.10.xC8

Automatic table storage location arguments

SQL administration API function arguments to manage the list of dbspaces that store automatically allocated fragments.

Table 20. **admin()** and **task()** Function Arguments for table storage commands

Argument	Version
autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)	12.10.xC3
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)	12.10.xC3
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)	12.10.xC3
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)	12.10.xC3
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)	12.10.xC3

Compression

SQL administration API function arguments to manage the compression of data and to optimize storage.

Table 21. **admin()** and **task()** Function Arguments for Compression Commands

Argument	Version
index compress repack shrink arguments: Optimize the storage of B-tree indexes (SQL administration API)	12.10.xC1
index estimate_compression argument: Estimate index compression (SQL administration API)	12.10.xC1
table or fragment arguments: Compress data and optimize storage (SQL administration API)	11.50.xC4
purge compression dictionary arguments: Remove compression dictionaries (SQL administration API)	11.50.xC4

For an overview of compression and storage optimization commands, see Table and fragment compress and uncompress operations (SQL administration API).

Logical and physical logs

SQL administration API function arguments to manage logical and physical logs.

Table 22. **admin()** and **task()** Function Arguments for Log Commands

Argument	Version
add log argument: Add a new logical log (SQL administration API)	11.10.xC1
alter logmode argument: Change the database logging mode (SQL administration API)	11.10.xC1
alter plog argument: Change the physical log (SQL administration API)	11.10.xC1
drop log argument: Drop a logical log (SQL administration API)	11.10.xC1

Mirroring

SQL administration API function arguments to manage mirroring.

Table 23. **admin()** and **task()** Function Arguments for Mirror Commands

Argument	Version
add mirror argument: Add a mirror chunk (SQL administration API)	11.10.xC1
start mirroring argument: Starts storage space mirroring (SQL administration API)	11.10.xC1
stop mirroring argument: Stops storage space mirroring (SQL administration API)	11.10.xC1

Storage spaces

SQL administration API function arguments to manage chunks, blobspaces, dbspaces, and sbspaces.

Table 24. **admin()** and **task()** Function Arguments for Space Commands

Argument	Version
add chunk argument: Add a new chunk (SQL administration API)	11.10.xC1
alter chunk argument: Change chunk status to online or offline (SQL administration API)	11.10.xC1
clean sbspace argument: Release unreferenced smart large objects (SQL administration API)	11.10.xC1
create blobspace argument: Create a blobspace (SQL administration API)	11.10.xC1
create chunk argument: Create a chunk (SQL administration API)	11.10.xC1
create dbaccessdemo argument: Create the demonstration database (SQL administration API)	12.10.xC1

Argument	Version
create dbspace argument: Create a dbspace (SQL administration API)	11.10.xC1
create sbspace argument: Create an sbspace (SQL administration API)	11.10.xC1
create sbspace with accesstime argument: Create an sbspace that tracks access time (SQL administration API)	11.70.xC4
create sbspace with log argument: Create an sbspace with transaction logging (SQL administration API)	11.70.xC4
create tempdbspace argument: Create a temporary dbspace (SQL administration API)	11.10.xC1
create tempsbspace argument: Create a temporary sbspace (SQL administration API)	11.70.xC4
drop blobspace argument: Drop a blobspace (SQL administration API)	11.10.xC1
drop chunk argument: Drop a chunk (SQL administration API)	11.10.xC1
drop dbspace argument: Drop a dbspace (SQL administration API)	11.10.xC1
drop sbspace argument: Drop an sbspace (SQL administration API)	11.10.xC1
drop tempdbspace argument: Drop a temporary dbspace (SQL administration API)	11.10.xC1
onmode and O arguments: Mark a disabled dbspace as down (SQL administration API)	11.10.xC1
print error argument: Print an error message (SQL administration API)	11.10.xC1
rename space argument: Rename a storage space (SQL administration API)	11.10.xC1
set chunk argument: Change the status of a chunk (SQL administration API)	11.10.xC1
set sbspace accesstime argument: Control access time tracking (SQL administration API)	11.10.xC1
set sbspace avg_lo_size argument: Set the average size of smart large objects (SQL administration API)	11.10.xC1
set sbspace logging argument: Change the logging of an sbspace (SQL administration API)	11.10.xC1

Storage provisioning

SQL administration API function arguments to manage chunks, blobspaces, dbspaces, and sbspaces from storage pools.

Table 25. **admin()** and **task()** Function Arguments for Storage Provisioning Space Commands

Argument	Version
create blobspace from storagepool argument: Create a blobspace from the storage pool (SQL administration API)	11.70.xC1
create chunk from storagepool argument: Create a chunk from the storage pool (SQL administration API)	11.70.xC1
create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API)	11.70.xC1
create plogspace: Create a plogspace (SQL administration API)	12.10.xC3
create sbspace from storagepool argument: Create an sbspace from the storage pool (SQL administration API)	11.70.xC1
create tempdbspace argument: Create a temporary dbspace (SQL administration API)	11.70.xC1
create tempsbspace from storagepool argument: Create a temporary sbspace from the storage pool (SQL administration API)	11.10.xC1
create tempdbspace from storagepool argument: Create a temporary dbspace from the storage pool (SQL administration API)	11.70.xC1
drop blobspace to storagepool argument: Return space from an empty blobspace to the storage pool (SQL administration API)	11.70.xC1
drop chunk to storagepool argument: Return space from an empty chunk to the storage pool (SQL administration API)	11.70.xC1
drop dbspace to storagepool argument: Return space from an empty dbspace to the storage pool (SQL administration API)	11.70.xC1
drop plogspace: Drop the plogspace (SQL administration API)	12.10.xC3
drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API)	11.70.xC1
drop tempdbspace to storagepool argument: Return space from an empty temporary dbspace to the storage pool (SQL administration API)	11.70.xC1
drop tempsbspace to storagepool argument: Return space from an empty temporary sbspace to the storage pool (SQL administration API)	11.70.xC1
modify chunk extend argument: Extend the size of a chunk (SQL administration API)	11.70.xC1
modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API)	11.70.xC1
modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)	11.70.xC1
modify space expand argument: Expand the size of a space (SQL administration API)	11.70.xC1
modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)	11.70.xC1
storagepool add argument: Add a storage pool entry (SQL administration API)	11.70.xC1
storagepool modify argument: Modify a storage pool entry (SQL administration API)	11.70.xC1
storagepool delete argument: Delete one storage pool entry (SQL administration API)	11.70.xC1

Argument	Version
storagepool purge argument: Delete storage pool entries (SQL administration API)	11.70.xC1

TENANT privilege group

The TENANT privilege group includes SQL administration API function arguments to manage tenant databases.

Table 26. **admin()** and **task()** Function Arguments for Tenant Database Commands

Argument	Version
tenant create argument: Create a tenant database (SQL Administration API)	12.10.xC4
tenant drop argument: Drop a tenant database (SQL Administration API)	12.10.xC4
tenant update argument: Modify tenant database properties (SQL Administration API)	12.10.xC4

add bufferpool argument: Add a buffer pool (SQL administration API)

Use the add bufferpool argument with the **admin()** or **task()** function to create a buffer pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"add bufferpool"--,--"page_size"--)--;---------------------><
```

Element	Description	Key Considerations
page_size	The page size in KB.	The page size must be an integral multiple of the default page size, and cannot be greater than 16 KB. On Windows, the page size is always 4 KB.

Usage

Use add bufferpool argument to create a buffer pool for a page size that does not already have a buffer pool. All other characteristics of the buffer pool that you create are set to the values of the fields in the default line of the BUFFERPOOL configuration parameter.

This function is equivalent to the **onparams -b -g** command and the BUFFERPOOL configuration parameter.

Example

The following example adds a buffer pool with a page size of 8 KB:

```
EXECUTE FUNCTION task("add bufferpool","8");
```

Related reference:
onparams -b: Add a buffer pool
BUFFERPOOL configuration parameter

add chunk argument: Add a new chunk (SQL administration API)

Use the add chunk argument with the **admin()** or **task()** function to add a chunk to a dbspace or blobspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"add chunk"--,--"space_name"--,--"path_name"----------------->

>--+-------------------------------------------------------------------------+-->
   '-,--"disk_size"--+-------------------------------------------------------+-'
                     '-,--"offset"--+-------------------------------------+-'
                                    '-,--"mirror_path"--+-----------------+-'
                                                        '-,--"mirror_offset"-'

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
disk_size	The amount of disk space to add in kilobytes.	See admin() and task() Argument Size Specifications.
mirror_offset	The location of the mirror chunk.	
mirror_path	The path to the mirror chunk.	If you are adding a chunk to a mirrored storage space, you must also add a mirror chunk.
offset	The location of the new chunk.	
path_name	The path of the added disk space.	
space_name	The name of the dbspace, blobspace, or sbspace to which you are adding disk space.	

Usage

The size of the chunk must be equal to or greater than 1000 KB and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 TB.

This function is equivalent to the **onspaces -a** command.

Example

The following example adds a 5 MB chunk of raw disk space, at an offset of 5200 kilobytes, to a dbspace named **dbspc3**:

```
EXECUTE FUNCTION task("add chunk", "dbspc3","\\.\e:","5120","5200");
```

The following example adds a 10 MB mirror chunk to a blobspace named **blobsp3** with an offset of 200 kilobytes for both the primary and mirror chunks:

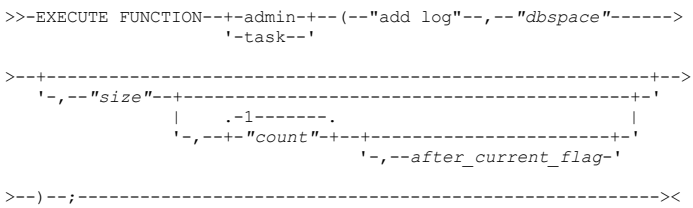
```
EXECUTE FUNCTION task("add chunk","blobsp3","/dev/raw_dev1","10240",
"200","/dev/raw_dev2","200");
```

Related reference:
onspaces -a: Add a chunk to a dbspace or blobspace
onspaces -a: Add a chunk to an sbspace

add log argument: Add a new logical log (SQL administration API)

Use the add log argument with the **admin()** or **task()** function to add a logical log to a dbspace.

Syntax



Element	Description	Key Considerations
after_current_flag	Whether to add the new log after the current log or after the last logical log (default).	Possible values are: • 1 = Add the new log after the current log. • 0 = Add the new log after the last log.
count	The number of log files to create. The default is 1.	The number must not cause the total number of logical-log files to exceed 32,767.
dbspace	The name of the dbspace in which to insert a logical-log file.	You can add a log file to a dbspace only if the database server has adequate contiguous space. You can add a log file during a backup. You cannot add a log file to a blobspace or sbspace.
size	The size in kilobytes of the new logical-log file. The default is the size specified by the LOGSIZE configuration parameter.	This value must be an unsigned integer greater than or equal to 200 KB. Also see admin() and task() Argument Size Specifications.

Usage

The newly added log files have a status of A and are immediately available for use. Use **onstat -l** to view the status of your logical-log files. It is recommended that you take a level-0 backup of the root dbspace and the dbspace that contains the log file as soon as possible after running this function.

By default, the new log file is added after the last logical log. Include `1` as the fifth argument to insert the logical-log file after the current log file.

This function resembles the **onparams -a -d** command, which can add a single logical-log file. You can add multiple logical-log files to the specified dbspace, however, with a single invocation of this function.

Example

The command in the following example adds three logical logs after the current log, each with a size of 5 MB:

```
EXECUTE FUNCTION task ("add log","logdbs","5M",3,1);
```

Related reference:
onparams -a -d dbspace: Add a logical-log file

add memory argument: Increase shared memory (SQL administration API)

Use the add memory argument with the **admin()** or **task()** function to add to the virtual portion of shared memory.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"add memory"--,--"memory_size"--)--;------------------------><
```

Element	Description	Key Considerations
memory_size	The size, in kilobytes, of the new virtual shared-memory segment.	This value must not exceed the operating system limit for the size of shared-memory segments. Also see admin() and task() Argument Size Specifications.

Usage

This size defaults to the SHMADD configuration parameter.

This function is equivalent to the **onmode -a** command.

Example

The following example adds 500 KB of virtual shared-memory:

```
EXECUTE FUNCTION task("add memory","500");
```

Related reference:
onmode -a: Add a shared-memory segment

add mirror argument: Add a mirror chunk (SQL administration API)

Use the add mirror argument with the **admin()** or **task()** function to add a mirror chunk to a dbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"add mirror"--,--"space_name"--,--"path_name"--,------------->

>--"offset"--,--"mirror_path"--,--"mirror_offset"--)--;-------><
```

Element	Description	Key Considerations
mirror_path	The disk partition or unbuffered device of the initial chunk of the dbspace, blobspace, or sbspace that performs the mirroring.	
mirror_offset	The offset to reach the mirrored chunk of the newly mirrored dbspace, blobspace, or sbspace.	See admin() and task() Argument Size Specifications.

Element	Description	Key Considerations
offset	The offset into the disk partition or into the unbuffered device in kilobytes to reach the initial chunk of the newly mirrored dbspace, blobspace, or sbspace.	See admin() and task() Argument Size Specifications.
path_name	The disk partition or unbuffered device of the initial chunk of the dbspace, blobspace, or sbspace that you want to mirror.	
space_name	The name of a dbspace, blobspace, or sbspace to mirror.	

Usage

This function is equivalent to the **onspaces -m** command.

Example

The following example adds a mirror chunk to a blobspace named **blobsp3**:

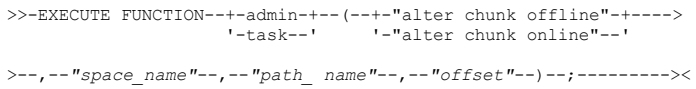
```
EXECUTE FUNCTION task("add mirror","blobsp3","/dev/raw_dev1",
"10240","/dev/raw_dev2","200");
```

Related reference:
onspaces -m: Start mirroring

alter chunk argument: Change chunk status to online or offline (SQL administration API)

Use the alter chunk argument with the **admin()** or **task()** function to bring a chunk online or take a chunk offline in a dbspace, blobspace, or sbspace.

Syntax



Element	Description	Key Considerations
space_name	The name of the blobspace, dbspace, or sbspace.	
path_name	The disk partition or unbuffered device of the chunk.	
offset	The offset (in kilobytes) into the disk partition or unbuffered device to reach the chunk. The default is 0.	See admin() and task() Argument Size Specifications.

Usage

The chunk must be in a mirrored pair, or a non-primary chunk within a noncritical dbspace.

Use the alter chunk online argument to change the chunk status to online.

Use the alter chunk offline argument to change the chunk status to offline.

This function is equivalent to the **onspaces -s** command.

Example

The following example brings a chunk in a space named dbspace4 online:

```
EXECUTE FUNCTION task("alter chunk online","dbspace4","/dev/raw_dev1","0");
```

Related reference:
onspaces -s: Change status of a mirrored chunk

alter logmode argument: Change the database logging mode (SQL administration API)

Use the alter logmode argument with the **admin()** or **task()** function to change the database logging mode to ANSI, buffered, non-logging, or unbuffered.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"alter logmode"--,--"database_name"--,--+-"a"-+--)--;------><
                                           +-"b"-+
                                           +-"n"-+
                                           '-"u"-'
```

Element	Description	Key Considerations
database_name	The name of the database with the logging mode that you want to alter.	

Usage

Unlike when you change the database logging mode with the **ondblog** or **ontape** utilities, when you use this function, the database remains accessible, and a level-0 backup is not always required. Ensure that no other session is active before running this function or it will fail.

Use the "a" argument to change the database logging to be ANSI compliant. After you create or convert a database to ANSI mode, you cannot change it back to any of the other logging modes.

Use the "b" argument to change the database logging to be buffered, so that transaction information is written to a buffer before it is written to a logical log.

Use the "n" argument to change the database logging to be non-logging, so that no database transactions are logged. You must perform a level-0 backup prior to using this argument.

Use the "u" argument to change the database logging to be unbuffered, so that data is not written to a buffer before it is written to a logical log.

Example

The following example changes the logging mode of a database named **employee** to unbuffered logging:

```
EXECUTE FUNCTION task("alter logmode","employee","u");
```

Related concepts:
The onlog utility

alter plog argument: Change the physical log (SQL administration API)

Use the alter plog argument with the **admin()** or **task()** function to change the location and size of the physical log.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"alter plog"--,--"dbspace"--+---------------------+--)--;---><
                               '-,--"phys_log_size"-'
```

Element	Description	Key Considerations
dbspace	The location of the physical log.	The space allocated for the physical log must be contiguous.
phys_log_size	The size, specified in kilobytes, of the physical log.	See admin() and task() Argument Size Specifications.

Usage

To change only the size, specify the current dbspace of the physical log.

This function is equivalent to the **onparams -p** command.

Example

The following example moves the physical log to a dbspace called **phsdbs**:

```
EXECUTE FUNCTION task ("alter plog","physdbs","49 M");
```

Related reference:
onparams -p: Change physical-log parameters

archive fake argument: Perform an unrecorded backup (SQL administration API)

Use the archive fake argument with the **admin()** or **task()** function to perform a backup operation to clone the data in a server without creating a persistent backup that could be used to perform a restore.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"archive fake"--)--;-------><
                     '-task--'
```

Usage

Use this function to populate the secondary server in a High-Availability Data Replication pair.

This function is equivalent to running the **ontape** command with the **-F** option.

Example

The following example starts an unrecorded backup:

```
EXECUTE FUNCTION task("archive fake");
```


autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)

Use the autolocate database add argument with the **admin()** or **task()** function to add a dbspace to the list of available dbspaces for the automatic location and fragmentation of tables for the specified database.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"autolocate database add"--,--"database_name"--,--"dbspace"-->

>--)--;-----------------------------------------------------><
```

Element	Description	Key Considerations
database_name	Name of the database	
dbspace	Name of a dbspace to add to the list of names of the dbspaces in which the database server can automatically create fragments.	The dbspace must exist.

Usage

The AUTOLOCATE configuration parameter or session environment variable must be set to a positive integer.

The list of available dbspaces is stored in the **sysautolocate** system catalog table.

Example

The following command adds the dbspace **dbspace9** to the list of available dbspaces for automatic location and fragmentation for tables in the **customer** database.

```
EXECUTE FUNCTION task("autolocate database add", "customer", "dbspace9");
```

Related reference:
AUTOLOCATE configuration parameter
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)

autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)

Use the autolocate database anywhere argument with the **admin()** or **task()** function to specify that the database server can use any non-critical dbspace for the automatic location and fragmentation of tables for the specified database.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"autolocate database anywhere"--,--"database_name"--)--;---><
```

Element	Description	Key Considerations
database_name	Name of the database	Cannot be the name of a tenant database.

Usage

This command replaces any previous list of dbspaces with a list of all available dbspaces. Dbspaces that are dedicated to tenant database are not available. The list of available dbspaces is stored in the **sysautolocate** system catalog table.

The AUTOLOCATE configuration parameter or session environment variable must be set to a positive integer.

Example

The following command adds all non-critical dbspaces to the list of available dbspaces for automatic location and fragmentation for tables in the **potential_cust** database:

```
EXECUTE FUNCTION task("autolocate database anywhere", "potential_cust");
```

Related reference:
AUTOLOCATE configuration parameter
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)
autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)

autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)

Use the autolocate database argument with the **admin()** or **task()** function to specify the list of available dbspaces for the automatic location and fragmentation of tables for the specified database.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"autolocate database"--,--"database_name"--,--"dbspace_list"-->

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
database_name	Name of the database	Cannot be the name of a tenant database.
dbspace_list	A comma-separated list of names of the dbspaces in which the database server can automatically create fragments.	The dbspaces must exist. The dbspaces cannot be dedicated to a tenant database.

Usage

The AUTOLOCATE configuration parameter or session environment variable must be set to a positive integer.

By default, all dbspaces are available. The list of available dbspaces is stored in the **sysautolocate** system catalog table.

Example

The following command limits the list of available dbspaces for automatic location and fragmentation for tables in the **customer** database:

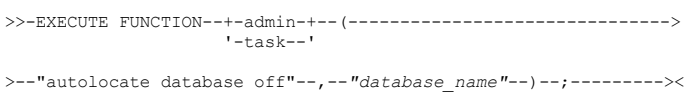
```
EXECUTE FUNCTION task("autolocate database", "customer",
 "dbspace1,dbspace2,dbspace4,dbspace8");
```

Related reference:
AUTOLOCATE configuration parameter
autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)

autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)

Use the autolocate database off argument with the **admin()** or **task()** function to disable the automatic location and fragmentation of tables for a specified database.

Syntax



Element	Description	Key Considerations
database_name	Name of the database	

Usage

New tables that you create in the specified database are stored in the same dbspace as the database and are not fragmented. Existing tables that were automatically fragmented are not allocated new fragments as the table grows.

Example

The following command disables automatic location and fragmentation of tables in the **customer_old** database:

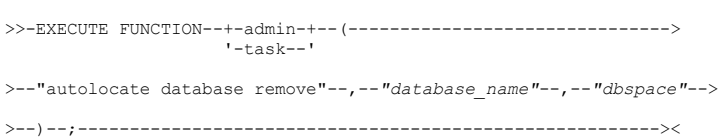
```
EXECUTE FUNCTION task("autolocate database off", "customer_old");
```

Related reference:
AUTOLOCATE configuration parameter
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)
autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)

autolocate database remove argument: Remove a dbspace from the dbspace list (SQL administration API)

Use the autolocate database remove argument with the **admin()** or **task()** function to remove a dbspace from the list of available dbspaces into which the database server can automatically locate and fragment tables for the specified database.

Syntax



Element	Description	Key Considerations
database_name	Name of the database	
dbspace	Name of the dbspace to remove from the list of names of dbspaces in which the database server can automatically create fragments.	The dbspace must exist.

Usage

The AUTOLOCATE configuration parameter or session environment variable must be set to a positive integer.

The list of available dbspaces is stored in the **sysautolocate** system catalog table.

Example

The following command removes **dbspace1** from the list of available dbspaces for the **customer** database.

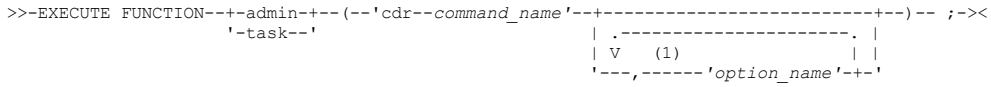
```
EXECUTE FUNCTION task("autolocate database remove", "customer", "dbspace1");
```

Related reference:
AUTOLOCATE configuration parameter
autolocate database argument: Specify dbspaces for automatic location and fragmentation (SQL administration API)
autolocate database add argument: Add a dbspace to the dbspace list (SQL administration API)
autolocate database anywhere argument: Add all dbspaces to the dbspace list (SQL administration API)
autolocate database off argument: Disable automatic fragmentation for a database (SQL administration API)

cdr argument: Administer Enterprise Replication (SQL administration API)

Use the cdr argument with the **admin()** or **task()** function to administer Enterprise Replication.

Syntax



Notes:

1. Maximum of six option arguments.

Element	Description	Key Considerations
command_name	The name of a **cdr** command.	You cannot include any hyphens, flags, or other constraining options to *command_name* that the **cdr** command-line utility requires. You cannot use abbreviations.
option_name	One or more elements of the **cdr** command-line options to the *command_name*.	The elements must be delimited by quotation marks. Also, include (in the correct order) any hyphens, flags, or other elements of **cdr** command-line options that the *command_name* requires. You can use abbreviations.

Usage

Use these functions to produce the same effect as with the **cdr** command-line utility to manage Enterprise Replication.

The SQL administration API supports **cdr** commands used to administer Enterprise Replication. The following commands for monitoring Enterprise Replication are not supported:

- **cdr list grid**
- **cdr list replicate**
- **cdr list replicateset**
- **cdr list server**
- **cdr list template**
- **cdr stats recv**
- **cdr stats rqm**
- **cdr –V**
- **cdr view**

The first argument must include only the **cdr** command names exactly as specified in the appendix for the **cdr** utility in the , such as **cdr define server**. Command names are case-sensitive and abbreviations (such as **cdr sto replset** instead of **cdr stop replicateset**) are **not** supported. The SQL administration API does not perform any validation before passing the parameters to the **cdr** utility.

The second and any following arguments include the command options. The options can be specified in one or up to six arguments.

The following example illustrates the use of the SQL administration API to define an Enterprise Replication server:

```
EXECUTE FUNCTION task ( 'cdr define server', '--connect=g_amsterdam
        --ats=/local0/er/ats --ris=/local0/er/ris --init g_amsterdam' );
```

The following example shows how the options can be spread over several arguments; the above statement can also be written as:

```
EXECUTE FUNCTION task( 'cdr define server',
                      '--connect=g_amsterdam',
                      '--ats=/local0/er/ats',
                      '--ris=/local0/er/ris',
                      '--init g_amsterdam' );
```

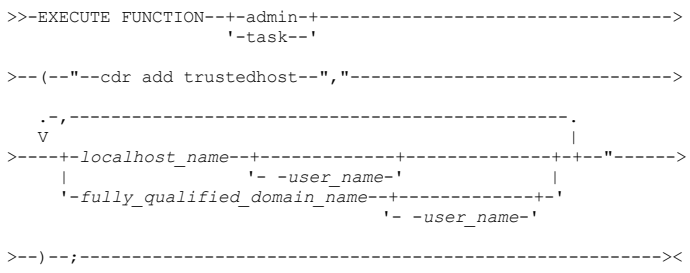
The following example shows double quoted strings within an argument:

```
EXECUTE FUNCTION task('cdr change replicate',
'-d repl_1 -"db1@server1:antonio.table1" "db2@server2:carlo.table2"');
```


cdr add trustedhost argument: Add trusted hosts (SQL administration API)

Use the cdr add trustedhost argument with the **admin()** or **task()** function to add trusted-host relationships for database servers in a high-availability cluster or Enterprise Replication domain. For a database to participate in a high-availability cluster or Enterprise Replication domain, its host must be listed in the trusted-host files of the other high-availability or replication servers.

Syntax



Element	Description	Key Considerations
localhost_name	The localhost name for a database server.	
fully_qualified_domain_name	The full domain name for a database server.	
user_name	A user account with authority over the database-server instance at the specified host.	

Usage

The cdr add trustedhost argument with the **admin()** or **task()** function adds values to the file that is specified by a database server's REMOTE_SERVER_CFG configuration parameter. If a database server is part of a high-availability cluster, trusted-host information also propagates to the trusted-host files of the other cluster servers. The trusted-host values specify the localhost names or fully qualified domain names for the other database servers in a shard cluster. For added security, you can specify user names that are associated with specific hosts.

If the REMOTE_SERVER_CFG configuration parameter is not set, and you run the SQL administration API **task()** or **admin()** function with the **cdr add trustedhost** argument, the database server performs the following actions:

1. The REMOTE_SERVER_CFG configuration parameter is set to authfile.*DBSERVER*.
2. The authfile.*DBSERVER* file is created in $INFORMIXDIR/etc.
3. The specified trusted-host information is added to $INFORMIXDIR/etc/authfile.*DBSERVER*.

If you run cdr add trustedhost argument with the **admin()** or **task()** function on a server in a high-availability cluster, the trusted-host information is added to the trusted-host files of all database servers in the cluster.

You must be a Database Server Administrator (DBSA) to run the **admin()** or **task()** function with the cdr add trustedhost argument.

To see the entries in the trusted host file, run the **admin()** or **task()** function with the cdr list trustedhost argument.

Example 1: Adding trusted host values to a trusted-host file

The following command adds six trusted-host values to the file specified by database server's REMOTE_SERVER_CFG configuration parameter:

```
EXECUTE FUNCTION task("cdr add trustedhost","myhost1, myhost1.ibm.com,
   myhost2, myhost2.ibm.com, myhost3, myhost3.ibm.com");
```

The task specifies localhost names and fully qualified domain names for three database servers.

Example 2: Adding trusted host and trusted user values to a trusted-host file

The following command adds four trusted host and user combinations to the file specified by database server's REMOTE_SERVER_CFG configuration parameter:

```
EXECUTE FUNCTION task("cdr add trustedhost", "myhost1 informix,
    myhost1.ibm.com informix, myhost2 user_1, myhost2.ibm.com user_1");
```

The task specifies localhost names, fully qualified domain names, and user names for two database servers.

Related reference:
REMOTE_SERVER_CFG configuration parameter
Related information:
Enabling sharding for JSON or relational data

cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API)

The cdr autoconfig serv argument with the **admin()** or **task()** function can autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can autoconfigure replication.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--'--cdr autoconfig serv--'---->
                     '-task--'

>--+-------------------------------------------------------------------------+-->
   '-,'--+----------------------+--+-| Source options |----------------------+-'
         '-| Connect Option  |-'  +-| Target options |----------------------+
                                   '-| Source options |--| Target options |-'

>--)--;-------------------------------------------------------><

Connect option

|--+- -c -server-------------+---------------------------------|
   +- --connect=server-------+
   +- -c -server_group-------+
   '- --connect=server_group-'

Source options

|-- --sourcehost--host-- --sourceport--port---------------------|

Target options

|-- --targethost--host-- --targetport--port---------------------|
```

Element	Purpose	Restrictions
host	The name of a database server host.	
port	The port number that is used for communication	
server	Name of the database server to connect to	The name must be the name of a database server or server connection.
server_group	Name of the database server group that includes the database server to connect to	The name must be the name of an existing database server group.

The following table describes the options to **cdr autoconfig serv**.

Long Form	Short Form	Meaning
--sourcehost	**-H**	The host of the database server that is sending autoconfiguration information. If **--sourcehost** and **--sourceport** are not specified, the database server where the function is run is considered the source database server.
--sourceport	**-P**	The port that is used by the database server that is sending autoconfiguration information.
--targethost	**-h**	The host of the database server that is receiving autoconfiguration information.
--targetport	**-p**	The port that is used by the database server that is receiving autoconfiguration information.

Usage

Run the **admin()** or **task()** function with the **cdr autoconfig serv** argument can autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can autoconfigure replication if you are adding database servers to an Enterprise Replication domain. The CDR_AUTO_DISCOVER configuration parameter must be set to `1` on all database servers that are participating in an Enterprise Replication domain, before you can run the **admin()** or **task()** function with the **cdr autoconfig serv** argument. A newly installed database server that is added to an Enterprise Replication domain through the **cdr autoconfig serv** argument must have a configured storage pool.

If the source server is already configured for Enterprise Replication, the function performs the following actions:

1. The source server propagates its trusted-host file to target server.
2. The target server adds entries for itself and all other replication servers to its sqlhosts file.
3. The source server updates its sqlhost file with entries for the target server.
4. Each replication server updates its sqlhost file and trusted-host file with entries for the target server.

5. The target server sets its CDR_DBSPACE configuration parameter and creates the dbspace that is required for Enterprise Replication.
6. The target server sets its CDR_QDATA_SBSPACE configuration parameter and creates the sbspace that is required for Enterprise Replication.
7. The aborted transactions spooling (ATS) file directory $INFORMIXDIR/tmp/ats_*dbservername* is created on the target server.
8. The row information spooling (RIS) file directory $INFORMIXDIR/tmp/ris_*dbservername* is created on the target server.
9. Replication to the target server starts.

If the source server is not configured for Enterprise Replication, the function performs the additional actions:

1. The source server adds entries for itself to its sqlhosts file.
2. The source server sets its CDR_DBSPACE configuration parameter and creates the dbspace that is required for Enterprise Replication.
3. The source server sets its CDR_QDATA_SBSPACE configuration parameter and creates the sbspace that is required for Enterprise Replication.
4. The aborted transactions spooling (ATS) file directory $INFORMIXDIR/tmp/ats_*dbservername* is created on the source server.
5. The row information spooling (RIS) file directory $INFORMIXDIR/tmp/ris_*dbservername* is created on the source server.
6. Replication on the source server begins before replication on the target server begins.

The following restrictions apply to the **admin()** or **task()** function with the **cdr autoconfig serv** argument:

- All replication servers must be active, or the function fails.
- Do not run the **admin()** or **task()** function with the **cdr autoconfig serv** argument if you have configured trusted-host information, manually, rather than through running the **admin()** or **task()** function with the **cdr add trustedhost** argument.
- Do not run the **admin()** or **task()** function with the **cdr autoconfig serv** argument if your replication servers have secure ports that are configured.
- The **admin()** or **task()** function with the **cdr autoconfig serv** argument does not copy hosts.equiv information to the trusted-host file that is set by the REMOTE_SERVER_CFG configuration parameter. Run the **admin()** or **task()** function with the **cdr add trustedhost** argument if you must add information from the hosts.equiv file to the trusted-host file that is set by the REMOTE_SERVER_CFG configuration parameter.

Database servers are configured serially. Parallel configuration is not supported.

You can run this function as a **cdr utility** command.

Example 1: Configure Enterprise Replication on the local server

For this example, you have a local database server that is not configured for Enterprise Replication:

The following task function is run on the local server:

```
EXECUTE FUNCTION task('cdr autoconfig server');
```

The task function configures Enterprise Replication on the local server.

Example 2: Configure connectivity and ER between two stand-alone servers by using source syntax

For this example, you have two database servers:

- **server_1** on **host_1** is configured for Enterprise Replication
- **server_2** on **host_2** is not configured for Enterprise Replication

```
EXECUTE FUNCTION task('cdr autoconfig server', '--connect server_2
   --sourcehost host_1 --sourceport 9020');
```

The task function performs the following actions:

1. The command connects to **server_2**.
2. Enterprise Replication is defined on **server_2**.
3. **server_1** replicates its data to **server_2**

Example 3: Configure connectivity and ER between two stand-alone servers by using target syntax

The following command

```
EXECUTE FUNCTION task('cdr autoconfig server', '--connect server_1
   --targethost host_2 --targetport 9030');
```

The task function performs the following actions:

1. The command connects to **server_1**.
2. Enterprise Replication is defined on **server_2**.
3. **server_1** replicates its data to **server_2**

Related information:
cdr autoconfig serv
CDR_AUTO_DISCOVER configuration parameter

cdr list trustedhost argument: List trusted hosts (SQL administration API)

Use the cdr list trustedhost argument with the **admin()** or **task()** function to list trusted-host information from the file that is specified by the database server's REMOTE_SERVER_CFG configuration parameter.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--("--cdr list trustedhost--");-->
                     '-task--'
```

Usage

You must be a Database Server Administrator (DBSA) to run this function.

Example

The following command lists the trusted-host entries from the database server's trusted-host file:

```
EXECUTE FUNCTION task("cdr list trustedhost");
```

The following example output shows a potential result of using the **cdr list trustedhost** argument.

```
myhost1 user_1
myhost1.example.com user_1
myhost2 user_2
myhost2.example.com user_2
```

Related reference:
REMOTE_SERVER_CFG configuration parameter
Related information:
Enabling sharding for JSON or relational data

cdr remove trustedhost argument: Remove trusted hosts (SQL administration API)

Use the cdr remove trustedhost argument with the **admin()** or **task()** function to remove entries from a database server's trusted-host file.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--("--cdr remove trustedhost--","------------------------------>

   .-,-------------------------------------------.
   V                                             |
>----+-localhost_name---------------------------+-+--"--)--;--->
     +-fully_qualified_domain_name--------------+
     +-localhost_name-- -user_name--------------+
     '-fully_qualified_domain_name-- -user_name-'
```

Element	Description	Key Considerations
localhost_name	The localhost name for a database server.	If you do not specify a *user_name* in the command, all entries that include the specified host name are removed.
fully_qualified_domain_name	The full domain name for a database server.	If you do not specify a *user_name* in the command, all entries that include the specified fully qualified domain name are removed.
user_name	The user account with authority over the database-server instance at the specified host.	

Usage

The cdr remove trustedhost argument removes trusted-host entries from the trusted-host file that is specified by a database server's REMOTE_SERVER_CFG configuration parameter. For a database to participate in a high-availability cluster or Enterprise Replication domain, its host must be listed in the trusted-host files of the other high-availability or replication servers. When you run the **admin()** or **task()** function with the cdr remove trustedhost argument on a server in a high-availability cluster, the trusted-host entries are removed from the trusted host files of all cluster servers.

To see the entries in the trusted host file, run the **admin()** or **task()** function with the cdr list trustedhost argument.

You must be a Database Server Administrator (DBSA) to run the cdr remove trustedhost argument with the **admin()** or **task()** function.

Example 1: Removing host entries from a trusted-host file

The following command removes a localhost name value and a fully qualified domain name value from the trusted-host file that is specified by the database server's REMOTE_SERVER_CFG configuration parameter:

```
EXECUTE FUNCTION task("cdr remove trustedhost","myhost1, myhost1.ibm.com");
```

The **myhost1** and **myhost1.ibm.com** entries from the database server's trusted-host file are removed.

Example 2: Removing host and user entries from a trusted-host file

The following command removes localhost name values, fully qualified domain name values, and user name values from the trusted-host file that is specified by the database server's REMOTE_SERVER_CFG configuration parameter:

```
EXECUTE FUNCTION task("cdr remove trustedhost",
   "myhost2 john,myhost2.ibm.com john,myhost3 informix,myhost3.ibm.com informix");
```

The **myhost2** with user **john**, **myhost2.ibm.com** with user **john**,**myhost3** with user **informix**, and **myhost3.ibm.com** with user **informix** entries from the database server's trusted-host file are removed.

Related reference:
REMOTE_SERVER_CFG configuration parameter
Related information:
Enabling sharding for JSON or relational data

check data argument: Check data consistency (SQL administration API)

Use the check data argument with the **admin()** or **task()** function to check or repair all pages in the specified partition for consistency.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                     '-task--'

>--(--+-"check data"--------+--,--"partition_number"--)--;----->
      +-"check data only"---+
      '-"check data repair"-'
```

Element	Description	Key Considerations
partition_number	The partition number in which to check the data.	Find the partition numbers in the **partnum** column of the **systables** system catalog table.

Usage

Use the **check data** argument to read all pages, except for sbpages, and check each page for consistency. This argument is equivalent to the **oncheck -cD** command.

Use the **check data only** argument to read all pages, except for blobpages and sbpages, and check each page for consistency. This argument is equivalent to the **oncheck -cd** command.

Use the **check data repair** argument to repair inconsistent pages. This argument is equivalent to the **oncheck -cD -y** command.

Example

The following example checks the consistency of all pages in the partition `1048611`:

```
EXECUTE FUNCTION task("check data","1048611");
```

Related reference:
oncheck -cd and oncheck -cD commands: Check pages

check extents argument: Check extent consistency (SQL administration API)

Use the check extents argument with the **admin()** or **task()** function to verify that the extents on disk correspond to the current control information.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                     '-task--'

>--(--"check extents"--+------------------+--)--;------------->
                       '-,--dbspace_number-'
```

Element	Description	Key Considerations
dbspace_number	The number of the dbspace to check.	

Usage

Run this function to check each chunk-free list and corresponding free space and each tblspace extent. If you do not specify a dbspace number, all dbspaces are checked. The function checks dbspaces, blobspaces, smart-large-object extents, and user-data and metadata information in sbspace chunks.

This function is equivalent to the **oncheck -ce** command.

Example

The following example checks the extents in the dbspace with the number **2**:

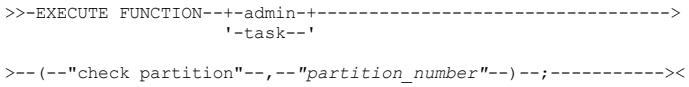
```
EXECUTE FUNCTION task("check extents",2);
```

Related reference:
oncheck -ce, -pe: Check the chunk-free list

check partition argument: Check partition consistency (SQL administration API)

Use the check partition argument with the **admin()** or **task()** function to print tblspace information for a table or fragment.

Syntax



Element	Description	Key Considerations
partition_number	The number of the partition that you want to check for consistency.	Find the partition numbers in the **partnum** column of the **systables** system catalog table.

Usage

The check partition argument with the **task()** function returns information that is equivalent to output of the **oncheck -pt** command. The output contains general information such as the maximum row size, the number of keys, the number and size of extents, the pages allocated and used per extent, the current serial value, and the date that the table was created.

The **admin()** function returns an integer that you can use to find information in the **command_history** table in the **sysadmin** database.

Example

The following example prints information for partition `1048611`:

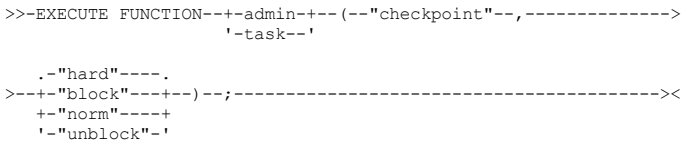
```
EXECUTE FUNCTION task("check partition","1048611");
```

Related reference:
oncheck -pt and -pT: Display tblspaces for a Table or Fragment

checkpoint argument: Force a checkpoint (SQL administration API)

Use the checkpoint argument with the **admin()** or **task()** function to force a checkpoint.

Syntax



Usage

This function forces a checkpoint that flushes the buffers to disk. You can use this function to force a checkpoint if the most recent checkpoint record in the logical log was preventing the logical-log file from being freed (status U-B-L).
Use the block argument to prevent the database server from processing any transactions. Use this option to perform an external backup on Informix®. While the database server is blocked, users cannot access it, except in read-only mode. No transactions can complete until the database server is unblocked.

Use the hard argument to force a blocking checkpoint. This is the default.

Use the norm argument to force a nonblocking checkpoint.

Use the unblock argument to unblock the database server. When the database server is unblocked, data transactions and normal database server operations can resume. Use this option after you complete an external backup on Informix.

This function is equivalent to the **onmode -c** command.

Example

The following example starts a blocking checkpoint:

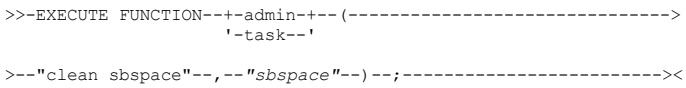
```
EXECUTE FUNCTION task("checkpoint","block");
```

Related reference:
onmode -c: Force a checkpoint

clean sbspace argument: Release unreferenced smart large objects (SQL administration API)

Use the clean sbspace argument with the **admin()** or **task()** function to release any unreferenced BLOB or CLOB objects from the sbspace.

Syntax



Element	Description	Key Considerations
sbspace	The name of the sbspace to clean.	

Usage

This function is equivalent to the **onspaces -cl** command.

Example

The following example cleans an sbspace named **sbsp1**:

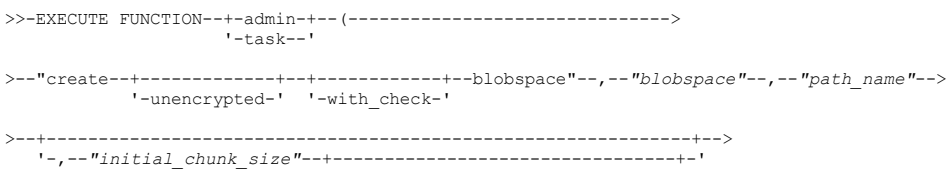
```
EXECUTE FUNCTION task("clean sbspace","sbsp1");
```

Related reference:
onspaces -cl: Clean up stray smart large objects in sbspaces

create blobspace argument: Create a blobspace (SQL administration API)

Use the create blobspace argument with the **admin()** or **task()** function to create a blobspace.

Syntax



```
                    '-,--"offset"--+----------------+-'
                                   '-,--"page_size"-'

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
blobspace	The name of the blobspace to be created.	
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new blobspace.	See admin() and task() Argument Size Specifications.
offset	The offset, in kilobytes, into the disk partition or into the device to reach the initial chunk of the new blobspace.	See admin() and task() Argument Size Specifications.
page_size	The blobspace blobpage size.	You specify the size of a blobpage in multiples of the default Informix® page size for your operating system. For more information, see blobpage size considerations in the *IBM® Informix Performance Guide*.
path_name	The disk partition or device of the initial chunk of the blobspace that you are creating.	

Usage

Use the create with_check blobspace argument to check the specified path name and return an error if it does not exist.

Use the **create unencrypted blobspace** argument to create an unencrypted blobspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -b** command.

Example

The following example creates a blobspace that has a size of 20 MB with an offset of 0 and page size of 2. The blob pages are 2*base page size = 8 K on Windows (4 K base page size).

```
EXECUTE FUNCTION task ("create with_check blobspace","blobs3",
"$INFORMIXDIR/WORK/blobs3","20 M","0","2");
```

Related reference:
onspaces -c -b: Create a blobspace
Avoid overwriting a chunk

create blobspace from storagepool argument: Create a blobspace from the storage pool (SQL administration API)

Use the create blobspace from storagepool argument with the **admin()** or **task()** function to create a blobspace from an entry from the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--create -+--------------+--blobspace from storagepool--"---->
              '-unencrypted-'

>--,--"blobspace"--,--"initial_chunk_size"--,--+-------------------------------------------+-->
                                               '-,--"blobpage_size"--+---------------------+-'
                                                                     '-,--"mirroring_flag"-'

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
blobspace	The name of the blobspace.	The blobspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the `$` character.
blobpage_size	The blobpage size, specified in terms of *page_unit*, the number of disk pages per blobpage	The page size is optional. However if you specify `1` for mirroring, you must also specify a page size.
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new blobspace.	See admin() and task() Argument Size Specifications.

Element	Description	Key Considerations
mirroring_flag	Either: • 1 = mirroring • 0 = no mirroring	The mirroring flag is optional.

Use the **create unencrypted blobspace from storagepool** argument to create an unencrypted blobspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

Examples

The following command creates a mirrored blobspace named `blobspace1`. The new blobspace has a size of 100 gigabytes and a blobpage size of 100 pages.

```
EXECUTE FUNCTION task("create blobspace from storagepool", "blobspace1", "100 GB",
 "100", "1");
```

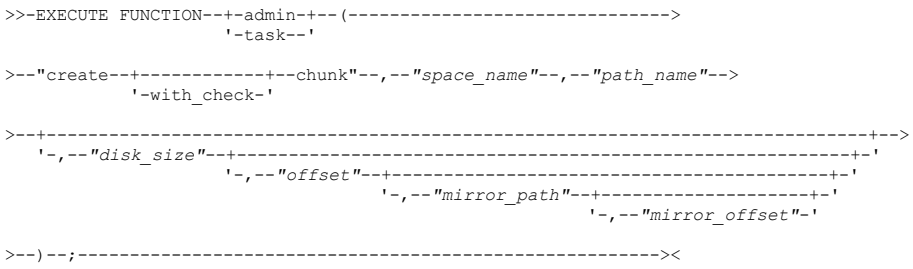
The following command creates an unmirrored blobspace named `blobspace2` with the default blobpage size, so a blobpage size is not specified:

```
EXECUTE FUNCTION task("create blobspace from storagepool", "blobspace2", "5000");
```


create chunk argument: Create a chunk (SQL administration API)

Use the create chunk argument with the **admin()** or **task()** function to create a chunk in a dbspace or in a blobspace.

Syntax



Element	Description	Key Considerations
disk_size	The amount of disk space to add in kilobytes.	See admin() and task() Argument Size Specifications.
mirror_offset	The location of the mirror chunk.	
mirror_path	The path to the mirror chunk. If you are adding a chunk to a mirrored storage space, you must also add a mirror chunk.	
offset	The location of the new chunk.	
path_name	The path of the added disk space.	
space_name	The name of the dbspace, blobspace, or sbspace to which you are adding disk space.	

Usage

Use the create with_check chunk argument to check the specified path name and return an error if it does not exist.

This function is equivalent to the **onspaces -a** command.

Example

The following example adds a 5 MB chunk of raw disk space, at an offset of 5200 kilobytes, to a dbspace named **dbspc3**:

```
EXECUTE FUNCTION task("create chunk", "dbspc3","\\.\e:","5120","5200");
```

The following example adds a 10 MB mirror chunk to a blobspace named **blobsp3** with an offset of 200 kilobytes for both the primary and mirror chunks:

```
EXECUTE FUNCTION task("create with_check chunk","blobsp3","/dev/raw_dev1","10240",
"200","/dev/raw_dev2","200");
```

Related reference:
onspaces -a: Add a chunk to a dbspace or blobspace
Avoid overwriting a chunk
onspaces -a: Add a chunk to an sbspace

create chunk from storagepool argument: Create a chunk from the storage pool (SQL administration API)

Use the create chunk from storagepool argument with the **admin()** or **task()** function to manually create a chunk from an entry in the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--create chunk from storagepool--"-------------------------->

>--,--"space_name"--,--"initial_chunk_size"--)--;-------------><
```

Element	Description	Key Considerations
space_name	The name of the storage space to which you are adding the chunk.	
initial_chunk_size	The size, in kilobytes, of the initial chunk.	See admin() and task() Argument Size Specifications.

Usage

You can also use an SQL administration API command with the modify space expand argument to manually create a chunk from the storage pool and add the chunk to the specified storage space. However, if the space has extendable chunks, Informix® might extend a chunk instead of creating a new one. Unlike the modify space expand argument, the create chunk from storagepool argument forces Informix to add a chunk.

Example

The following command adds a chunk to the dbspace named `logdbs`. The new chunk has a size of 200 megabytes.

```
EXECUTE FUNCTION task("create chunk from storagepool", "logdbs", "200 MB");
```

Related reference:
modify space expand argument: Expand the size of a space (SQL administration API)

create database argument: Create a database (SQL administration API)

Use the create database argument with the **admin()** or **task()** function to create a database.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"create database------------->
                     '-task--'

>--+-------------------------+--+-------------------------+----->
   '-with--+-log-----------+-'  '-+-nlscase insensitive-+-'
           +-buffered log--+      '-nlscase sensitive---'
           '-log mode ansi-'

>--"--,--"database_name"--+--------------------------------+----->
                          '-,--"dbspace"--+-------------+-'
                                          '-,--"locale"-'

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
database_name	The name of the database.	
dbspace	The name of the dbspace to store the data for this database. The default is the root dbspace.	The dbspace must already exist on the database server.
locale	The locale associated with the database.	The values for *locale* are the same as the values for the DB_LOCALE environment variable. If you omit this property, the locale is set by the value of the DB_LOCALE environment variable. The default locale is US English.

Usage

This function is equivalent to the CREATE DATABASE statement.

You cannot use this function to create a tenant database. You create a tenant database with the **tenant create** argument.

Examples

The following example creates the database named `demodbs` with unbuffered logging:

```
EXECUTE FUNCTION task("create database with log","demodbs");
```

The following example creates a database that is not case-sensitive named `demodbs2` with ANSI compliant logging in the dbspace named `dataspace1`:

```
EXECUTE FUNCTION task("create database with log mode ansi nlscase insensitive",
"demodbs2","dataspace1");
```

The following example creates a database named `demodbs3` with a French-Canadian locale in the dbspace name `dataspace1`:

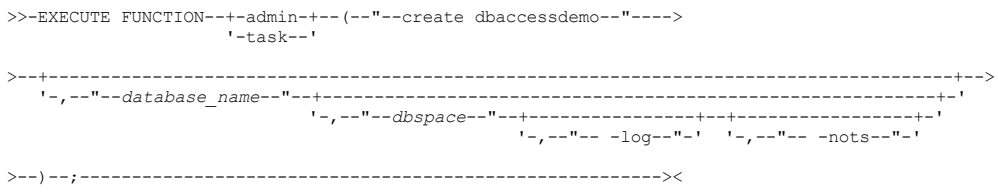
```
EXECUTE FUNCTION task("create database","demodbs3","dataspace1","fr_ca.8859-1");
```

Related reference:
tenant create argument: Create a tenant database (SQL Administration API)
Related information:
CREATE DATABASE statement
DB_LOCALE environment variable

create dbaccessdemo argument: Create the demonstration database (SQL administration API)

Use the create dbaccessdemo argument with the **admin()** or **task()** function to create the **stores_demo** demonstration database.

Syntax



Element	Purpose	Key considerations
database_name	The name of the database to create.	Default database name is **stores_demo**.
dbspace	The name of the dbspace in which to create the database.	Default dbspace is the root dbspace.

Usage

Run this function to create the **stores_demo** database.

Use the **-log** option to enable transaction logging for the demonstration database.

Use the **-nots** option to prevent the creation of the tables with time series data in the demonstration database.

Examples

The following command creates the **stores_demo** database in the root dbspace:

```
EXECUTE FUNCTION task("create dbaccessdemo");
```

The following command creates the demonstration database named **demo2**, in a dbspace named **dbs1**:

```
EXECUTE FUNCTION task("create dbaccessdemo","demo2","dbs1");
```

The following command creates the **stores_demo** database in a dbspace named **dbs1** with transaction logging:

```
EXECUTE FUNCTION task("create dbaccessdemo","stores_demo","dbs1","-log");
```

Related information:

create dbspace argument: Create a dbspace (SQL administration API)

Use the create dbspace argument with the **admin()** or **task()** function to create a dbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"create--+-+-------------+-+--dbspace"--,--"dbspace"--,--"path_name"-->
            | '-unencrypted-' |
            '-+------------+--'
              '-with_check-'

>--+-------------------------------------------------------------------------------------------------+-->
   '-,--"initial_chunk_size"--+-------------------------------------------------------------------+-'
                              '-,--"offset"--+-------------------------------------------------+-'
                                             '-,--"page_size"--+-------------------------------+-'
                                                               '-| Extents and mirror chunk |-'

>--)--;-------------------------------------------------------><

Extents and mirror chunk

|--+-------------------------------------------------------------------------------------------+--|
   '-,--"first_extsize"--+-----------------------------------------------------------------+-'
                         '-,--"next_extsize"--+-----------------------------------------+-'
                                              '-,--"mirror_path"--+-------------------+-'
                                                                  '-,--"mirror_offset"-'
```

Element	Description	Key Considerations
dbspace	The name of the dbspace to be created.	
first_extsize	The size, in KB, of the first extent for the tblspace **tblspace**.	See admin() and task() Argument Size Specifications.
initial_chunk_size	The size, in KB, of the initial chunk of the new dbspace. The size is rounded to a multiple of the page size.	See admin() and task() Argument Size Specifications.
mirror_offset	The offset, in KB, of the mirror chunk.	
mirror_path	The path name to the chunk that mirrors the initial chunk of the dbspace.	
next_extsize	The size, in KB, of the next extents in the tblspace **tblspace**.	See admin() and task() Argument Size Specifications.
offset	The offset, in KB, into the disk partition or into the device to reach the initial chunk of the new dbspace.	
page_size	The non-default page size, in KB, for the new dbspace.	See admin() and task() Argument Size Specifications.
path_name	The disk partition or device of the initial chunk of the dbspace that you are creating.	Valid page sizes depend on the default page size for the computer: • 2 KB default page size: 2, 4, 6, 8, 10, 12, or 16 KB • 4 KB default page size: 4, 8, 12, or 16 KB

Usage

Use the **create with_check dbspace** argument to check the specified path name and return an error if it does not exist.

Use the **create unencrypted dbspace** argument to create an unencrypted dbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -d** command.

Example

The following example creates a dbspace that has a size of 20 MB with an offset of 0.

```
EXECUTE FUNCTION task ("create dbspace", "dbspace3",
"$INFORMIXDIR/WORK/dbspace3", "20 M", "0");
```

Related reference:
onspaces -c -d: Create a dbspace
Avoid overwriting a chunk

DISK_ENCRYPTION configuration parameter

create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API)

Use the create dbspace from storagepool argument with the **admin()** or **task()** function to create a permanent dbspace from an entry in the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                     '-task--'

>--"--create -+-------------+--dbspace from storagepool--"------>
              '-unencrypted-'

>--,--"dbspace"--,--"initial_chunk_size"--,---------------------->

>--+-------------------------------------------------------------------------------------------+-->
   '-,--"page_size"--+-------------------------------------------------------------------------+-'
                     '-,--"mirroring_flag"--+-----------------------------------------+-'
                                            '-,--"first_extent"--+------------------+-'
                                                                 '-,--"next_extent"-'

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
dbspace	Name of the dbspace to be created.	Must be unique among dbspace names, and cannot exceed 128 bytes. It must begin with a letter or underscore, and can include only letters, digits, underscore (_) symbols, or the $ character.
first_extent	Size, in kilobytes, of the first extent for the tblspace **tblspace**.	See admin() and task() Argument Size Specifications.
initial_chunk_size	Size of the initial chunk of the new dbspace.	See admin() and task() Argument Size Specifications.
mirroring_flag	Either: • 1 = mirroring • 0 = no mirroring	The mirroring flag is optional. If none is specified, the default is an unmirrored dbspace.
next_extent	Size, in kilobytes, of the next extents in the tblspace **tblspace**	See admin() and task() Argument Size Specifications.
page_size	Nondefault page size, in kilobytes, for the new dbspace.	The page size is optional. If you specify 1 for mirroring, however, you must also specify a page size. Valid page sizes depend on the default page size for the computer: • 2 KiB default page size: 2, 4, 6, 8, 10, 12, or 16 KiB • 4 KiB default page size: 4, 8, 12, or 16 KiB

Use the **create unencrypted dbspace from storagepool** argument to create an unencrypted dbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

For the **admin()** or **task()** syntax for creating a temporary dbspace, see create tempdbspace argument: Create a temporary dbspace (SQL administration API).

Examples

The following command creates a mirrored dbspace named dbspace3. The new dbspace has a size of 1 gigabyte, a page size of 6 kilobytes, a tblspace **tblspace** first extent size of 200 kilobytes, and a next extent size of 400 kilobytes.

```
EXECUTE FUNCTION task("create dbspace from storagepool",
 "dbspace3",  "1 GB", "6", "1", "200", "400");
```

The following command creates an unmirrored dbspace named dbspace8. The size of the new dbspace is 50 megabytes. Because no page size is specified, the new dbspace has the default page size.

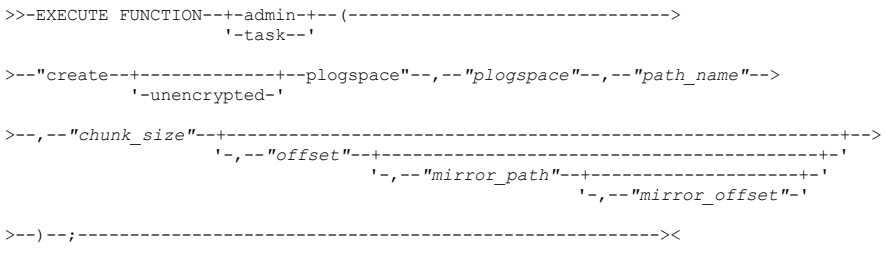
```
EXECUTE FUNCTION task("create dbspace from storagepool",
 "dbspace8", "50000");
```

Related reference:
DISK_ENCRYPTION configuration parameter

create plogspace: Create a plogspace (SQL administration API)

Use the create plogspace argument with the **admin()** or **task()** function to create a plogspace in which to store the physical log.

Syntax



Element	Description	Key Considerations
chunk_size	The size, in KB, of the chunk of the new plogspace. The size is rounded to a multiple of the page size.	See admin() and task() Argument Size Specifications.
mirror_offset	The offset, in KB, of the mirror chunk.	Unsigned integer. The size must be equal to or greater than 1000 KB and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 TB, depending on the platform.
mirror_path	The path name to the chunk that mirrors the chunk of the plogspace.	If you mirror the plogspace, the plogspace chunk cannot be extendable.
offset	The offset, in KB, into the disk partition or into the device to reach the chunk of the new plogspace.	Unsigned integer. The size must be equal to or greater than 1000 KB and a multiple of the page size. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum chunk size is 2 or 4 TB, depending on the platform.
path_name	The disk partition or device of the chunk of the plogspace that you are creating.	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server. UNIX example (unbuffered device): /dev/rdsk/c0t3d0s4 UNIX example (buffered device): /ix/ifmx/db1chunk Windows example: c:\Ifmxdata\ol_icecream\mychunk1.dat
plogspace	The name of the plogspace to be created.	The plogspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character. The syntax must conform to the Identifier segment. For more information, see Identifier.

Usage

Use the **create unencrypted plogspace** argument to create an unencrypted plogspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -P** command.

An instance can have only one plogspace. If a plogspace exists, when you create a new plogspace, the physical log is moved to the new space and the old plogspace is dropped.

The physical log must be stored on a single chunk. The chunk is marked as extendable by default so that the database server can expand the plogspace if necessary to expand the physical log. If you mirror the plogspace, the space cannot expand because a mirror chunk cannot be extendable.

Examples

The following example creates a plogspace that has a size of 30000 KB with an offset of 0.

```
EXECUTE FUNCTION task ("create plogspace", "plogdbs",
"/dev/chk1", 30000, 0);
```

The following example creates a mirrored plogspace that has a size of 30000 KB with an offset of 0.

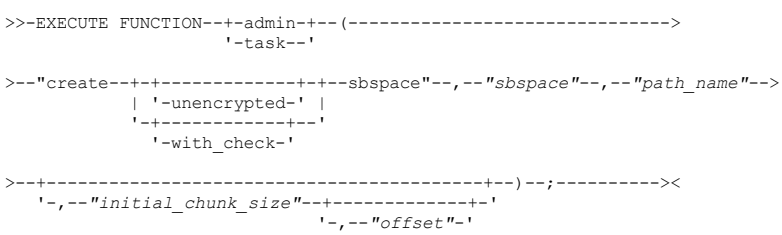
```
EXECUTE FUNCTION task ("create plogspace", "plogdbs",
"/dev/chk1", 30000, 0, "/dev/mchk1", 0);
```

Related reference:
onspaces -c -P: Create a plogspace
DISK_ENCRYPTION configuration parameter
Related information:
Plogspace
Manage the plogspace

create sbspace argument: Create an sbspace (SQL administration API)

Use the create sbspace argument with the **admin()** or **task()** function to create an sbspace.

Syntax



Element	Description	Key Considerations
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new sbspace.	See admin() and task() Argument Size Specifications.
offset	The offset, in kilobytes, into the disk partition or into the device to reach the initial chunk of the new sbspace.	
path_name	The disk partition or unbuffered device of the initial chunk of the sbspace.	
sbspace	The name of the sbspace to be created.	

Usage

Use the **create with_check sbspace** argument to check the specified path name and return an error if it does not exist.

Use the **create unencrypted sbspace** argument to create an unencrypted sbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -S** command.

Example

The following example creates a new sbspace that has a size of 20 MB with an offset of 0:

```
EXECUTE FUNCTION task ("create sbspace","sbspace2",
"$INFORMIXDIR/WORK/sbspace2","20 M","0");
```

Related reference:
onspaces -c -S: Create an sbspace
Avoid overwriting a chunk
create tempsbspace argument: Create a temporary sbspace (SQL administration API)
DISK_ENCRYPTION configuration parameter

create sbspace from storagepool argument: Create an sbspace from the storage pool (SQL administration API)

Use the create sbspace from storagepool argument with the **admin()** or **task()** function to create an sbspace from an entry from the storage pool.

Syntax

```
>--"--create -+--------------+--sbspace from storagepool--"------>
              '-unencrypted-'

>--,--"sbspace"--,--"initial_chunk_size"--+-------------------------------------------+-->
                                          '-,--"logging_flag"--+---------------------+-'
                                                               '-,--"mirroring_flag"-'

>--)--;-----------------------------------------------------><
```

Element	Description	Key Considerations
sbspace	The name of the sbspace.	The sbspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the `$` character.
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new sbspace.	See admin() and task() Argument Size Specifications.
logging_flag	Either: • `1` = logging • `0` = no logging	The logging flag is optional. However if you specify `1` for mirroring, you must also specify a logging flag.
mirroring_flag	Either: • `1` = mirroring • `0` = no mirroring	The mirroring flag is optional.

Use the **create unencrypted sbspace from storagepool** argument to create an unencrypted sbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

Examples

The following command creates a mirrored and logged sbspace named `sbspace1`. The new sbspace has a size of 240 megabytes.

```
EXECUTE FUNCTION task("create sbspace from storagepool", "sbspace1",
 "240 MB", "1", "1");
```

The following command creates an unmirrored and unlogged sbspace named `sbspace2`. This sbspace has a size of 5 gigabytes.

```
EXECUTE FUNCTION task("create sbspace from storagepool", "sbspace2", "5 GB");
```

Related reference:
DISK_ENCRYPTION configuration parameter

create sbspace with accesstime argument: Create an sbspace that tracks access time (SQL administration API)

Use the create sbspace with accesstime argument with the **admin()** or **task()** function to create an sbspace that tracks the time of access for all smart large objects stored in the sbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"create--+-------------+--+-------------+--sbspace with accesstime"--,--"sbspace"--,--"path_name"-->
            '-unencrypted-'  '-with_check-'

>--+-------------------------------------------+--)--;---------><
   '-,--"initial_chunk_size"--+-------------+-'
                              '-,--"offset"-'
```

Element	Description	Key Considerations
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new sbspace.	See admin() and task() Argument Size Specifications.
offset	The offset, in kilobytes, into the disk partition or into the device to reach the initial chunk of the new sbspace.	
path_name	The disk partition or unbuffered device of the initial chunk of the sbspace.	
sbspace	The name of the sbspace to be created.	

Usage

Use the **create with_check sbspace with accesstime** argument to check the specified path name and return an error if it does not exist.

Use the **create unencrypted sbspace with accesstime** argument to create an unencrypted sbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -S** command for creating an sbspace and using the set sbspace accesstime argument with the **admin()** or **task()** function to start tracking the time of access for all smart large objects stored in the sbspace.

Example

The following example creates a new sbspace that tracks access time. This sbspace has a size of 20 MB with an offset of 0:

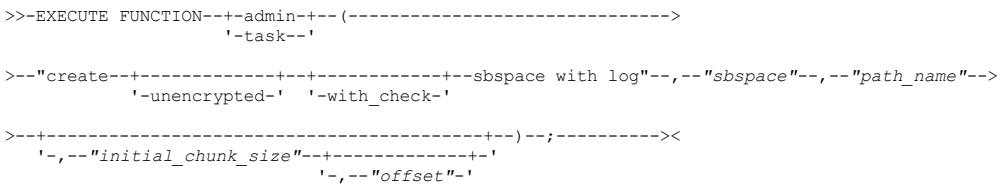
```
EXECUTE FUNCTION task ("create sbspace with accesstime","sbspace4",
"$INFORMIXDIR/WORK/sbspace4","20 M","0");
```

Related reference:
set sbspace accesstime argument: Control access time tracking (SQL administration API)
DISK_ENCRYPTION configuration parameter

create sbspace with log argument: Create an sbspace with transaction logging (SQL administration API)

Use the create sbspace with log argument with the **admin()** or **task()** function to create an sbspace with transaction logging turned on.

Syntax



Element	Description	Key Considerations
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new sbspace.	See admin() and task() Argument Size Specifications.
offset	The offset, in kilobytes, into the disk partition or into the device to reach the initial chunk of the new sbspace.	
path_name	The disk partition or unbuffered device of the initial chunk of the sbspace.	
sbspace	The name of the sbspace to be created.	

Usage

Use the **create with_check sbspace with log** argument to check the specified path name and return an error if it does not exist.

Use the **create unencrypted sbspace with log** argument to create an unencrypted sbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -S** command to create an sbspace with the option for turning logging on.

Example

The following example creates a new sbspace with transaction logging turned on. This sbspace has a size of 20 MB with an offset of 0:

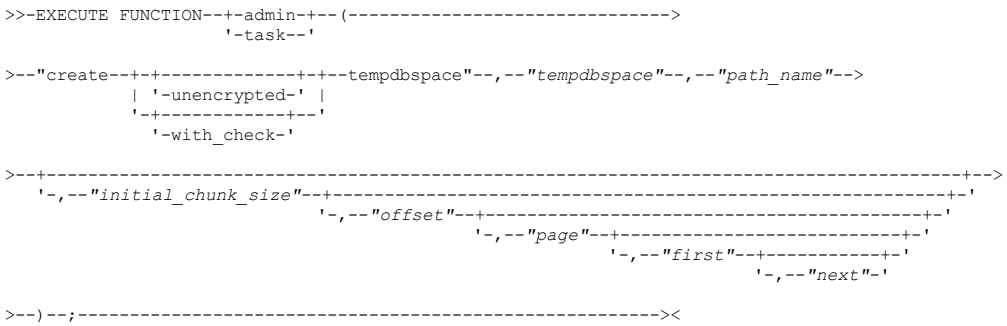
```
EXECUTE FUNCTION task ("create sbspace with log","sbspace2",
"$INFORMIXDIR/WORK/sbspace2","20 M","0");
```

Related reference:
onspaces -c -S: Create an sbspace
DISK_ENCRYPTION configuration parameter
Related information:
Sbspace logging

create tempdbspace argument: Create a temporary dbspace (SQL administration API)

Use the **create tempdbspace** argument with the **admin()** or **task()** function to create a temporary dbspace.

Syntax



Element	Description	Key Considerations
first	Size, in kilobytes, of the first extent for the tblspace **tblspace**.	See admin() and task() Argument Size Specifications.
initial_chunk_size	Size, in kilobytes, of the initial chunk of the new temporary dbspace.	See admin() and task() Argument Size Specifications.
next	Size, in kilobytes, of the next extents in the tblspace **tblspace**.	See admin() and task() Argument Size Specifications.
offset	Offset, in kilobytes, into the disk partition or into the device to reach the initial chunk of the new temporary dbspace.	See admin() and task() Argument Size Specifications
page	Non-default page size, in kilobytes, for the new temporary dbspace.	Valid page sizes depend on the default page size for the computer: • 2 KiB default page size: 2, 4, 6, 8, 10, 12, or 16 KiB • 4 KiB default page size: 4, 8, 12, or 16 KiB
path_name	Path to the disk partition or device of the initial chunk of the temporary dbspace that you are creating.	
tempdbspace	Name of the temporary dbspace to be created.	Cannot exceed 128 bytes. It must begin with a letter or underscore, and can include only letters, digits, underscore (_) symbols, or the $ character.

Usage

Use the **create with_check tempdbspace** argument to check the specified path name and return an error if the path does not exist.

Use the **create unencrypted tempdbspace** argument to create an unencrypted temporary dbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -d -t** command.

Example

The following example creates a temporary dbspace that has a size of 20 MiB with an offset of 0:

```
EXECUTE FUNCTION task("create tempdbspace","tempdbspace3",
"$INFORMIXDIR/WORK/tempdbspace3","20 M","0");
```

For the **admin()** or **task()** syntax to create a permanent dbspace from the storage pool, see create dbspace from storagepool argument: Create a dbspace from the storage pool (SQL administration API).

Related reference:
onspaces -c -d: Create a dbspace
DISK_ENCRYPTION configuration parameter

create tempdbspace from storagepool argument: Create a temporary dbspace from the storage pool (SQL administration API)

Use the create tempdbspace from storagepool argument with the **admin()** or **task()** function to create a temporary dbspace from an entry from the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--create--+-------------+--tempdbspace from storagepool--"-->
              '-unencrypted-'

>--,--"tempdbspace"--,--"initial_chunk_size"--,--+-------------+-->
                                                 '-"page_size"-'

>--)--;------------------------------------------------------><
```

Element	Description	Key Considerations
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new temporary dbspace.	See admin() and task() Argument Size Specifications.
page_size	The non-default page size, in kilobytes, for the new temporary dbspace.	The page size is optional.
tempdbspace	The name of the temporary dbspace.	

Use the **create unencrypted tempdbspace from storagepool** argument to create an unencrypted temporary dbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

Example

The following command creates a temporary dbspace named `tempdbspace1`. The new dbspace has a size of 1 gigabyte and a page size of 12 kilobytes.

```
EXECUTE FUNCTION task("create tempdbspace from storagepool", "tempdbspace1",
 "1 GB", "12");
```

Related reference:
DISK_ENCRYPTION configuration parameter

create tempsbspace argument: Create a temporary sbspace (SQL administration API)

Use the create sbspace argument with the **admin()** or **task()** function to create an sbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"create--+-+-------------+-+--tempsbspace"--,--"tempsbspace"--,--"path_name"-->
            | '-unencrypted-' |
            '-+-------------+--'
              '-with_check-'

>--+-------------------------------------------+--)--;---------><
   '-,--"initial_chunk_size"--+-------------+-'
                              '-,--"offset"-'
```

Element	Description	Key Considerations
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new temporary sbspace.	See admin() and task() Argument Size Specifications.
offset	The offset, in kilobytes, into the disk partition or into the device to reach the initial chunk of the new temporary sbspace.	
path_name	The disk partition or unbuffered device of the initial chunk of the temporary sbspace.	
tempsbspace	The name of the temporary sbspace to be created.	

Usage

Use the **create with_check tempsbspace** argument to check the specified path name and return an error if it does not exist.

Use the **create unencrypted tempsbspace** argument to create an unencrypted temporary sbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

This function is equivalent to the **onspaces -c -S** command with the **-t** option for creating a temporary sbspace.

Example

The following example creates a temporary sbspace that has a size of 20 MB with an offset of 0:

```
EXECUTE FUNCTION task ("create tempsbspace","tempsbspace3",
"$INFORMIXDIR/WORK/tempsbspace3","20 M","0");
```

Related reference:
create sbspace argument: Create an sbspace (SQL administration API)
onspaces -c -S: Create an sbspace
DISK_ENCRYPTION configuration parameter
Related information:
Temporary sbspaces

create tempsbspace from storagepool argument: Create a temporary sbspace from the storage pool (SQL administration API)

Use the create tempsbspace from storagepool argument with the **admin()** or **task()** function to create a temporary sbspace from an entry from the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--create--+-------------+--tempsbspace from storagepool--"-->
              '-unencrypted-'

>--,--"tempsbspace"--,--"initial_chunk_size"--)--;------------><
```

Element	Description	Key Considerations
initial_chunk_size	The size, in kilobytes, of the initial chunk of the new sbspace.	See admin() and task() Argument Size Specifications.
tempsbspace	The name of the temporary sbspace.	The temporary sbspace name must be unique and cannot exceed 128 bytes. It must begin with a letter or underscore and must contain only letters, numbers, underscores, or the $ character.

Use the **create unencrypted tempsbspace from storagepool** argument to create an unencrypted temporary sbspace when encryption is enabled by the DISK_ENCRYPTION configuration parameter.

Example

The following command creates a temporary sbspace named `tempsbspace5`. The temporary sbspace has a size of 240 megabytes.

```
EXECUTE FUNCTION task("create tempsbspace from storagepool",
 "tempsbspace5", "240 MB");
```

Related reference:
DISK_ENCRYPTION configuration parameter

defragment argument: Dynamically defragment partition extents (SQL administration API)

Use the defragment argument with the **admin()** or **task()** function to defragment tables or indexes to merge non-contiguous extents.

Defragmenting a table brings data rows closer together to avoid partition header page overflow problems, and can improve performance.

Before you defragment a partition you should review the Partition defragmentation.

Syntax

You can specify either the `defragment` argument or `defragment partnum` argument using the following syntax:

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"--defragment--"--,----"--database--:--owner--.--table--"----)--;-><
```

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                    '-task--'

                                    .-,------------------.
                                    V                    |
>--(--"--defragment partnum--"--,----partition_number--"-+--)--;-><
```

Element	Description	Key considerations
database	Name of the database that includes the table or index that you want to defragment.	
owner	User ID of the owner of the table.	
table	Name of the table to defragment.	
partition_number	One or more partition numbers to defragment.	Use a comma-separated list of partition numbers to specify more than one partition.

Usage

Use the `defragment` argument to defragment specific tables. Use the `defragment partnum` argument to defragment one or more specific disk partitions.

Information about defragmentation is stored in shared memory. Use the oncheck -pt and -pT: Display tblspaces for a Table or Fragment command to display information about the number of extents for a specific table or fragment. Use the onstat -g defragment command: Print defragment partition extents.

If the defragment request reduces the number of extents by at least 1 extent, the request returns 0 (success), even if there are many extents in the partition.

If a partition has a single extent, the defragment request returns 0 to indicate that the request was a success, even though no extents were merged.

Examples

To defragment the customer table in the `stores_demo` database, use either of the following functions:

```
EXECUTE FUNCTION task("defragment","stores_demo:informix.customer");
```

```
EXECUTE FUNCTION admin("defragment","stores_demo:informix.customer");
```

To defragment an index, you must specify the partition number for the index, as in these two function examples:

```
EXECUTE FUNCTION task("defragment partnum","2097154");
```

```
EXECUTE FUNCTION admin("defragment partnum","2097154");
```

To defragment a list of partitions, use either of the following functions:

```
EXECUTE FUNCTION task("defragment partnum", "16777217,28477346");
```

```
EXECUTE FUNCTION admin("defragment partnum", "16777217,28477346");
```


drop blobspace argument: Drop a blobspace (SQL administration API)

Use the drop blobspace argument with the **admin()** or **task()** function to drop the specified blobspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"drop blobspace"------------->
                    '-task--'

>--,--"blobspace"--)--;----------------------------------------><
```

Element	Description	Key Considerations
blobspace	The name of the blobspace to drop.	Must be an existing blobspace. Before you drop a blobspace, drop all tables that include a TEXT or BYTE column that references the blobspace.

Usage

This function is equivalent to the **onspaces -d** command.

Example

The following example drops the blobspace named **blobspace3**:

```
EXECUTE FUNCTION task("drop blobspace","blobspace3");
```

Related reference:
onspaces -d: Drop a space

drop blobspace to storagepool argument: Return space from an empty blobspace to the storage pool (SQL administration API)

Use the drop blobspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty blobspace to the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--drop blobspace to storagepool--"--,--"blobspace"--)--;--->
```

Element	Description	Key Considerations
blobspace	The name of the empty blobspace.	

Example

The following command drops an empty blobspace named `blob2` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop blobspace to storagepool", "blob2");
```


drop chunk argument: Drop a chunk (SQL administration API)

Use the drop chunk argument with the **admin()** or **task()** function to drop the specified chunk from a dbspace, blobspace, or sbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"drop chunk"----------------->
                     '-task--'

>--,--"space_name"--+-----------------------------------+--)--;-->
                    '-,--"path_name"--,--+----------+-'
                                         '-"offset"-'
```

Element	Description	Key Considerations
offset	The offset, in kilobytes, into the disk partition or into the unbuffered device to reach the initial chunk of the dbspace, blobspace, or sbspace that you are dropping.	The starting offset, an unsigned integer, must be equal to or greater than 0. The starting offset plus the chunk size cannot exceed the maximum chunk size. The maximum offset is 4 TB. Also see admin() and task() Argument Size Specifications.
path_name	The disk partition or unbuffered device of the initial chunk of the dbspace, blobspace, or sbspace that you are dropping.	The chunk must be an existing unbuffered device or buffered file. When you specify a path name, you can use either a full path name or a relative path name. However, if you use a relative path name, it must be relative to the directory that was the current directory when you initialized the database server.
space_name	The name of the dbspace, sbspace, or blobspace from which to drop a chunk.	You can drop a chunk from a dbspace, temporary dbspace, or sbspace when the database server is online or quiescent. You can drop a chunk from a blobspace only when the database server is in quiescent mode.

Usage

This function is equivalent to the **onspaces -d** command.

Example

The following example drops a chunk at an offset of 5200 kilobytes from a dbspace named **dbspc3**:

```
EXECUTE FUNCTION task("drop chunk", "dbspc3","\\.\e:","5200");
```

Related reference:
onspaces -d: Drop a chunk in a dbspace, blobspace, or sbspace

drop chunk to storagepool argument: Return space from an empty chunk to the storage pool (SQL administration API)

Use the drop chunk to storagepool argument with the **admin()** or **task()** function to return the space from an empty chunk to the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--drop chunk to storagepool--"----------------------------->

>--,--"space_name"--,--"path_name"--,--"offset"--)--;---------><
```

Element	Description	Key Considerations
space_name	The name of the storage space in which the chunk resides.	
path_name	The path of the chunk.	
offset	The offset, in kilobytes, of the chunk.	

Example

The following command drops an empty chunk in a dbspace named `bigdbs` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop chunk to storagepool", "bigdbs", "/dev/rawdisk23",
 "100 KB");
```


drop database argument: Drop a database (SQL administration API)

Use the drop database argument with the **admin()** or **task()** function to drop a database.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"drop database"-------------->
                     '-task--'

>--,--"database_name"--)--;------------------------------------><
```

Element	Description	Key Considerations
database_name	The name of the database.	

Usage

This function is equivalent to the DROP DATABASE statement. This function deletes the entire database, including all of the system catalog tables, objects, and data.

Example

The following example drops the database named `demodbs`:

```
EXECUTE FUNCTION task("drop database","demodbs");
```

Related information:
DROP DATABASE statement

drop dbspace argument: Drop a dbspace (SQL administration API)

Use the drop dbspace argument with the **admin()** or **task()** function to drop the specified dbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"drop dbspace"--------------->
                    '-task--'

>--,--"dbspace"--)--;---------------------------------------><
```

Element	Description	Key Considerations
dbspace	The name of the dbspace to drop.	The dbspace must exist. Before you drop a dbspace, drop all databases and tables that you previously created in the dbspace.

Usage

This function is equivalent to the **onspaces -d** command.

Example

The following example drops the dbspace named **dbspace4**:

```
EXECUTE FUNCTION task("drop dbspace","dbspace4");
```

Related reference:
onspaces -d: Drop a space

drop dbspace to storagepool argument: Return space from an empty dbspace to the storage pool (SQL administration API)

Use the drop dbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty dbspace to the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------->
                    '-task--'

>--"--drop dbspace to storagepool--"--,--"dbspace"--)--;------->
```

Element	Description	Key Considerations
dbspace	The name of the empty dbspace.	

Example

The following command drops an empty dbspace named `dbs5` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop dbspace to storagepool", "dbs5");
```


drop log argument: Drop a logical log (SQL administration API)

Use the drop log argument with the **admin()** or **task()** function to drop the specified logical log.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"drop log"------------------->
                    '-task--'

>--,--"log_number"--)--;---------------------------------------><
```

Element	Description	Key Considerations
log_number	The logical log file number.	The number must be an unsigned integer greater than or equal to 0.

Usage

Use this function to drop a single logical log file.

The database server requires a minimum of three logical-log files at all times. You cannot drop a log file if the database server has only three logical-log files.

Important: Before you can drop any of the first three logical-log files, you must add new logical-log files and run a backup of the logical-log files. The backup must be run using either the **ontape -a** command or the **ontape -c** command. After you add the new logical-log files and run a backup, you can then use **onparams -d -l***lognum* to delete the first three logical-log files.
The status of the log file determines if the log file can be dropped, and the actions taken by the database server when the log file is dropped:

- If you drop a log file that has never been written to, status is newly Added (**A**), the database server deletes the log file and frees the space immediately.
- If you drop a used log file that has a status of User (**U**) or Free (**F**), the database server marks the log file as Deleted (**D**). After you take a level-0 backup of the dbspaces that contain the log files and the root dbspace, the database server deletes the log file and frees the space.
- You cannot drop a log file that is currently in use (**C**) or contains the last checkpoint record (**L**).

You can obtain the log number from the number field of the **onstat -l** command. The sequence of log numbers might be out of order.

This function is equivalent to the **onparams -d -l** *lognum* command.

Example

The following example drops the logical log with a file number of 2:

```
EXECUTE FUNCTION task("drop log","2");
```

The following example drops the log for a specific chunk by looking up the log number based on the chunk number:

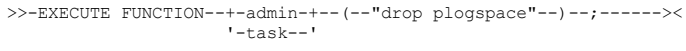
```
SELECT task("drop log", number) FROM sysmaster:syslogfil WHERE chunk = 1;
```

Related reference:
onparams -d -l lognum: Drop a logical-log file

drop plogspace: Drop the plogspace (SQL administration API)

Use the drop plogspace argument with the **admin()** or **task()** function to drop the plogspace.

Syntax



Usage

The plogspace must be empty to be dropped. For example, if you move the physical log out of the plogspace and into a dbspace by running the **onparams -p** command, you can drop the plogspace. Alternatively, you can move the plogspace to a different chunk by creating a new plogspace. The old plogspace is removed automatically.

This function is equivalent to the **onspaces -d** command.

Example

The following example drops the plogspace:

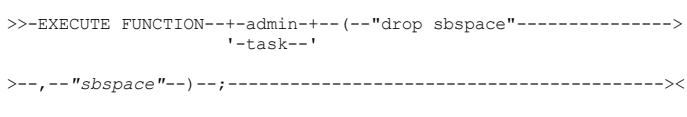
```
EXECUTE FUNCTION task("drop plogspace");
```

Related reference:
onspaces -d: Drop a chunk in a dbspace, blobspace, or sbspace

drop sbspace argument: Drop an sbspace (SQL administration API)

Use the drop sbspace argument with the **admin()** or **task()** function to drop the specified sbspace.

Syntax



Element	Description	Key Considerations

Element	Description	Key Considerations
sbspace	The name of the sbspace to drop.	The sbspace must exist. Before you drop an sbspace, drop all tables that include a BLOB or CLOB column that references the sbspace.

Usage

This function is equivalent to the **onspaces -d** command.

Example

The following example drops the sbspace named **sbspace3**:

```
EXECUTE FUNCTION task("drop dbspace","sbspace3");
```

Related reference:
onspaces -d: Drop a space

drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API)

Use the drop sbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty sbspace to the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--,--"--drop sbspace to storagepool--"--"sbspace"--)--;------><
```

Element	Description	Key Considerations
sbspace	The name of the empty sbspace.	

Example

The following command drops an empty sbspace named `sbspace8` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop sbspace to storagepool", "sbspace8");
```


drop tempdbspace argument: Drop a temporary dbspace (SQL administration API)

Use the drop tempdbspace argument with the **admin()** or **task()** function to drop the specified temporary dbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"drop tempdbspace"----------->
                     '-task--'

>--,--"tempdbspace"--)--;-------------------------------------><
```

Element	Description	Key Considerations
tempdbspace	The name of the temporary dbspace to drop.	The temporary dbspace must exist. Before you drop a temporary dbspace, drop all databases and tables that you previously created in the temporary dbspace.

Usage

This function is equivalent to the **onspaces -d** command.

Example

The following example drops the temporary dbspace named **tdbspace2**:

```
EXECUTE FUNCTION task("drop tempdbspace","tdbspace2");
```

Related reference:
onspaces -d: Drop a space

drop tempdbspace to storagepool argument: Return space from an empty temporary dbspace to the storage pool (SQL administration API)

Use the drop tempdbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty temporary dbspace to the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--drop tempdbspace to storagepool--"--,--"tempdbspace"------>

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
tempdbspace	The name of the empty temporary dbspace.	

Example

The following command drops an empty temporary dbspace named `tempdbs1` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop tempdbspace to storagepool", "tempdbs1");
```


drop tempsbspace to storagepool argument: Return space from an empty temporary sbspace to the storage pool (SQL administration API)

Use the drop tempsbspace to storagepool argument with the **admin()** or **task()** function to return the space from an empty temporary sbspace to the storage pool.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--drop tempsbspace to storagepool--"--,--"tempsbspace"------>

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
tempsbspace	The name of the empty temporary sbspace.	

Example

The following command drops an empty temporary sbspace named `tempsbspace3` and adds all of the freed space to the storage pool.

```
EXECUTE FUNCTION task("drop tempsbspace to storagepool", "tempsbspace3");
```


export config argument: Export configuration parameter values (SQL administration API)

Use the export config argument with the **admin()** or **task()** function to export a file that contains all configuration parameters and their current values.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--export config--"---------->
                     '-task--'

>--,--"file_path"--)--;-----------------------------------------><
```

Table 1. **export config** command elements

Element	Description	Key Considerations
file_path	Full path name for the file	Do not add an extension.

Usage

The SQL administration API export command automatically creates an ASCII file, assigning it the name that you specified in the command. The format of the file is the same as the format of the onconfig.std file.

You must specify the full path name. You cannot specify a relative path.

This command is the equivalent of the **onmode –we** commend.

Example

The following command exports all configuration parameters and their current values to a file named `cfg_12` in the `/tmp` directory:

```
EXECUTE FUNCTION task("export config", "/tmp/cfg_12");
```

Related tasks:
Modifying the onconfig file
Related reference:
import config argument: Import configuration parameter values (SQL administration API)
onmode -we: Export a file that contains current configuration parameters

file status argument: Display the status of a message log file (SQL administration API)

Use the file status argument with the **admin()** or **task()** function to specify the status of an online, ON-Bar activity, or ON-Bar debug message log file.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"file status"--,--"--file_path--"--)--;------------------><
```

Element	Purpose	Key considerations
file_path	Full path name for the online, ON-Bar activity, or ON-Bar debug message log file.	

Example

The following example shows the argument that you can use to display the status of the /usr/informix/online.log file:

```
execute function task("file status", "/usr/informix/online.log");
```

The server then displays information such as:

```
(expression)  File name          = /tmp/x
              Is File            = 1
              Is Directory       = 0
              Is Raw Device      = 0
              Is Block Device    = 0
              Is Pipe            = 0
              File Size          = 554
              Last Access Time   = 11/29/2010 21:55:02
              Last Modified Time = 11/29/2010 21:51:45
              Status Change Time = 11/29/2010 21:51:45
              User Id            = 200
              Group id           = 102
              File Flags         = 33206
```

Related reference:
message log rotate argument: Rotate the message log file (SQL administration API)
message log truncate argument: Delete the contents of a message log file (SQL administration API)
message log delete argument: Delete a message log file (SQL administration API)

grant admin argument: Grant privileges to run SQL administration API commands

Use the grant admin argument with the **admin()** or **task()** function to grant privileges to run SQL administration API commands.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--grant admin--"------------>
                    '-task--'

>--,--"--user_name--"--,--"--privilege_group--"--)--;---------><
```

Element	Description	Key Considerations
user_name	The name of the user for which privileges are granted.	
privilege_group	The name of the privilege group.	See SQL administration API portal: Arguments by privilege groups for a list of privilege groups.

Usage

Individual users can be granted privileges to administer their database servers by running SQL administration API commands. Users with these privileges can connect to a database server with their user name and run SQL administration API commands, by connecting directly.

Only user **informix**, or a user with ADMIN or GRANT privilege for SQL administration API commands, can use the **grant admin** argument.

Example

The following command grants the privilege for running backup and restore SQL administration commands to the user Bob:

```
EXECUTE FUNCTION task("grant admin", "Bob", "BAR");
```

Related reference:
DBCREATE_PERMISSION configuration parameter

ha make primary argument: Change the mode of a secondary server (SQL administration API)

Use the ha make primary argument with the **admin()** or **task()** function to change the specified secondary server to a primary or standard server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                    '-task--'

>--+-"ha make primary"-------+--,--"database_server"--)--;----->
   '-"ha make primary force"-'
```

Element	Description	Key Considerations
database_server	The name of the database server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

This function has different results depending on the type of secondary server:

- HDR Secondary: The current primary server is shut down and the HDR secondary server is made the primary server.
- RS secondary: The RS secondary server is changed to a standard server.
- SD secondary: The SD secondary server is made the new primary server.

Use the ha make primary argument to change an inactive secondary server to a primary server when there is an active connection between them.

Use the ha make primary force argument to change an inactive secondary server to a primary server, whether or not a secondary server is connected to it. If the connection is active, the function succeeds, however, if you run the function with the force argument on an SD secondary server, the shared disk subsystem can become corrupted.

This function is equivalent to the **onmode -d make primary** command.

Example

The following example converts an HDR secondary server named **ids_stores2** into a primary server:

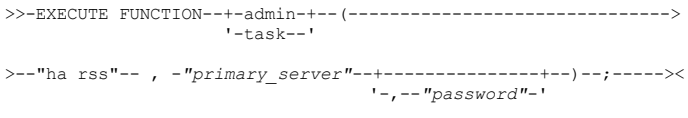
```
EXECUTE FUNCTION task("ha make primary","ids_stores2");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha rss argument: Create an RS secondary server (SQL administration API)

Use the ha rss argument with the **admin()** or **task()** function to create a remote standalone (RS) secondary server.

Syntax



Element	Description	Key Considerations
password	A password to set or to change.	The password is used only during the first connection attempt. After the primary and secondary server have connected, the password cannot be changed.
primary_server	The name of the primary database server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on a standard server or a quiescent HDR secondary server to convert it an RS secondary server.

This function is equivalent to the **onmode -d RSS** command.

Example

The following example converts a standard server into an RS secondary server with a primary server named **ids_stores**:

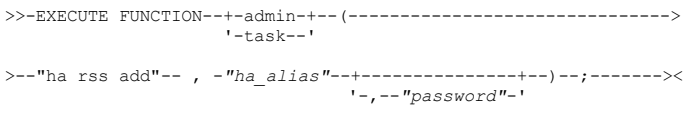
```
EXECUTE FUNCTION task("ha rss","ids_stores");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha rss add argument: Add an RS secondary server to a primary server (SQL administration API)

Use the ha rss add argument with the **admin()** or **task()** function to associate a primary server with a remote standalone (RS) secondary server.

Syntax



Element	Description	Key Considerations
password	The password to set or to change.	The password is used only during the first connection attempt. After the primary and secondary server have connected, the password cannot be changed.

Element	Description	Key Considerations
ha_alias	The high-availability alias of the database server to convert to an RS secondary server.	The name must be defined in the HA_ALIAS configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function from an established primary server to create an RS secondary server and register the RS secondary server name in the **sysha** database.

This function is equivalent to the **onmode -d add RSS** command.

Example

The following example associates a server with a high-availability alias of **ids_stores2** as an RS secondary server to the primary server:

```
EXECUTE FUNCTION task("ha rss add","ids_stores2");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha rss change argument: Change the password of an RS secondary server (SQL administration API)

Use the ha rss change argument with the **admin()** or **task()** function to change the connection password for the specified RS secondary server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"ha rss change"--,--"secondary_server"-- , -"password"------->

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
password	The password to set or to change.	The password is used only during the first connection attempt. After the primary and secondary server have connected, the password cannot be changed.
secondary_server	The name of the database server to convert to an RS secondary server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on an established primary server to change the password for the connection between the primary and secondary server.

This function is equivalent to the **onmode -d change RSS** command.

Example

The following example changes the password for the RS secondary server to `secure`:

```
EXECUTE FUNCTION task("ha rss change","ids_stores2","secure");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha rss delete argument: Delete an RS secondary server (SQL administration API)

Use the ha rss delete argument with the **admin()** or **task()** function to stop replication and delete the RS secondary server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'
```

```
>--"ha rss delete"--,--"secondary_server"--)--;---------------><
```

Element	Description	Key Considerations
secondary_server	The name of the database server to convert to an RS secondary server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function from an established primary server to convert the RS secondary server to a standard server and delete the RS secondary server.

This function is equivalent to the **onmode -d delete RSS** command.

Example

The following example deletes the RS secondary server named **ids_stores2**:

```
EXECUTE FUNCTION task("ha rss delete","ids_stores2");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha sds clear argument: Stop shared-disk replication (SQL administration API)

Use the ha sds clear argument with the **admin()** or **task()** function to stop replication to shared disk (SD) secondary servers and convert the primary server to a standard server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"ha sds clear"--,--"primary_server"--)--;------------------><
```

Element	Description	Key Considerations
primary_server	The name of the primary server to convert to a standard server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on an established primary server to stop replication to the SD secondary servers.

This function is equivalent to the **onmode -d clear SDS primary** command.

Example

The following example stops replication from the primary server named **ids_stores** to SD secondary servers:

```
EXECUTE FUNCTION task("ha sds clear","ids_stores");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha sds primary argument: Convert an SD secondary server to a primary server (SQL administration API)

Use the ha sds primary argument with the **admin()** or **task()** function to change a shared disk (SD) secondary server to a primary server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"ha sds primary"-------+--->
                     '-task--'     '-"ha sds primary force"-'
```

```
>--,--"secondary_server"--)--;--------------------------------><
```

Element	Description	Key Considerations
secondary_server	The name of the SD secondary server to set as a primary server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on an established SD secondary server to convert it to the primary server.

Use the ha sds primary argument to convert an inactive SD secondary server to a primary server, if the SD secondary servers are connected to it.

Use the ha sds primary force argument to convert an inactive SD secondary server to a primary server, whether or not any SD secondary servers are connected to it. If sessions are active, the call succeeds, but the shared disk subsystem can become corrupted.

This function is equivalent to the **onmode -d make primary** command.

Example

The following example converts an SD secondary server named **ids_stores3** to the primary server:

```
EXECUTE FUNCTION task("ha sds primary","ids_stores3");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha sds set argument: Create a shared-disk primary server (SQL administration API)

Use the ha sds set argument with the **admin()** or **task()** function to define a primary server to replicate to shared disk (SD) secondary servers.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"ha sds set"-------+------->
                     '-task--'     '-"ha sds set force"-'

>--,--"primary_server"--)--;-----------------------------------><
```

Element	Description	Key Considerations
primary_server	The name of the database server to set as a primary server.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on a standard server to define it as a primary server for SD secondary servers.

Use the ha sds set argument to define an inactive standard server as a primary server, if the SD secondary servers are connected to it.

Use the ha sds set force argument to define an inactive standard server as a primary server, whether or not any SD secondary servers are connected to it. If sessions are active, the call succeeds, but the shared disk subsystem can become corrupted.

This function is equivalent to the **onmode -d set SDS primary** command.

Example

The following example converts a standard server named **ids_stores** to a primary server:

```
EXECUTE FUNCTION task("ha sds set","ids_stores");
```

Related reference:
onmode -d: Set High Availability server characteristics

ha set idxauto argument: Replicate indexes to secondary servers (SQL administration API)

Use the ha set idxauto argument with the **admin()** or **task()** function to control whether indexes are automatically replicated to secondary servers.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"ha set idxauto off"-+----->
                     '-task--'     '-"ha set idxauto on"--'

>--)--;-------------------------------------------------------><
```

Usage

Run this function on an established primary server to enable or disable automatic index replication to secondary servers.

You can run this function on any type of primary server.

This function is equivalent to the **onmode -d idxauto** command.

Example

The following example enables automatic index replication:

```
EXECUTE FUNCTION task("ha set idxauto on");
```

Related reference:
onmode -d command: Replicate an index with data-replication

ha set ipl argument: Log index builds on the primary server (SQL administration API)

Use the ha set ipl argument with the **admin()** or **task()** function to control whether to log index builds on the primary server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"ha set ipl off"-+--)--;-->
                     '-task--'     '-"ha set ipl on"--'
```

Usage

Run this function on an established primary server to enable or disable the logging of index builds. This function resets the value of the LOG_INDEX_BUILDS configuration parameter in the ONCONFIG file.

You can run this function on any type of primary server.

This function is equivalent to the **onmode -wf LOG_INDEX_BUILDS** command.

Example

The following example enables the logging of index builds:

```
EXECUTE FUNCTION task("ha set ipl on");
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

ha set primary argument: Define an HDR primary server (SQL administration API)

Use the ha set primary argument with the **admin()** or **task()** function to define a High-Availability Data Replication (HDR) primary server and specify the secondary server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"ha set primary"--,--"secondary_server"--)--;--------------><
```

Element	Description	Key Considerations

Element	Description	Key Considerations
secondary_server	The name of the HDR secondary server to connect to.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on a standard server to convert it to an HDR primary server and connect to the specified HDR secondary server. If the connection is successful, replication begins.

This function is equivalent to the **onmode -d primary** command.

Example

The following example converts a standard server named **ids_stores** to an HDR primary server:

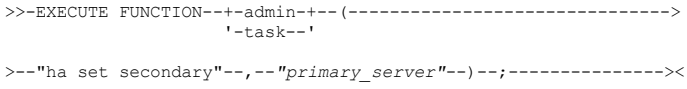
```
EXECUTE FUNCTION task("ha set primary","ids_stores");
```

Related reference:
onmode -d: Set data-replication types

ha set secondary argument: Define an HDR secondary server (SQL administration API)

Use the ha set secondary argument with the **admin()** or **task()** function to define a High-Availability Data Replication (HDR) secondary server and specify the primary server.

Syntax



Element	Description	Key Considerations
primary_server	The name of the HDR primary server to connect to.	The name must be defined in the DBSERVERNAME or DBSERVERALIASES configuration parameter, or as an Enterprise Replication group name.

Usage

Run this function on a standard database server to convert it to an HDR secondary server, and connect to the specified primary server. If the connection is successful, replication begins.

This function is equivalent to the **onmode -d secondary** command.

Example

The following example converts a standard server to an HDR secondary server, with a primary server named **ids_stores**:

```
EXECUTE FUNCTION task("ha set secondary","ids_stores");
```

Related reference:
onmode -d: Set data-replication types

ha set standard argument: Convert an HDR server into a standard server (SQL administration API)

Use the **ha set standard** argument with the **admin()** or **task()** function to convert a High-Availability Data Replication (HDR) primary or secondary server to a standard server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"ha set standard"--)--;----->
                    '-task--'
```

Usage

Run this function on a HDR primary or secondary server to convert it to a standard server. The connection between the primary and secondary servers is dropped and replication stops. The mode of the other server in the HDR pair is not changed.

This function is equivalent to the **onmode -d standard** command.

Example

The following example converts an HDR secondary server to a standard server:

```
EXECUTE FUNCTION task("ha set standard");
```

Related reference:
onmode -d: Set data-replication types

ha set timeout argument: Change SD secondary server timeout (SQL administration API)

Use the ha set timeout argument with the **admin()** or **task()** function to change the amount of time in seconds that the primary server waits for acknowledgments from shared disk (SD) secondary servers.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                    '-task--'

>--"ha set timeout"--,--"seconds"--)--;------------------------><
```

Element	Description	Key Considerations
seconds	The number of seconds the primary server waits before disconnecting the SD secondary server.	The value must be a positive integer in the range: • from 2 • to 2 147 483 647

Usage

Run this function on an established shared disk primary server to specify the amount of time in seconds that the primary server waits for a log position acknowledgment to be sent from an SD secondary server. If there is no log position acknowledgment received from the SD secondary server in the specified amount of time, the primary server disconnects from the SD secondary server and continues. After waiting for the specified number of seconds, the primary server starts removing SD secondary servers if page flushing has timed out while waiting for an SD secondary server.

This function resets the value of the SDS_TIMEOUT configuration parameter in the ONCONFIG file.

This function is equivalent to the **onmode -wf SDS_TIMEOUT** command.

Example

The following example sets the timeout period to 5 seconds:

```
EXECUTE FUNCTION task("ha set timeout","5");
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

import config argument: Import configuration parameter values (SQL administration API)

Use the import config argument with the **admin()** or **task()** function to import a file that contains one or more dynamically updatable configuration parameters and apply the new values.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--import config--"---------->
                     '-task--'

>--,--"file_path"--)--;---------------------------------------><
```

Table 1. import config command elements

Element	Description	Key Considerations
file_path	The full path name of the previously exported file that contains the names and values of one or more dynamically updatable configuration parameters	

Usage

Dynamically updatable configuration parameters are those parameters that you can change for a session with an **onmode -wf** or **onmode -wm** command.

You must specify the full path name. You cannot specify a relative path.

This command is the equivalent of the **onmode –wi** commend.

Example

The following command imports a file named `cfg_12` in the `/tmp` directory:

```
EXECUTE FUNCTION task("import config", "/tmp/cfg_12");
```

Related tasks:
Modifying the onconfig file
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters
export config argument: Export configuration parameter values (SQL administration API)
onmode -wi: Import a configuration parameter file

index compress repack shrink arguments: Optimize the storage of B-tree indexes (SQL administration API)

Use the index compress repack shrink argument with the **admin()** or **task()** function to compress detached B-tree indexes, consolidate free space (repack), and return free space (shrink) in partitions.

Syntax: Index compression command arguments

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--index--------------------->
                     '-task--'

>--+-compress--+--------+--+--------+--+----------+-+--"-------->
   |           '-repack-'  '-shrink-'  '-parallel-' |
   +-repack--+--------+--+----------+---------------+
   |         '-shrink-'  '-parallel-'               |
   '-shrink----------------------------------------'

>--,--"index_name"--,--"database_name"--,--"owner"--)--;------->
```

Command arguments

The following table contains a brief explanation of each argument.

Table 1. Arguments for index compression operations

Argument	Description
compress	Compresses the index.
parallel	Runs the compress or repack operation in parallel. A thread is started for each fragment of the table or fragment list and the operation is run in parallel across those fragments.
repack	Consolidates free space by moving data to the front of the index.
shrink	Returns free space at the end of the index to the dbspace, thus reducing the total size of the index.

Command elements

The following table shows the elements that you can use to compress, repack, and shrink indexes.

Table 2. Index compress command elements

Element	Description	Key Considerations
index_name	The name of the index that you want to compress.	Required. You must use the same uppercase or lowercase letters that are in system catalog tables.
database_name	The name of the database that contains the index that you want to compress.	Optional. If you do not specify a database, the database server uses the current database. If you enter a database name, you must use the same uppercase or lowercase letters that are in system catalog tables.
owner	The name of the owner of the database that contains the index that you want to compress.	Optional. If you do not specify an owner, the database server uses the current owner. If you enter an owner name, you must use the same uppercase or lowercase letters that are in system catalog tables.

Usage

You can compress a detached B-tree index that is on a fragmented or non-fragmented table. You cannot compress an attached index.

To be compressed, an index must have at least 2000 keys. If a fragment within the index does not have at least 2000 keys, the database server does not compress the index or fragment when it creates the index. The index remains uncompressed even if new keys are added to it. If you want to compress the index, run another SQL Admin API **task()** or **admin()** function with the index compress argument.

To determine if an index contains the minimum number of keys, run the **oncheck -pT** command and view information in the Number of keys field.

Typically you perform a repack operation after a compress operation and the shrink after a repack operation.

The compression operation compresses only the leaves (bottom level) of the index.

You can cancel a command, for example, by typing CTRL-C in DB-Access.

You cannot uncompress an index. If you want an uncompressed index, you can drop the compressed index and recreate it.

Example

The following command compresses, repacks, and shrinks an index in parallel.

```
EXECUTE FUNCTION task("index compress repack shrink parallel",
"ind5", "customer", "jayson");
```


index estimate_compression argument: Estimate index compression (SQL administration API)

Use the index estimate_compression argument with the **admin()** or **task()** function to estimate if you can save disk space by compressing a B-tree index.

```
Syntax: index estimate_compression command argument

>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"--index estimate_compression--"-------------------------->

>--,--"index_name"--,--"database_name"--,--"owner"--)--;------><
```

Command elements

The following table shows the elements that you can use to estimate index compression.

Table 1. Index estimate_compression command elements

Element	Description	Key Considerations
index_name	The name of the index for which you want to estimate compression benefits.	Required. You must use the same uppercase or lowercase letters that are in system catalog tables.
database_name	The name of the database that contains the index.	Optional. If you do not specify a database, the database server uses the current database. If you enter a database name, you must use the same uppercase or lowercase letters that are in system catalog tables.

Element	Description	Key Considerations
owner	The name of the owner of the database that contains the index.	Optional for an index. If you do not specify an owner, the database server uses the current owner. If you enter an owner name, you must use the same uppercase or lowercase letters that are in system catalog tables.

Usage

You can estimate compression only for a detached B-tree index on a fragmented or non-fragmented table.

The estimate compression operation displays the name of the index, the estimated compression ratio that can be achieved, the current compression ratio, and an estimate of the percentage gain or loss. The current ratio is 0.0 percent if the index is not compressed.

Example

The following command estimates compression benefits for an index named **ind4** in the **customer** database for which **anjul** is the owner.

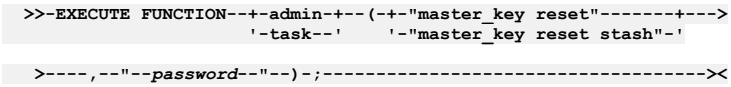
```
EXECUTE FUNCTION task("index estimate_compression","ind4",
"customer","anjul");
```

Related reference:
Output of the estimate compression operation (SQL administration API)

master_key reset argument: Change the keystore password (SQL administration API)

Use the master_key reset argument with the **admin()** or **task()** function to change the password for the storage space encryption keystore. The password is used to encrypt the keystore for storage space encryption.

Syntax



Element	Purpose	Key considerations
password	A password that you supply for the storage space encryption keystore.	The maximum length is 32 bytes and the minimum length is 8 bytes.

Usage

You must be logged in as user **informix** or the root user to run this command.

The password protects the storage space encryption keystore. You can run the **admin()** or **task()** function with the **master_key reset** argument at any time to change the password. The keystore gets encrypted with the new password. The optional keyword "stash" enforces the stashing of the new password even if the old password was not stashed before.

Example

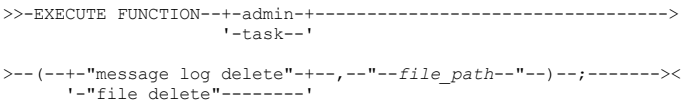
The following example shows how to change the password:

```
execute function task("master_key reset", "This is my new password");
```


message log delete argument: Delete a message log file (SQL administration API)

Use the message log delete argument or file delete argument with the **admin()** or **task()** function to specify the particular online, ON-Bar activity, or ON-Bar debug message log to delete.

Syntax



Element	Purpose	Key considerations
file_path	Full path name for the particular online, ON-Bar activity, or ON-Bar debug message log file.	

Examples

The following examples show the arguments that you can use to delete the /usr/informix/online.log file:

```
execute function task("message log delete", "/usr/informix/online.log");
```

```
execute function task("file delete", "/usr/informix/online.log");
```

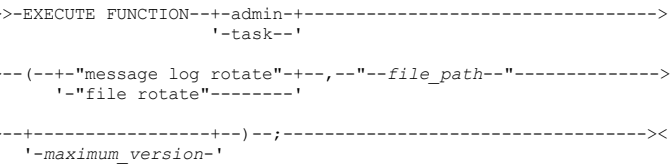
Related reference:
message log rotate argument: Rotate the message log file (SQL administration API)
message log truncate argument: Delete the contents of a message log file (SQL administration API)
file status argument: Display the status of a message log file (SQL administration API)

message log rotate argument: Rotate the message log file (SQL administration API)

Use the message log rotate argument or the file rotate argument with the **admin()** or **task()** function to specify the particular online, ON-Bar activity, or ON-Bar debug message log file to rotate, and to indicate the maximum number of message logs to rotate.

When the message log file rotates, the database server switches to a new online message log file and increments the ID numbers for the previous log files by one. When the maximum number of log files is reached, the log file with the highest ID is deleted.

Syntax



Element	Purpose	Key considerations
file_path	Full path name of the online, ON-Bar activity, or ON-Bar debug message log file that the server will rotate, for example, /usr/informix/online.log.	
maximum_version	The log file with the highest ID. This is the maximum message log version that the server will rotate.	

Examples

The following examples show the arguments that you can use to rotate a maximum of 52 /usr/informix/online.log files:

```
execute function task("message log rotate", "/usr/informix/online.log",52);
```

```
execute function task("file rotate", "/usr/informix/online.log",52);
```

When the database server rotates these files, the server deletes version 52 of the file. Version 51 becomes version 52, version 50 becomes version 51, and so on. The new online log becomes version 1.

Related reference:
message log truncate argument: Delete the contents of a message log file (SQL administration API)
message log delete argument: Delete a message log file (SQL administration API)
file status argument: Display the status of a message log file (SQL administration API)

message log truncate argument: Delete the contents of a message log file (SQL administration API)

Use the message log truncate argument or file truncate argument with the **admin()** or **task()** function to specify the particular online, ON-Bar activity, or ON-Bar debug message log file to truncate. When the database server truncates a message log file, it deletes the messages in the log file, but keeps the log file.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                    '-task--'

>--(--+-"message log truncate"-+--,--"--file_path--"--)--;----->< 
      '-"file truncate"--------'
```

Element	Purpose	Key considerations
file_path	Full path name for the online, ON-Bar activity, or ON-Bar debug message log file.	

Examples

The following examples show the arguments that you can use to truncate the /usr/informix/online.log file:

```
execute function task("message log truncate", "/usr/informix/online.log");
```

```
execute function task("file truncate", "/usr/informix/online.log");
```

Related reference:
message log rotate argument: Rotate the message log file (SQL administration API)
message log delete argument: Delete a message log file (SQL administration API)
file status argument: Display the status of a message log file (SQL administration API)

modify chunk extend argument: Extend the size of a chunk (SQL administration API)

Use the modify chunk extend argument with the **admin()** or **task()** function to extend the size of the chunk by a specified minimum amount. The chunk must be marked as extendable.

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                    '-task--'

>--(--"--modify chunk extend--"--,--"chunk_number"--,--"extend_amount"--)-- ;-><
```

Element	Description	Key Considerations
chunk_number	The number of the chunk.	
extend_amount	The minimum amount of space in kilobytes to add to the chunk.	See admin() and task() Argument Size Specifications.

Usage

You must mark a chunk as extendable before the chunk can be extended, either manually or automatically. Use the modify chunk extendable argument with the **admin()** or **task()** function to mark a chunk as extendable.

The **modify chunk extend** SQL administration API command is an alternative to the **adm_add_storage** task that the server can run to automatically extend the size of a chunk when the space containing the chunk runs low or out of free pages.

You cannot extend a chunk in a mirrored space, and you will receive an error if you provide the number of a mirror chunk when you run a **modify chunk extend** SQL administration API command.

To identify primary and mirror chunks in a mirrored space, look for the `P` (primary) or `M` (mirror) in position 1 of the `flags` field in **onstat -d** command output.

The server might round up the requested size, depending on the page size and the configured create size and extend size of the space.

Examples

Suppose that your **onstat -d** command output shows that chunk number 3 is a mirror chunk and chunk number 4 is a not a mirror chunk. You cannot extend the size of chunk number 3. However, you can modify chunk number 4. The following command extends the size of chunk number 4 by 10000 kilobytes:

```
EXECUTE FUNCTION task("modify chunk extend", "4", "10000");
```

Related reference:
modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)
modify chunk extend argument: Extend the size of a chunk (SQL administration API)
modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)
modify space expand argument: Expand the size of a space (SQL administration API)
onstat -d command: Print chunk information

modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)

Use the modify chunk extendable argument with the **admin()** or **task()** function to specify that a particular chunk in an unmirrored dbspace or temporary dbspace can be extended..

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"--modify chunk extendable--"--,--"--chunk_number--"--)-- ;-><
```

Element	Description	Key Considerations
chunk_number	The number of the chunk.	

Usage

If a chunk is marked as extendable, either:

- The server can automatically extend the chunk when the unmirrored dbspace or temporary dbspace containing the chunk runs low or out of free pages.
- You can use the modify chunk extend argument with the **admin()** or **task()** function to extend the size of the chunk.

However, if the extend size for the dbspace or temporary dbspace is set to `0`, the server cannot automatically extend an extendable chunk in that space. In this situation, you can still manually extend the chunk.

The server will automatically mark chunks that are allocated from extendable storage pool entries as extendable. Therefore, you do not need to mark these chunks as extendable. For information on extendable storage pool entries, see storagepool add argument: Add a storage pool entry (SQL administration API).

Chunks in mirrored spaces cannot be extended. If you try to make a mirror chunk extendable, you will receive an error.

To identify primary and mirror chunks in a mirrored space, look for the `P` (primary) or `M` (mirror) in position 1 of the `flags` field in **onstat -d** command output.

Example

The following snippet of **onstat -d** output shows that chunk number 3 is a mirror chunk:

```
Chunks
address      chunk/dbs  offset   size     free     bpages   flags pathname
451191c8     1      1   0        225000   101572           PO-B-- /reg1/rootchunk
451197d0     2      2   0        1250     1149             PO-B-- /reg1/dbs1
451199d0     3      3   0        1250     1149             PO-B-- /reg1/dbs2
46a36638     3      3   0        1250     0                MO-B-- /reg1/chunk2
45119bd0     4      4   0        1250     1149             PO-B-- /reg1/dbs3
```

Thus, you cannot extend the size of chunk number 3. However, you can specify that chunk number 4 is extendable, as follows:

```
EXECUTE FUNCTION sysadmin:task("modify chunk extendable", "4");
```

Related reference:
modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API)
modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)
modify chunk extend argument: Extend the size of a chunk (SQL administration API)
modify space expand argument: Expand the size of a space (SQL administration API)

modify chunk extendable off argument: Mark a chunk as not extendable (SQL administration API)

Use the modify chunk extendable off argument with the **admin()** or **task()** function to specify that a particular chunk cannot be extended.

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"--modify chunk extendable off--"--,--"--chunk_number--"--)-- ;-><
```

Element	Description	Key Considerations
chunk_number	The number of the chunk.	

Usage

The default status for chunks is not extendable. If you previously marked a chunk as extendable, you can change the status to not extendable.

If a chunk is marked as not extendable:

- The server cannot automatically extend the chunk when the space containing the chunk runs low or out of free pages.
- You cannot manually extend the size of the chunk.

If the storage pool contains entries, the server can extend a storage space by adding another chunk to the storage space.

Example

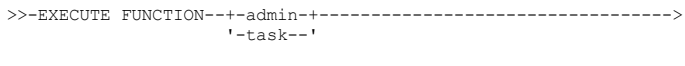
The following example specifies that the you or the server cannot extend chunk 9:

```
EXECUTE FUNCTION task("modify chunk extendable off", "9");
```

Related reference:
modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)

modify chunk swap_mirror argument: Switch primary and mirror chunk files without any downtime(SQL administration API)

Use the modify chunk swap_mirror argument with the **admin()** or **task()** function to easily migrate data from old disk drives to new ones without downtime.



Element	Description	Key Considerations
chunk_number	The number of the chunk.	The chunk must be mirrored. Both the primary and the mirror chunk must be on-line.

Usage

To migrate data in a chunk from one disk drive to another on the fly, do the following:

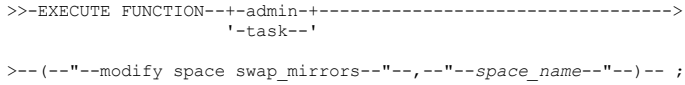
1. Add a mirror chunk to the original, primary chunk, placing the mirror on the new disk drive.
   Note: All chunks in a mirrored space must be mirrored.
   For more information see, **onspaces -m** or **add mirror** SQL administration API command.
2. Swap the primary chunk and the mirror using the **modify chunk swap_mirror** command.
3. Drop the mirror chunk (the original primary) by turning off mirroring for the space. For more information see, **onspaces -r** or **stop mirroring** SQL administration API command.

Example

```
EXECUTE FUNCTION sysadmin:task("create dbspace","newdbs","/prod1/IFX_CHUNKS/chunk5",10000,0);
EXECUTE FUNCTION sysadmin:task("add mirror","newdbs","/prod1/IFX_CHUNKS/chunk5",0,"/prod8/IFX_CHUNKS/chunk1",0);
select chknum from sysmaster:syschunks where fname = "/prod1/IFX_CHUNKS/chunk5";
EXECUTE FUNCTION sysadmin:task("modify chunk swap_mirror", "12");
EXECUTE FUNCTION sysadmin:task("stop mirroring","newdbs");
```


modify space swap_mirror argument: Switch all primary and mirror chunk files for a space without any downtime (SQL administration API)

Use the modify space swap_mirrors argument with the **admin()** or **task()** function to easily migrate data from old disk drives to new ones without downtime.



Element	Description	Key Considerations
space_name	The name of the storage space	The space must be mirrored. All primary and mirror chunks in the space must be on-line.

Usage

To migrate data in a chunk from one disk drive to another without any downtime, do the following:

1. Add mirror chunks to the original, primary chunks using the **onspaces -m** command, placing the mirrors on the new disk drive.
   Note: All chunks in a mirrored space must be mirrored.
2. Swap the primary chunks and the mirrors using the **modify space swap_mirrors** command.
3. Drop the mirror chunks (the original primary) by turning off mirroring for the space. For more information see, **onspaces -r** or **stop mirroring** SQL administration API command.

Example

```
EXECUTE FUNCTION sysadmin:task("create dbspace","newdbs","/prod1/IFX_CHUNKS/chunk5",10000,0);
EXECUTE FUNCTION sysadmin:task("add chunk","newdbs","/prod1/IFX_CHUNKS/chunk6",10000,0);
onspaces -m newdbs -p /prod1/IFX_CHUNKS/chunk5 -o 0 -m /prod8/IFX_CHUNKS/chunk1 0 -p /prod1/IFX_CHUNKS/chunk6 -o 0 -m
/prod8/IFX_CHUNKS/chunk2 0 -y
EXECUTE FUNCTION sysadmin:task("modify space swap_mirrors", "newdbs");
EXECUTE FUNCTION sysadmin:task("stop mirroring","newdbs");
```


modify config arguments: Modify configuration parameters (SQL administration API)

Use the modify config argument with the **admin()** or **task()** function to change the value of a configuration parameter in memory until you restart the database server. Use the modify config persistent argument to change the value of a configuration parameter in memory and preserve the value in the onconfig file after you restart the server.

Syntax

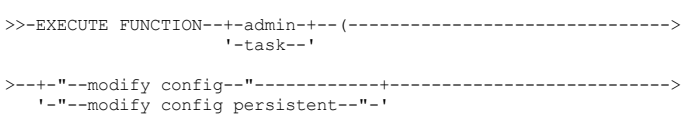


Table 1. modify config command elements

Element	Description	Key Considerations
configuration_parameter_name	The name of the configuration parameter that you want to modify.	
new_value	The new value of the configuration parameter.	For information about the valid values for a configuration parameter, see Database configuration parameters.

Usage

This SQL administration API command is equivalent to using an **onmode -wm** or **-wf** command to change the value of a configuration parameter.

Examples

The following command changes the value of the DYNAMIC_LOGS configuration parameter to 2 in memory for current use:

```
EXECUTE FUNCTION task("modify config","DYNAMIC_LOGS",
"2");
```

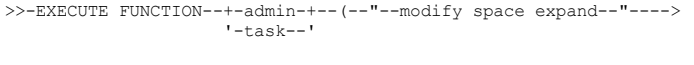
The following command changes the value of the DYNAMIC_LOGS configuration parameter for current use. The changed value remains in the onconfig file after you restart the server.

```
EXECUTE FUNCTION task("modify config persistent","DYNAMIC_LOGS",
"2");
```

Related tasks:
Modifying the onconfig file
Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

modify space expand argument: Expand the size of a space (SQL administration API)

Use the modify space expand argument with the **admin()** or **task()** function to immediately expand the size of a space, when you do not want to wait for Informix® to automatically expand the space.



```
>--,--"space_name"--,--"minimum_size"--)--;--------------------><
```

Element	Description	Key Considerations
space_name	The name of the storage space.	
minimum_size	The minimum size by which you want to expand the space.	See admin() and task() Argument Size Specifications.

Usage

The modify space expand SQL administration API command expands a storage space immediately, either by extending an extendable chunk in the space or by adding a new chunk. The create size and extend size settings for the space do not affect this operation.

The actual number of kilobytes added to the space might exceed your requested size, depending on factors such as the page size of the space and the chunk size settings for available entries in the storage pool.

The storage pool must contain entries (such as raw devices, cooked files, or directories) that the server can use to expand the space.

After you run a modify space expand SQL administration API command, Informix first attempts to expand the space by extending an extendable chunk in the space. If the space does not contain any extendable chunks, the server uses entries in the storage pool to expand the space.

You cannot expand a mirrored storage space.

Examples

The following command expands `dbspace5` by 10 megabytes:

```
EXECUTE FUNCTION task("modify space expand", "dbspace5", "10 MB");
```

Related reference:
modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)
modify chunk extend argument: Extend the size of a chunk (SQL administration API)
modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)
create chunk from storagepool argument: Create a chunk from the storage pool (SQL administration API)

modify space sp_sizes argument: Modify sizes of an extendable storage space (SQL administration API)

Use the modify space sp_sizes argument with the **admin()** or **task()** function to modify the create, extend, and maximum sizes that are associated with expanding a storage space. Modify the sizes to control how Informix® uses storage pool entries for a particular storage space.

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--modify space sp_sizes--"--,--"--space_name--"------------->

>--,--"--new_create_size--"--,--"--new_extend_size--"----------->

>--+-------------------+--)--;--------------------------------><
   '-,--"--max_size--"-'
```

Element	Description	Key Considerations
space_name	The name of the storage space.	
max_size	The maximum size of the storage space, in KB.	The default maximum size is 0, which indicates unlimited size.
new_create_size	The minimum size of a new chunk that the server can create when automatically expanding this space using the storage pool. You can define the size as a number of KB or as a percentage of the total space.	The default create size is set to 10 percent of the total size of the space. The size that you specify affects chunks the server creates automatically. It does not affect any manual chunks that you might create for the associated space.
new_extend_size	The minimum size that the server can use when automatically extending a chunk in an unmirrored dbspace or temporary dbspace. The size can be a specified number of KB or a percentage of the total space.	The default extend size is 10 MB. The size that you specify affects chunks the server extends automatically. It does not affect any manual chunk extensions that you might initiate for the associated space.

Usage

If the create or extend size value is 100 or a lower value, Informix interprets the value as a percentage (for example, 10 = 10 percent and 2.84 = 2.84 percent). If the value is 1000 or higher, the server interprets the value as a specific number of KB. Values 100 - 1000 are not valid.

If you set the create size and the extend size to `0`, Informix does not automatically expand the space, even when the space becomes full. Additionally, if you set the extend size to `0`, you also remove the “Extendable” flag from all chunks in that space. This is an easy way to mark all chunks in a space as not extendable, using one operation.

The create and extend size values are minimum sizes. The actual size by which a space is expanded might be larger, depending on the chunk size of the storage pool entry that the server is using or the amount of space that the server needs at that particular time.

For example, suppose you created a storage pool entry to expand storage space when necessary. Then suppose that a dbspace named **logdbs** is out of free pages and requires an extra 500 MB for a new log. If none of the chunks in **logdbs** can be extended, Informix adds a chunk that has the minimum size that is specified by the create size value for the **logdbs** dbspace. If the create size for the **logdbs** dbspace is less than or equal to 500 MB, the server attempts to find a minimum of 500 MB of space. If the create size for **logdbs** is 1 GB, the server ignores the requested size and adds a 1 GB chunk.

If the server is unable to find the minimum amount of space that is required, the server returns an out-of-space error and the log creation fails.

If you set the maximum size of the storage space to a value other than 0, the storage space cannot exceed the maximum size, regardless of the new extend size. When the amount of expansion space that is left before the maximum size is less than the new extend size, the extend size is truncated and the space is extended to the maximum size. The event alarm 86001 is triggered when the space reaches the maximum size. When the amount of expansion space left is less than the minimum chunk size for the storage pool, the space is not expanded and an error is returned.

Examples

The following command sets the minimum create size to 60 MB, the minimum extend size to 10 MB, and the maximum size to 100 MB for a dbspace that is named **dbspace3**:

```
EXECUTE FUNCTION task("modify space sp_sizes", "dbspace3", "60000",
     "10000", "100000");
```

The following command sets the minimum create size to 20 percent and the minimum extend size 1.5 percent for a dbspace that is named **dbspace8**:

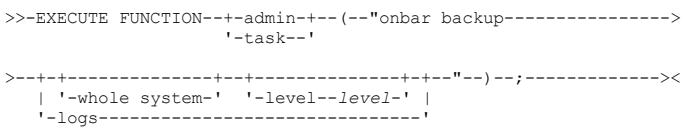
```
EXECUTE FUNCTION task("modify space sp_sizes", "dbspace8", "20", "1.5");
```

Related reference:
modify chunk extendable argument: Mark a chunk as extendable (SQL administration API)
modify chunk extend argument: Extend the size of a chunk (SQL administration API)
modify space expand argument: Expand the size of a space (SQL administration API)

onbar argument: Backup the storage spaces (SQL administration API)

Use the onbar argument with the **admin()** or **task()** function to backup the storage spaces.

Syntax



Element	Purpose	Key considerations
onbar backup	Performs a complete backup of the storage spaces	If you do not specify a level, a level 0 backup is performed.
whole system	Performs a whole-system backup	This is equivalent of issuing the **onbar** command with the -w option from the command line. If you do not specify a level, a level 0 backup is performed.
level *level*	Specifies the level of backup to perform on storage spaces: • 0 for a complete backup. This is the default. • 1 for changes since the last level-0 backup • 2 for changes since the last level-1 backup	If you request an incremental backup and a level backup has not been performed for a particular storage space, this function backs up that storage space at the previous level. For example, if you request a level-1 backup, and the function finds no level-0 backup, a level-0 backup is made instead. This is equivalent of issuing the **onbar** command with the -L *level* option from the command line.
logs	Performs a back of the logical-log files	This is equivalent of issuing the **onbar** command with the -l option from the command line.

Usage

This function is equivalent to invoking specific options of the **onbar** command to create backups of the storage spaces and logical-log files.

Examples

The following example creates a level 0 backup of the storage spaces:

```
EXECUTE FUNCTION task("onbar backup");
```

The following example creates a level 1 backup of the storage spaces:

```
EXECUTE FUNCTION task("onbar backup level 1");
```

The following example creates a level 1 backup of the logical-log files:

```
EXECUTE FUNCTION task("onbar backup logs");
```

The following example creates a whole system level 0 backup of the storage spaces:

```
EXECUTE FUNCTION task("onbar backup whole system");
```

The following example creates a whole system level 2 backup of the storage spaces:

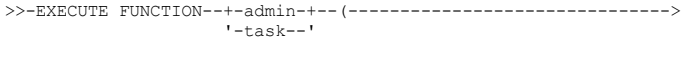
```
EXECUTE FUNCTION task("onbar backup whole system level 2");
```

Related information:
Back up with ON-Bar
onbar -b syntax: Backing up

onmode and a arguments: Add a shared-memory segment (SQL administration API)

Use the onmode and a arguments with the **admin()** or **task()** function to add a shared-memory segment.

Syntax



Element	Description	Key Considerations
memory_size	The size, in kilobytes, of a new virtual shared-memory segment.	The value of *size* must be a positive integer that does not exceed the operating-system limit on the size of shared-memory segments.

Usage

Ordinarily, you do not need to add segments to the virtual portion of shared memory because the database server automatically adds segments as they are needed. However, as segments are added, the database server might reach the operating-system limit for the maximum number of segments before it acquires the memory that it needs. This situation typically occurs when the SHMADD configuration parameter is set so small that the database server exhausts the number of available segments before it acquires the memory that it needs for some operation.

You can use this function to add a segment that is larger than the size specified by the SHMADD configuration parameter. By using this function to add a segment, you can adhere to the operating system limit for segments while meeting the need that the database server has for more memory.

This function is equivalent to the **onmode -a** command.

Example

The following example adds 500 KB of virtual shared-memory:

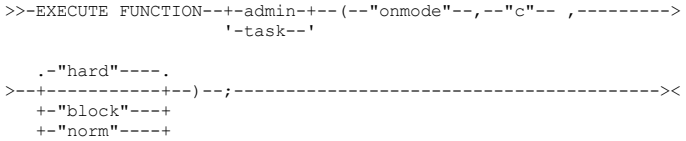
```
EXECUTE FUNCTION task("onmode","a","500");
```

Related reference:
onmode -a: Add a shared-memory segment

onmode and c arguments: Force a checkpoint (SQL administration API)

Use the onmode and c arguments with the **admin()** or **task()** function to force a checkpoint.

Syntax



```
'-"unblock"-'
```

Usage

This function forces a checkpoint that flushes the buffers to disk. You can use the **c** option to force a checkpoint if the most recent checkpoint record in the logical log was preventing the logical-log file from being freed (status U-B-L).

Use the block argument to prevent the database server from processing any transactions. Use this option to perform an external backup on Informix®. While the database server is blocked, users cannot access it, except in read-only mode. No transactions can complete until the database server is unblocked.

Use the hard argument to force a blocking checkpoint. This is the default.

Use the norm argument to force a nonblocking checkpoint.

Use the unblock argument to unblock the database server. When the database server is unblocked, data transactions and normal database server operations can resume. Use this option after you complete an external backup on Informix.

This function is equivalent to the **onmode -c** command.

Example

The following example starts a blocking checkpoint:

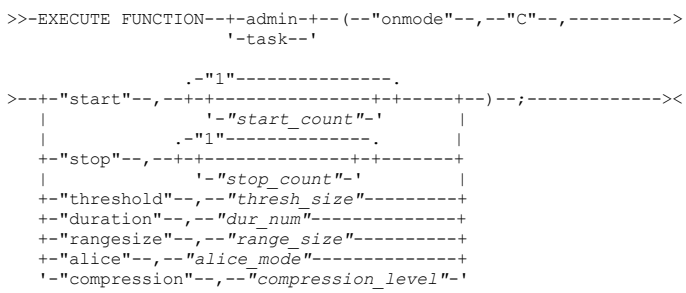
```
EXECUTE FUNCTION task("onmode","c","hard");
```

Related reference:
onmode -c: Force a checkpoint

onmode and C arguments: Control the B-tree scanner (SQL administration API)

Use the onmode and C arguments with the **admin()** or **task()** function to control the B-tree scanner for cleaning indexes of deleted items.

Syntax



Element	Description	Key Considerations
alice_mode	The **alice** mode for the system.	Valid integer values range from `0` (`OFF`) to `12`.
compression_level	For a database server instance, the level at which two partially used index pages are merged. The pages are merged if the data on those pages totals a set level.	Valid values for the level are `low`, `med` (medium), `high`, and `default`. The system default value is `med`.
dur_num	The number of seconds that the hot list is valid.	After this number of seconds expires, the hot list will be rebuilt by the next available B-tree scanner thread, even if unprocessed items are on the list. Scanners that are processing requests are not interrupted.
range_size	The size of an index before index range cleaning is enabled.	A size of `-1` can be used to disable range scanning.
start_count	The number of B-tree scanner threads to start.	If *start_count* is not specified, 1 more thread is started. A maximum of 32 threads can be started at one time. But, there is no limit on the number of scanner threads run simultaneously.
stop_count	The number of B-tree scanner threads to stop.	If *stop_count* is not specified, a single thread is stopped. Stopping all index scanners prevents all index cleaning. If you specify a larger *stop_count* value than the number of threads than are running, no error is issued, but all scanner threads are stopped.
thresh_size	The minimum number of deleted items an index must encounter before an index is placed on the hot list.	After all indexes above the threshold have been cleaned and there is no other work for the B-tree scanner to do, the indexes below the threshold are added to the hot list.

Usage

The B-tree scanner has statistical information that tracks index efficiency and how much extra work the index places on the server. Based on the amount of extra work the index has accomplished because of committed deleted index items, the B-tree scanner develops an ordered list of indexes that have caused the server to do extra work, called the hot list. The index causing the highest amount of extra work is cleaned first and the rest of the indexes are cleaned in descending order. The DBA can allocate cleaning threads dynamically to configure workloads.

This function is equivalent to the **onmode -C** command.

Example

The following commands start 60 B-tree scanner threads:

```
EXECUTE FUNCTION admin("onmode","C","start","30");
EXECUTE FUNCTION admin("onmode","C","start","30");
```

The following command stops all of these threads:

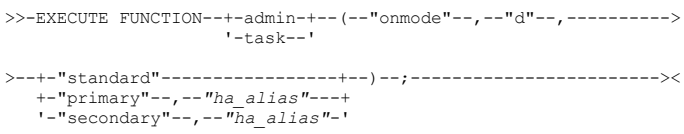
```
EXECUTE FUNCTION admin("onmode","C","stop","30000");
```

No error is issued when the *stop_count* value is greater than the number of running threads.
Related reference:
onmode -C: Control the B-tree scanner
BTSCANNER Configuration Parameter

onmode and d arguments: Set data-replication types (SQL administration API)

Use the onmode and d arguments with the **admin()** or **task()** function to change the mode of a server participating in high-availability data replication (HDR).

Syntax



Element	Description	Key Considerations
ha_alias	The high-availability alias of the primary or secondary database server.	*ha_alias* must correspond to the HA_ALIAS configuration parameter in the ONCONFIG file of the intended secondary database server.

Usage

Use this function to set the High-Availability Data Replication type as standard, primary, or secondary. You can use the standard argument when the database server is in quiescent, online, or read-only mode.

The *ha_alias* argument of the other database server in the data-replication pair and the type of a database server (standard, primary, or secondary) is preserved after reinitialization of shared memory.

The **standard** argument drops the connection between database servers in a data replication pair (if one exists) and sets the database server type of the current database server to standard. This option does not change the mode or type of the other database server in the pair.

The **primary** and *ha_alias* arguments set the database server type to primary and attempt to connect with the database server that *ha_alias* specifies. If the connection is successful, data replication is turned on. The primary database server goes into online mode, and the secondary database server goes into read-only mode. If the connection is not successful, the database server comes to online mode, but data replication is not turned on.

The **secondary** and *ha_alias* arguments set the database server type to secondary and attempt to connect with the database server that *ha_alias* specifies. If the connection is successful, data replication is turned on. The primary database server goes online, and the secondary database server goes into read-only mode. If the connection is not successful, the database server comes to read-only mode, but data replication is not turned on.

This function is equivalent to the **onmode -d** command.

Example

The following example sets a server named **ids_stores** as an HDR primary server:

```
EXECUTE FUNCTION task("onmode","d","primary","ids_stores");
```

Related reference:
onmode -d: Set data-replication types

onmode and D arguments: Set PDQ priority (SQL administration API)

Use the onmode and D arguments with the **admin()** or **task()** function to temporarily reset the PDQ resources that the database server can allocate to any one decision-support query.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"onmode"-- , -"D"--,--"max_priority"--)--;------------------><
```

Element	Description	Key Considerations
max_priority	The percentage of the user-requested PDQ resources actually allocated to the query.	The value must be an unsigned integer from 0 to 100.

Usage

Use this function to override the limit set by the MAX_PDQPRIORITY configuration parameter while the database server is online. The new values affect only the current instance of the database server; the values are not recorded in the **onconfig** file. If you shut down and restart the database server, the values of the parameter reverts to the values in the **onconfig** file.

This function is equivalent to the **onmode -D** command.

Example

The following example sets the percentage of PDQ resources that can be allocated to a query to 50 percent:

```
EXECUTE FUNCTION task("onmode","D","50");
```

Related reference:
onmode -D, -M, -Q, -S: Change decision-support parameters

onmode and e arguments: Change usage of the SQL statement cache (SQL administration API)

Use the onmode and e arguments with the **admin()** or **task()** function to temporarily change the mode of the SQL statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"e"--,---------->
                     '-task--'

>--+-"enable"-+--)--;----------------------------------------><
   +-"flush"--+
   +-"off"----+
   '-"on"-----'
```

Usage

Use the enable argument to enable the SQL statement cache if it is disabled. Individual user sessions can use the statement cache only after they perform either of the following actions:

- Set the environment variable **STMT_CACHE** to `1`.
- Execute the SQL statement SET STATEMENT CACHE ON.

Use the flush argument to flush the statements that are not in use from the SQL statement cache, which remains enabled. After the cache is flushed, the **onstat -g ssc ref_cnt** field shows 0.

Use the off argument to turn off the SQL statement cache, so that no statements are cached.

Use the on argument to cache all statements except those a user turns off by one of the following actions:

- Use this command to specify the OFF mode.
- Set the environment variable **STMT_CACHE** to `0`.
- Execute the SQL statement SET STATEMENT CACHE OFF statement.

This function cannot modify the STMT_CACHE configuration parameter setting in the ONCONFIG file, but the last argument overrides that setting (or the default value, if STMT_CACHE is not set). Any changes to the statement cache behavior that you make with this command are in effect for the current database server session only. When you restart the database server, it uses the setting of the STMT_CACHE parameter in the ONCONFIG file. If the STMT_CACHE configuration parameter is not defined in the ONCONFIG file, the server does not use a statement cache.

This function is equivalent to the **onmode -e** command.

Example

The following example enables the SQL statement cache:

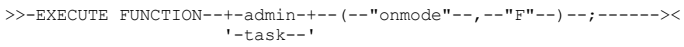
```
EXECUTE FUNCTION task("onmode","e","enable");
```

Related reference:
onmode -e: Change usage of the SQL statement cache

onmode and F arguments: Free unused memory segments (SQL administration API)

Use the onmode and F arguments with the **admin()** or **task()** function to free unused memory segments.

Syntax



Usage

When you execute this function, the memory manager examines each memory pool for unused memory. The memory manager immediately frees unused blocks of memory that it locates. After the memory manager checks each memory pool, it begins checking memory segments and frees any that the database server no longer needs.

Running this command causes a significant degradation of performance for any users that are active when you execute the utility. Although the execution time is brief (1 to 2 seconds), degradation for a single-user database server can reach 100 percent. Systems with multiple CPU virtual processors experience proportionately less degradation.

To confirm that the unused memory was freed, check the message log. If the memory manager frees one or more segments, it displays a message that indicates how many segments and bytes of memory were freed.

Tip: Run this command from an operating-system scheduling facility regularly and after the database server performs any function that creates more memory segments, including large index builds, sorts, or backups.
This function is equivalent to the **onmode -F** command.

Example

The following example frees unused memory blocks:

```
EXECUTE FUNCTION task("onmode","F");
```

Related reference:
onmode -F: Free unused memory segments

onmode and h arguments: Update sqlhosts caches (SQL administration API)

Use the onmode and h arguments with the **admin()** or **task()** function to Update sqlhosts caches.

Syntax

Usage (UNIX only)

You can use this function only when the database server is in online mode.

This function is equivalent to the **onmode -h** command.

Example

The following example triggers an unconditional reload of the sqlhosts caches:

```
EXECUTE FUNCTION task("onmode", "h", "force");
```


onmode and j arguments: Switch the database server to administration mode (SQL administration API)

Use the onmode and j arguments with the **admin()** or **task()** function to change the database server to administration mode.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"j"--)--;------><
                     '-task--'
```

Usage

When the server is changed to administration mode, all sessions lose their connection to the database server except for sessions of the following users:

- User **informix**
- Users in the **DBSA** group
- Users who are identified in ADMIN_MODE_USERS settings

This function is equivalent to the **onmode -j** command.

Example

The following example changes the server to administration mode:

```
EXECUTE FUNCTION task("onmode","j");
```

Related reference:
onmode -k, -m, -s, -u, -j: Change database server mode

onmode and l arguments: Switch to the next logical log (SQL administration API)

Use the onmode and l arguments with the **admin()** or **task()** function to switch the current logical-log file to the next logical-log file.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"l"--)--;------><
                     '-task--'
```

Usage

This function is equivalent to the **onmode -l** command.

For information on switching to the next logical-log file, see the section on managing logical-log files in the *IBM® Informix® Administrator's Guide*.

Example

The following example moves the logical log out of the **root** chunk

```
SELECT task("onmode", "l") FROM sysmaster:syslogfil
   WHERE chunk = 1 AND sysmaster:bitval(flags,"0x02")>0;
```

Related reference:
onmode -l: Switch the logical-log file

onmode and m arguments: Switch to multi-user mode (SQL administration API)

Use the onmode and m arguments with the **admin()** or **task()** function to change the database server to multi-user mode.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"m"--)--;------><
                    '-task--'
```

Usage

Use this function to bring the database server online from quiescent mode or from administration mode.

This function is equivalent to the **onmode -m** command.

Example

The following example changes the server to multi-user mode:

```
EXECUTE FUNCTION task("onmode","m");
```

Related reference:
onmode -k, -m, -s, -u, -j: Change database server mode

onmode and M arguments: Temporarily change decision-support memory (SQL administration API)

Use the onmode and M arguments with the **admin()** or **task()** function to temporarily change the size of memory available for parallel queries.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                    '-task--'

>--"onmode"--,--"M"--,--"memory_size"--)--;--------------------><
```

Element	Description	Key Considerations
memory_size	The new size limit (in kilobytes) of the maximum amount of memory available for parallel queries.	The maximum value for 32-bit platform is 2 gigabytes. The maximum value for 64-bit platform is 4 gigabytes.

Usage

Use this function to override the limit set by the DS_TOTAL_MEMORY configuration parameter while the database server is online. The new values affect only the current instance of the database server; the values are not recorded in the ONCONFIG file. If you shut down and restart the database server, the values of the parameter revert to the values in the ONCONFIG file.

This function is equivalent to the **onmode -M** command.

Example

The following example sets the size limit for parallel queries to 50 MB:

```
EXECUTE FUNCTION task("onmode","M","50000");
```

Related reference:
onmode -D, -M, -Q, -S: Change decision-support parameters

onmode and n arguments: Unlock resident memory (SQL administration API)

Use the onmode and n arguments with the **admin()** or **task()** function to end forced residency of the resident portion of shared memory.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"n"--)--;------><
                    '-task--'
```

Usage

The RESIDENT configuration parameter must be set to `1` in the ONCONFIG file before you can run this function.

This function does not affect the value of the RESIDENT configuration parameter, the forced-residency parameter in the ONCONFIG file.

This function is equivalent to the **onmode -n** command.

Example

The following example unlocks resident memory:

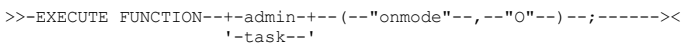
```
EXECUTE FUNCTION task("onmode","n");
```

Related reference:
onmode -n, -r: Change shared-memory residency

onmode and O arguments: Mark a disabled dbspace as down (SQL administration API)

Use the onmode and O arguments with the **admin()** or **task()** function to mark a disabled dbspace as down so that the checkpoint that is being blocked by the disabled dbspace can continue and any blocked threads are released.

Syntax



Usage

This function overrides the WAIT mode of the ONDBSPACEDOWN configuration parameter. Use this command only in the following circumstances:

- ONDBSPACEDOWN is set to WAIT.
- A disabling I/O error occurs that causes the database server to block all updating threads.
- You cannot or do not want to correct the problem that caused the disabling I/O error.
- You want the database server to mark the disabled dbspace as down and continue processing.

This function is equivalent to the **onmode -O** command.

Example

The following example marks disabled dbspaces as down:

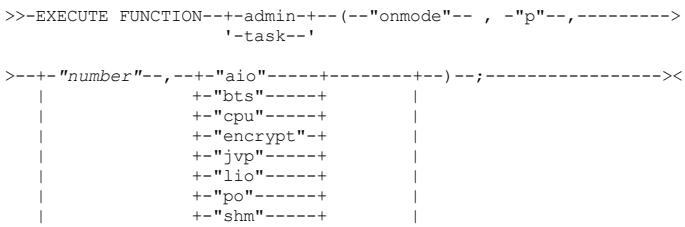
```
EXECUTE FUNCTION task("onmode","O");
```

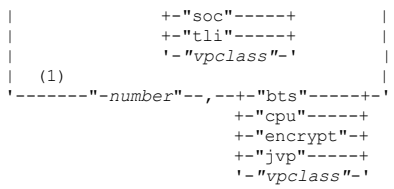
Related reference:
onmode -O: Override ONDBSPACEDOWN WAIT mode

onmode and p arguments: Add or remove virtual processors (SQL administration API)

Use the onmode and p arguments with the **admin()** or **task()** function to dynamically add or remove virtual processors for the current database server session. This function does not update the onconfig file.

Syntax





Notes:

1. UNIX only

Element	Description	Key Considerations
number	The number of virtual processors to add or to remove.	A positive number adds virtual processors. The maximum number of virtual processors you can add depends on the operating system. UNIX: A negative number removes virtual processors. The number of virtual processors to drop cannot exceed the actual number of processors of the specified type.
vpclass	The name of a user-defined virtual processor class.	Windows: The *number* argument must be set to `1` because you can only create one instance of a user-defined virtual processor.

Usage

You can use this function only when the database server is in online mode.

The number of CPU VPs should not exceed the number of physical processors on your system, but no error is issued if they do. The database server uses the number of CPU VPs to allocate resources for parallel database queries (PDQ). If you drop CPU VPs, your queries might run significantly slower. After you change the number of CPU VPs, the **Reinit** field in the output from the **onstat -g mgm** command shows how many queries are waiting for other queries to complete.

See the *IBM® Informix® Performance Guide* for more information about performance implications of the CPU VP class.

For a description of each virtual processor class, see the *IBM Informix Administrator's Guide*.

This function is equivalent to the **onmode -p** command.

Example

The following example adds one CPU virtual processor:

```
EXECUTE FUNCTION task("onmode","p","1","cpu");
```

The following example removes one Java™ virtual processor:

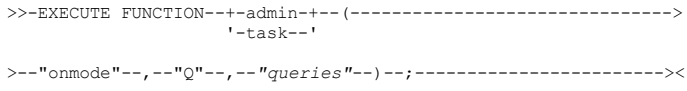
```
EXECUTE FUNCTION task("onmode","p","-1","jvp");
```

Related reference:
onmode -p: Add or drop virtual processors

onmode and Q arguments: Set maximum number for decision-support queries (SQL administration API)

Use the onmode and Q arguments with the **admin()** or **task()** function to change the maximum number of concurrently executing decision-support queries.

Syntax



Element	Description	Key Considerations
queries	The maximum number of concurrently executing parallel queries.	The number must be an unsigned integer from 1 to 8,388,608.

Usage

Use this function to override the limit set by the DS_MAX_QUERIES configuration parameter while the database server is online. The new values affect only the current instance of the database server; the values are not recorded in the ONCONFIG file. If you shut down and restart the database server, the values of the parameter revert to the values in the ONCONFIG file.

For information on parameters used for controlling PDQ, see the *IBM® Informix® Performance Guide*.

This function is equivalent to the **onmode -Q** command.

Example

The following example sets the maximum number of concurrently executing parallel queries to 8:

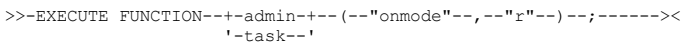
```
EXECUTE FUNCTION task("onmode","Q","8");
```

Related reference:
onmode -D, -M, -Q, -S: Change decision-support parameters

onmode and r arguments: Force residency of shared memory (SQL administration API)

Use the onmode and r arguments with the **admin()** or **task()** function to start forced residency of the resident portion of shared memory.

Syntax



Usage

The RESIDENT configuration parameter must be set to `1` in the ONCONFIG file before you can run this function.

This function does not affect the value of the RESIDENT configuration parameter, the forced-memory parameter in the ONCONFIG file.

This function is equivalent to the **onmode -r** command.

Example

The following example starts forced residency of shared memory:

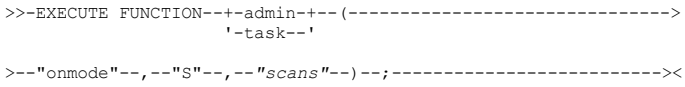
```
EXECUTE FUNCTION task("onmode","r");
```

Related reference:
onmode -n, -r: Change shared-memory residency

onmode and S arguments: Set maximum number of decision-support scans (SQL administration API)

Use the onmode and S arguments with the **admin()** or **task()** function to change the maximum number of concurrently executing decision-support scans for the current session.

Syntax



Element	Description	Key Considerations
scans	The maximum number of concurrently executing parallel scans.	The number must be an unsigned integer from 10 to 1 048 576.

Usage

Use this function to override the limit set by the DS_MAX_SCANS configuration parameter while the database server is online. The new value affects only the current instance of the database server; the values are not recorded in the ONCONFIG file. If you shut down and restart the database server, the value of the parameter reverts to the value in the ONCONFIG file.

For information on parameters used for controlling PDQ, see the *IBM® Informix® Performance Guide*.

This function is equivalent to the **onmode -S** command.

Example

The following example sets the maximum number of concurrently executing parallel scans to 2000:

```
EXECUTE FUNCTION task("onmode","S","2000");
```

Related reference:
onmode -D, -M, -Q, -S: Change decision-support parameters

onmode and W arguments: Reset statement cache attributes (SQL administration API)

Use the onmode and W arguments with the **admin()** or **task()** function to change whether and when a statement can be inserted into the SQL cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"W"--,---------->
                     '-task--'

>--+-"STMT_CACHE_HITS"--,--"hits"-----+--)--;------------------><
   '-"STMT_CACHE_NOLIMIT"--,--"value"-'
```

Element	Description	Key Considerations
hits	The number of hits (references) to a statement before it is fully inserted in the SQL statement cache.	Possible values are: • 0 = Insert all qualified statements and their memory structures in the cache. • 1 or more = Exclude ad hoc queries from entering the cache.
value	Whether statements are inserted in the SQL statement cache.	Possible values are: • 0 = The database server does not insert statements into the cache. • 1 = The database server always inserts statements in the cache.

Usage

Use this function to reset the value of the STMT_CACHE_HITS or STMT_CACHE_NOLIMIT configuration parameter while the database server is online. The new value affects only the current instance of the database server; the value is not recorded in the ONCONFIG file. If you shut down and restart the database server, the value of the parameter reverts to the value in the ONCONFIG file.

If you set the value of STMT_CACHE_HITS equal to 0, the database server inserts all qualified statements and their memory structures in the cache. If the value is greater than 0 and the number of times the SQL statement has been executed is less than the value of STMT_CACHE_HITS, the database server inserts *key-only* entries in the cache. The database server inserts qualified statements in the cache after the specified number of hits has occurred for the statement. The new value of STMT_CACHE_HITS displays in the **#hits** field of the **onstat -g ssc** output.

If none of the queries are shared, set STMT_CACHE_NOLIMIT to 0 to prevent the database server from allocating a large amount of memory for the statement cache.

This function is equivalent to the **onmode -W** command.

Example

The following example prevents ad hoc queries from entering the SQL statement cache:

```
EXECUTE FUNCTION task("onmode","W","STMT_CACHE_HITS","1");
```

Related reference:
onmode -W: Change settings for the SQL statement cache

onmode and wf arguments: Permanently update a configuration parameter (SQL administration API)

Use the onmode and wf arguments with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in the **onconfig** file.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
```

```
                    '-task--'

>--"onmode"--,--"wf"--,--"configuration_parameter_name=new_value"--)--;-><
```

Element	Description	Key Considerations
configuration_parameter_name	The name of a configuration parameter.	The configuration parameter must be one that you can update dynamically. The list of configuration parameters that you can update dynamically is the same as for the **onmode -wf** command.
new_value	The new value or values for the configuration parameter.	The value must be valid for the configuration parameter. The format of the new value must conform exactly to the syntax for that configuration parameter.

Usage

Use this function to permanently update the value of a configuration parameter. The new value takes effect immediately and persists in the ONCONFIG file after the server restarts.

This function is equivalent to the **onmode -wf** command.

Example

The following example sets the value of the DYNAMIC_LOGS configuration parameter to 2 in the **onconfig** file:

```
EXECUTE FUNCTION task("onmode","wf","DYNAMIC_LOGS=2");
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

onmode and wm arguments: Temporarily update a configuration parameter (SQL administration API)

Use the onmode and wm arguments with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in memory.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                    '-task--'

>--"onmode"--,--"wm"--,--"configuration_parameter_name=new_value"--)--;-><
```

Element	Description	Key Considerations
configuration_parameter_name	The name of a configuration parameter.	The configuration parameter that you specify must be one that you can update dynamically. The list of configuration parameters that you can update dynamically is the same as for the **onmode -wf** command.
new_value	The new value or values for the configuration parameter.	The value must be valid for the configuration parameter. The format of the new value must conform exactly to the syntax for that configuration parameter.

Usage

Use this function to temporarily update the value of a configuration parameter that can be dynamically updated. The new value takes effect immediately. The new value is not written to the ONCONFIG file and is lost when the database server is restarted.

This function is equivalent to the **onmode -wm** command.

Example

The following example sets the value of the DYNAMIC_LOGS configuration parameter to 2 for the current session:

```
EXECUTE FUNCTION task("onmode","wm","DYNAMIC_LOGS=2");
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

onmode, wm, and AUTO_LRU_TUNING arguments: Change LRU tuning status (SQL administration API)

Use the onmode, wm, and AUTO_LRU_TUNING arguments with the **admin()** or **task()** function to change the LRU tuning status without updating the onconfig file. .

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onmode"--,--"wm"--,--------->
                     '-task--'

>--+-"AUTO_LRU_TUNING=0-+--"--)--;----------------------------><
   '-"AUTO_LRU_TUNING=1-'
```

Usage

Use the AUTO_LRU_TUNING =1 argument to enable automatic LRU tuning.

Use the AUTO_LRU_TUNING=0 argument to disable automatic LRU tuning .

This function is equivalent to the **onmode -wm AUTO_LRU_TUNING** command.

Example

The following example enables automatic LRU tuning:

```
EXECUTE FUNCTION task("onmode","wm","AUTO_LRU_TUNING=1");
```

Related reference:
onmode -wm: Change LRU tuning status

onmode and Y arguments: Change query plan measurements for a session (SQL administration API)

Use the onmode and Y arguments with the **admin()** or **task()** function to change the output of query plan measurements for an individual session.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"onmode"--,--"Y"--"session_id"--,---------------------------->

>--+-0-----------------------+--)--;--------------------------><
   '-+-2-+--+--------------+-'
     '-1-'  '-,--file_name-'
```

Element	Description	Key Considerations
file_name	The explain output file name.	If the file's absolute path is not included, the example output file is created in the default example output file location. If the file already exists, explain output is appended to it. If a file already exists from the SET EXPLAIN statement, that file is not used until dynamic explain is turned off.
session_id	Identifies the specific session.	None.
-Y	Dynamically change the value of the SET EXPLAIN statement.	None.

Usage

You can use this function to emulate the SET EXPLAIN statement.

The last argument determines if record query measurements, including the plan of the query optimizer, an estimate of the number of rows returned, and the relative cost of the query.

Use the 2 argument to enable the database server to send the query plan to the explain output file.

Use the 1 argument to enable the database server to send the query plan and statistics, to the explain output file. This setting is equivalent to the SET EXPLAIN ON statement for a specific session.

Use the 0 argument to disable the output of query measurements to the explain output file for the current session. This setting is equivalent to the SET EXPLAIN OFF statement.

This function is equivalent to the **onmode -Y** command.

Example

The following example disables the output of query measurements for user session with an ID of 32:

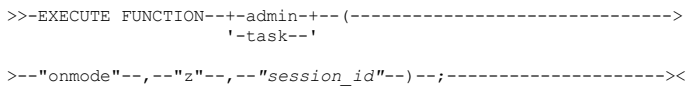
```
EXECUTE FUNCTION task("onmode","Y","32","0");
```

Related reference:
onmode -Y: Dynamically change SET EXPLAIN
Related information:
SET EXPLAIN statement
Using the FILE TO option
Default name and location of the explain output file on UNIX
Default name and location of the output file on Windows
Report that shows the query plan chosen by the optimizer
The explain output file
Query statistics section provides performance debugging information

onmode and z arguments: Terminate a user session (SQL administration API)

Use the onmode and z arguments with the **admin()** or **task()** function to terminate the specified user session.

Syntax



Element	Description	Key Considerations
session_id	The session ID.	The value must be an unsigned integer greater than 0, and must be the session identification number of a currently running session.

Usage

This function is equivalent to the **onmode -z** command.

Example

The following example terminates the user session with an ID of 14:

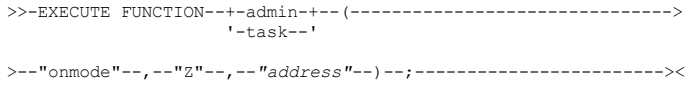
```
EXECUTE FUNCTION task("onmode","z","14");
```

Related reference:
onmode -z: Kill a database server session

onmode and Z arguments: Terminate a distributed transaction (SQL administration API)

Use the onmode and Z arguments with the **admin()** or **task()** function to terminate the specified distributed transaction. Use this function only if communication between the participating database servers has been lost. If applications are performing distributed transactions, terminating one of the distributed transactions can leave your client/server database system in an inconsistent state.

Syntax



Element	Description	Key Considerations

Element	Description	Key Considerations
address	The shared-memory address associated with a distributed transaction.	This must be the address of an ongoing distributed transaction that has exceeded the amount of time that the TXTIMEOUT configuration parameter specifies. The *address* must conform to the operating-system-specific rules for addressing shared-memory. This address is available from **onstat -x** output.

Usage

This function succeeds only if the distributed transaction has exceeded the amount of time that the TXTIMEOUT configuration parameter specifies.

This function is equivalent to the **onmode -Z** command.

Example

The following example terminates a distributed transaction with an address of `0xa509018`:

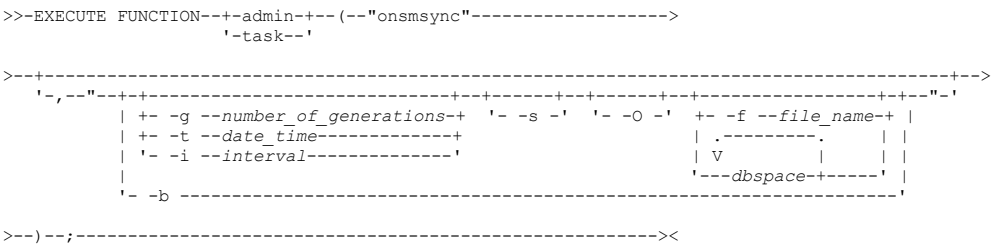
```
EXECUTE FUNCTION task("onmode","Z","0xa509018");
```

Related reference:
onmode -Z: Kill a distributed transaction

onsmsync argument: Synchronize with the storage manager catalog (SQL administration API)

Use the onsmsync argument with the **admin()** or **task()** function to synchronize the **sysutils** database and emergency boot file with the storage manager catalog.

Syntax



Element	Purpose	Key considerations
no options	Synchronizes the **sysutils** database and emergency boot file with the storage-manager catalog	None.
-b	Regenerates both the emergency boot file (ixbar.*servernum*) and the **sysutils** database from each other.	If the ixbar file is empty or does not exist, **onsmsync -b** recreates the ixbar file and populates it from the **syutils** tables. If the ixbar is not empty and contains object data, **onsmsync -b** updates the **sysutils** database and the ixbar file so that they are in sync. If the ixbar file has entries and the **sysutils** database has been rebuilt, but is empty because it does not contain data, the **onsmsync -b** option recreates the **sysutils** data from the ixbar file. The -b element is not used with the other **onsmsync** options. Additionally, it does not synchronize with the storage manager.
dbspace	Specifies the storage space or storage spaces to expire	If you enter more than one storage space, use a space to separate the names.
-f *file_name*	Specifies the path name of a file that contains a list of storage spaces to expire	Use this option to avoid entering a long list of storage spaces. The file name can be any valid UNIX or Windows file name.
-g *number_of_generations*	Retains a certain number of versions of each level-0 backup	The latest generation of backups are retained and all earlier ones are expired.
-i *interval*	Expires all backups older than some period of time	Retains backups younger than this interval. Backups older than interval are not expired if they are needed to restore from other backups after that interval. Use the ANSI or GLS format for the *interval*: YYYY-MM or DD HH:MM:SS
-s	Skips backups that the storage manager has expired	Use this option to skip synchronizing objects that are already expired from the storage manager. The object expiration will be based on other arguments if the -s option is provided.

Element	Purpose	Key considerations
-O	Enforces expiration policy strictly	If used with the -t, -g, or -i option, expires all levels of a backup, even if some of them are needed to restore from a backup that occurred after the expiration date. The -O option does not affect logical-log expiration. See Expire all backups.
-t *date_time*	Expires all backups before a particular date and time	Retains backups younger than this *datetime*. Backups older than *datetime* are not expired if they are needed to restore from other backups after that *datetime*. Use the ANSI or GLS_DATETIME format for *datetime*.

Usage

This function invokes the **onsmsync** utility to synchronize the **sysutils** database and emergency boot file with the storage manager catalog.

Example

The following example invokes the **onsmsync** utility and specifies that number of backups retained is 1 and all earlier backup versions are expired:

```
EXECUTE FUNCTION task("onsmsync", "-g 1");
```

Related information:
The onsmsync utility

onstat argument: Monitor the database server (SQL administration API)

Use the onstat argument with the **admin()** or **task()** function to monitor the database server.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"onstat"--,--"--option_name--"--)-- ;-><
                     '-task--'
```

Element	Description	Key Considerations
option_name	The **onstat** command option.	The option must include a hyphen and conform to **onstat** option syntax. For **onstat** options, see The onstat utility.

Usage

Use these commands to display the same information about the database server as running the **onstat** utility commands.

Example

The following example runs the **onstat -g ses** command:

```
EXECUTE FUNCTION task("onstat","-g ses");
```

Related reference:
The onstat utility

ontape archive argument: Backup the data on your database (SQL administration API)

Use the ontape archive argument with the **admin()** or **task()** function to create a backup of your database data.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"ontape archive-------------->
                     '-task--'

   .-file------.
>--+-----------+--+-------------+--"--,--"location"-------------->
   +-directory-+  |       .-0-. |
   '-tape------'  '-level-+-1-+-'
                          '-2-'
```

```
>--+---------------+--)--;-------------------------------------><
   '-,"block_size"-'
```

Element	Description	Key Considerations
location	The path to the file or directory or tape device.	
block_size	The block size, in KB, of the device to which ontape writes during a storage-space backup.	The default block size is 512 KB.

Usage

This function invokes the ontape utility to create a backup.

There are three devices that you can choose from for the location of the backup:

file
: An existing file. This is the default value.

directory or dir
: An existing directory path specified by *location*.

tape
: An existing tape device.

Examples

This function creates a level 0 archive in the directory path /local/informix/backup/:

```
EXECUTE FUNCTION task("ontape archive","/local/informix/backup/");
```

This function creates a level 0 archive in the directory path /local/informix/backup/ with a block size of 256 KB:

```
EXECUTE FUNCTION task("ontape archive directory level 0",
"/local/informix/backup/","256");
```

Related information:
Back up with ontape

print error argument: Print an error message (SQL administration API)

Use the print error argument with the **admin()** or **task()** function to print the message associated with the specified error number.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"print error"--,--"error_number"--)--;----------------------><
```

Element	Description	Key Considerations
error_number	The error number, without a minus sign.	The *error_number* must be an existing error number.

Usage

This function is equivalent to the **finderr** utility.

Example

The following example prints the message text for the error number -105:

```
EXECUTE FUNCTION task("print error","105");
(expression)  ISAM error:  bad isam file format.
```


print file info argument: Display directory or file information (SQL administration API)

Use the print file info argument with the **admin()** or **task()** function to display information about a directory or a file

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                    '-task--'

>--(--"print file info"--,--"--file_path--"--)--;-------------><
```

Element	Purpose	Key considerations
file_path	The path to the directory or file	

Example: File information

The following example shows the argument you would use to print information about the `x` file that is in the /tmp directory:

```
execute function task("print file info","/tmp/x");
```

The following information is returned:

```
(expression)  File name           = /tmp/x
              Is File             = 1
              Is Directory        = 0
              Is Raw Device       = 0
              Is Block Device     = 0
              Is Pipe             = 0
              File Size           = 554
              Last Access Time    = 11/29/2010 21:55:02
              Last Modified Time  = 11/29/2010 21:51:45
              Status Change Time  = 11/29/2010 21:51:45
              User Id             = 200
              Group id            = 102
              File Flags          = 33206
```

Example: Directory information

The following example shows the argument that you would use to print information about the `/tmp` directory:

```
execute function task("print file info","/tmp");
```

The following information is returned:

```
(expression)  File name           = /tmp
              Is File             = 0
              Is Directory        = 1
              Is Raw Device       = 0
              Is Block Device     = 0
              Is Pipe             = 0
              File Size           = 32768
              Last Access Time    = 12/06/2010 11:53:00
              Last Modified Time  = 12/06/2010 12:05:53
              Status Change Time  = 12/06/2010 12:05:53
              User Id             = 0
              Group id            = 0
              File Flags          = 17407
```


print partition argument: Print partition information (SQL administration API)

Use the print partition argument with the **admin()** or **task()** function to print the headers of a specified partition.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                    '-task--'

>--(--+-"print partition"------+--,--"partition_number"--)--;--><
      '-"print partition full"-'
```

Element	Description	Key Considerations
partition_number	The partition number.	Find the partition numbers in the **partnum** column of the **systables** system catalog table.

Usage

Use this function to print a tblspace report for the specified partition.

Run this function with the full argument to include index-specific information and page-allocation information by page type for dbspaces.

This function with the print partition argument is equivalent to the **oncheck -pt** command.

This function with the print partition full argument is equivalent to the **oncheck -pT** command.

Example

The following example prints the headers for a partition with a number of `1048611`:

```
EXECUTE FUNCTION task("print partition","1048611");
```

Related reference:
oncheck -pt and -pT: Display tblspaces for a Table or Fragment

rename space argument: Rename a storage space (SQL administration API)

Use the rename space argument with the **admin()** or **task()** function to rename a dbspace, blobspace, sbspace, or extspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"rename space"--,--"space_name"--,--"new_name"--)--;-------><
```

Element	Description	Key Considerations
new_name	The new name of the space.	
space_name	The name of the dbspace, blobspace, sbspace, or extspace that you want to rename.	

Usage

This function is equivalent to the **onspaces -ren** command.

Example

The following example renames a dbspace named **dbsp1** to **dbsp2**:

```
EXECUTE FUNCTION task("rename space","dbsp1","dbsp2");
```

Related reference:
onspaces -ren: Rename a dbspace, blobspace, sbspace, or extspace

reset config argument: Revert configuration parameter value (SQL administration API)

Use the reset config argument with the **admin()** or **task()** function to revert the value of a dynamically updatable configuration parameter to its value in the onconfig file. Dynamically updatable configuration parameters are those parameters that you can change for a session with an **onmode** or SQL administration API command.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--reset config--"----------->
                     '-task--'

>--,--"configuration_parameter_name"--)--;--------------------><
```

Table 1. reset config command elements

Element	Description	Key Considerations
configuration_parameter_name	The name of the configuration parameter with the value that you want to revert.	

Usage

The reset config argument reverts the value of the configuration parameter to the last saved value in the onconfig file, even if the value was changed after the database server started.

Example

The following command reverts the value of the DYNAMIC_LOGS configuration parameter to the value in the onconfig file.

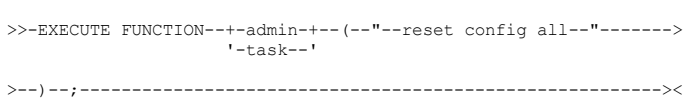
```
EXECUTE FUNCTION task("reset config","DYNAMIC_LOGS");
```

Related tasks:
Modifying the onconfig file
Related reference:
reset config all argument: Revert all dynamically updatable configuration parameter values (SQL administration API)

reset config all argument: Revert all dynamically updatable configuration parameter values (SQL administration API)

Use the reset config all argument with the **admin()** or **task()** function to revert the values of all dynamically updatable configuration parameter to their values in the onconfig file. Dynamically updatable configuration parameters are those parameters that you can change for a session with an **onmode** or SQL administration API command.

Syntax



Usage

The reset config all argument reverts the values of all dynamically updatable configuration parameters to the last saved values in the onconfig file, even if the values were changed after the database server started.

Example

The following command reverts the value of all dynamically tunable configuration parameters.

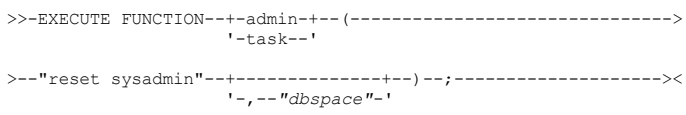
```
EXECUTE FUNCTION task("reset config all");
```

Related tasks:
Modifying the onconfig file
Related reference:
reset config argument: Revert configuration parameter value (SQL administration API)

reset sysadmin argument: Move the sysadmin database (SQL administration API)

Use the reset sysadmin argument with the **admin()** or **task()** function to move the **sysadmin** database to the specified dbspace. Moving the **sysadmin** database resets the database back to the original state when it was first created; all data, **command history**, and results tables are lost. Only built-in tasks, sensors, and thresholds remain in the **sysadmin** tables.

Syntax



Element	Description	Key Considerations
dbspace	The name of the dbspace.	

Usage

This function has no equivalent utility command.

If you specify no *dbspace* as the last argument, this command drops the **sysadmin** database, and then re-creates it in the rootdbs. All **ph_*** table and **command_history** rows are deleted, and all results tables are dropped.

Examples

The following example drops the existing **sysadmin** database and creates a new **sysadmin** database in a dbspace named **dbsp1**:

```
EXECUTE FUNCTION task("reset sysadmin","dbs1");
```

The next example drops the **sysadmin** database, and then re-creates it in the rootdbs.

```
EXECUTE FUNCTION admin("reset sysadmin");
```

Except for the built-in tasks, sensors, and thresholds, all data rows are deleted from the **ph_** tables and all results tables are dropped from **sysadmin** by this function call. The **command_history** table has no rows after the function completes execution.

Related concepts:
The sysadmin Database

restart listen argument: Stop and start a listen thread dynamically (SQL administration API)

Use the restart listen argument with the **admin()** or **task()** function to stop and then start an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"restart listen"--,--"server_name"--)--;--------------------><
```

Element	Description	Key Considerations
server_name	The name of the database server for which you want to stop and restart a listen thread.	

Usage

The definition of the listen thread must exist in the **sqlhosts** file.

If necessary, before you restart a listen thread, revise the **sqlhosts** entry. For example, if a running listen thread is bound to port 7777, you can change the port in the **sqlhosts** file, and then restart the thread.

This function is equivalent to the **onmode -P restart** *server_name* command.

This function does not update the **sqlhosts** file.

Example

The following command stops and then starts a listen thread for a server named **ids_serv1**:

```
EXECUTE FUNCTION task("restart listen","ids_serv1");
```

Related reference:
onmode -P: Start, stop, or restart a listen thread dynamically
start listen argument: Start a listen thread dynamically (SQL administration API)
stop listen argument: Stop a listen thread dynamically (SQL administration API)

revoke admin argument: Revoke privileges to run SQL administration API commands

Use the revoke admin argument with the **admin()** or **task()** function to revoke privileges to run SQL administration API commands.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--revoke admin--"----------->
                     '-task--'

>--,--"--user_name--"--+-------------------------+--)--;------><
                       '-,--"--privilege_group--"-'
```

Element	Description	Key Considerations
user_name	The name of the user from which to revoke privileges.	
privilege_group	The name of the privilege group.	See SQL administration API portal: Arguments by privilege groups for a list of privilege groups.

Usage

Only user **informix**, or a user with ADMIN or GRANT permissions for SQL administration API commands, can use the **revoke admin** argument.

Use the **revoke admin** argument to revoke the privilege to run SQL administration API commands from individual users. You can revoke the privilege for a specific privilege group or revoke all privileges.

Examples

The following command revokes the privilege for running backup and restore SQL administration commands from the user Bob:

```
EXECUTE FUNCTION task("revoke admin", "Bob", "BAR");
```

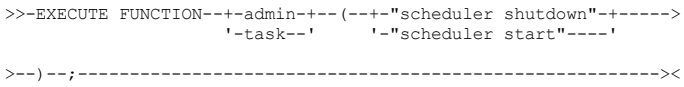
The following command revokes all privileges for running any SQL administration commands from the user Bob:

```
EXECUTE FUNCTION task("revoke admin", "Bob");
```


scheduler argument: Stop or start the scheduler (SQL administration API)

Use the scheduler argument with the **admin()** or **task()** function to start or stop the scheduler.

Syntax



Usage

Use the scheduler shutdown argument to stop the scheduler and deallocate its resources.

Use the scheduler start argument to start the scheduler.

This function has no equivalent utility command.

You can view the status of the scheduler threads with the **onstat -g dbc** command.

Example

The following example starts the scheduler after it has been shut down:

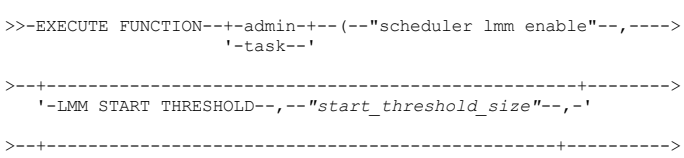
```
EXECUTE FUNCTION task("scheduler start");
```

Related reference:
onstat -g dbc command: Print dbScheduler and dbWorker thread statistics

scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)

Use the scheduler lmm enable argument with the **admin()** or **task()** function to start automatic low memory management and to update low memory threshold settings.

Syntax



```
   '-LMM STOP THRESHOLD--,--"stop_threshold_size"--,-'

>--+-------------------------------------------------+--)--;----->
   '-LMM IDLE TIME--,--"minimum_amount_of_time"--,-'
```

Element	Description	Key Considerations
start_threshold_size	The amount of free memory that you want the database server to maintain. If the amount of memory falls below the *start_threshold_size*, the server automatically frees memory and terminates applications.	The value can be expressed as either a percentage of the value of the SHMTOTAL configuration parameter or as a specific amount. If the value is less than 50, it is considered a percentage. The resulting value of the input parameter must be more than 5 MB and less than 95 MB. The default value is 5 MB. There must be a minimum 5 MB difference between the LMM START THRESHOLD and LMM STOP THRESHOLD values
stop_threshold_size	The amount of free memory that you want the database server to have, before the server stops automatically freeing memory and terminating applications.	The value can be expressed as either a percentage of the value of the SHMTOTAL configuration parameter or as a specific amount. If the value is less than 50, it is considered a percentage. The resulting value of the input parameter must be more than 10 MB and less than 100 MB. The value must also be at least 5 MB more than the LMM START THRESHOLD. The default value is 10 MB.
minimum_amount_of_time	The amount of time in seconds that defines a session as idle	The value must be between 1 and 86400. The default value is 300 seconds.

Usage

You use the scheduler lmm disable argument with the **admin()** or **task()** function to stop the current and subsequent low memory management processes in a primary or standard database server. When low memory management is triggered, the database server performs these tasks, in order:

1. The database server terminates sessions starting one at a time from the session with largest amount of idle time and continuing as necessary to the session with smallest amount of idle time that exceeds the amount specified in the LMM IDLE TIME setting. The server stops terminating sessions when the LMM STOP THRESHOLD is reached.
2. The database server terminates sessions starting with the session using the most memory and continuing as necessary to the session using the smallest amount of memory until the LMM STOP THRESHOLD is reached.
3. The database server performs memory reconfiguration by setting the VP_MEMORY_CACHE configuration parameter to `0` and running the **onmode -F** command to free unused shared memory segments.

When the low memory management operations are complete, the low memory manager returns to monitoring mode and restores the memory configuration of the database server by setting the VP_MEMORY_CACHE configuration parameter back to its original value.

The database server stores automatic low memory management settings in the **ph_threshold** table.

You can view low memory management settings and recent activity with the **onstat -g lmm** command.

Attention: If you enable automatic low memory management and configure the database server to use a percentage of the value specified in the SHMTOTAL configuration parameter for the start and stop thresholds, use caution when changing the value of the SHMTOTAL configuration parameter. Changing the value of the SHMTOTAL configuration parameter value can cause the configuration of automatic low memory management to become invalid, forcing Informix® to use the default settings.

Example of setting low memory management threshold settings

The following example specifies that when the database server has 10 MB or less of free memory, the server will start automatic low memory management to stop applications and to free memory. The example also specifies that a session is considered idle if it has not run for 300 seconds, and the example specifies that the server will stop automatic low memory management when the server has 20 MB or more of free memory.

```
EXECUTE FUNCTION task("scheduler lmm enable",
        "LMM START THRESHOLD", "10MB",
        "LMM STOP THRESHOLD", "20MB",
        "LMM IDLE TIME", "300");
```

Example of the SHMTOTAL configuration parameter impacting low memory management threshold settings

Suppose you set the SHMTOTAL configuration parameter to `1000000` (1000 MB or 1 GB), the LMM START THRESHOLD to `2`, and the LMM STOP THRESHOLD to `3`. Because any value that is less than 50 is a percentage of the value of SHMTOTAL, the actual LMM START THRESHOLD is 20000 (20 MB) and the actual LMM STOP THRESHOLD is 30000 (30 MB).

The database server begins managing low memory when the remaining free memory is 20 MB or less and stop managing memory when the amount of free memory is 30 MB or greater.

Suppose you decide to change the value of the SHMTOTAL configuration parameter because you know now that you don't need as much memory and you want memory to be available to the operating system. You set the value of SHMTOTAL to `250000` (250 MB). This changes the actual LMM START THRESHOLD to 5000 (5 MB) and the LMM STOP THRESHOLD to 7500 (7.5 MB). The LMM STOP THRESHOLD is now invalid because there must be a minimum 5 MB difference between the LMM START THRESHOLD and LMM STOP THRESHOLD values. The LMM STOP THRESHOLD value must also be at least 10 MB.

You might have decided that a 10 MB difference is the right amount for your system. But at 5 MB, the database server could spend too much time spent on low memory management processes and this could cause performance problems.

Related reference:
scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)
onstat -g lmm command: Print low memory management information
LOW_MEMORY_MGR configuration parameter
SHMTOTAL configuration parameter
VP_MEMORY_CACHE_KB configuration parameter
Related information:
Configure the server response when memory is critically low

scheduler lmm disable argument: Stop automatic low memory management (SQL administration API)

Use the scheduler lmm disable argument with the **admin()** or **task()** function to stop the current and subsequent invocations of automatic low memory management.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--"scheduler lmm disable"--,--->
                     '-task--'

>--)--;-------------------------------------------------------><
```

Usage

If automatic low memory management is enabled, you can disable it by specifying:

```
EXECUTE FUNCTION task("scheduler lmm disable");
```

You use the scheduler lmm enable argument with the **admin()** or **task()** function to start automatic low memory management and update threshold settings.

You can view information about automatic low memory management settings and recent activity with the **onstat -g lmm** command.

Related reference:
scheduler lmm enable argument: Specify automatic low memory management settings (SQL administration API)
onstat -g lmm command: Print low memory management information
LOW_MEMORY_MGR configuration parameter

set chunk argument: Change the status of a chunk (SQL administration API)

Use the set chunk argument with the **admin()** or **task()** function to change the status of a blobspace, dbspace, or sbspace to online or offline.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"set chunk offline"-+------>
                     '-task--'     '-"set chunk online"--'

>--,--"space_name"--,--"path_name"--,--"offset"--)--;---------><
```

Element	Description	Key Considerations
space_name	The name of the blobspace, dbspace, or sbspace.	
path_name	The disk partition or unbuffered device of the chunk.	
offset	The offset, in kilobytes, into the disk partition or unbuffered device to reach the chunk.	See admin() and task() Argument Size Specifications.

Usage

The chunk must be in a mirrored pair, or a non-primary chunk within a noncritical dbspace.

Use the set chunk offline argument to change the status of the chunk to offline.

Use the set chunk online argument to change the status of the chunk to online.

This function is equivalent to the **onspaces -s** command.

Example

The following example changes the status of a chunk to online:

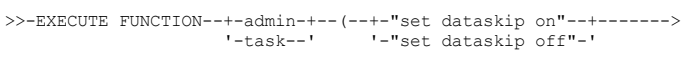
```
EXECUTE FUNCTION task("set chunk online","dbs1","/dev/raw_dev1","0");
     Database selected.

(expression)  Chunk status successfully changed.
              Chunk number 2 "/dev/raw_dev1" -- Online

1 row(s) retrieved.
```

Related reference:
onspaces -s: Change status of a mirrored chunk

set dataskip argument: Start or stop skipping a dbspace (SQL administration API)

Use the set dataskip argument with the **admin()** or **task()** function to specify whether the database server skips a dbspace that is unavailable during the processing of a transaction.

Syntax



Element	Description	Key Considerations
dbspace	The name of the dbspace to begin or to stop skipping.	

Usage

Run this function to update the value of the DATASKIP configuration parameter, which specifies whether the database server skips a dbspace that is unavailable (for example, due to a media failure) in the course of processing a transaction.

Use the set dataskip on argument to begin skipping the specified dbspace when it is down.

Use the set dataskip off argument to stop skipping the specified dbspace.

This function is equivalent to the **onspaces -f** command.

Example

The following example skips the dbspace named **dbsp1** if it is down:

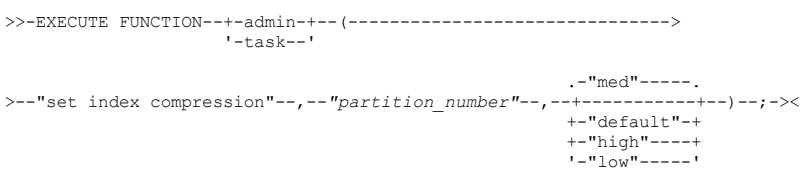
```
EXECUTE FUNCTION task("set dataskip on","dbsp1");
```

Related reference:
onspaces -f: Specify DATASKIP parameter
DATASKIP Configuration Parameter

set index compression argument: Change index page compression (SQL administration API)

Use the set index compression argument with the **admin()** or **task()** function to modify the level at which two partially used index pages are merged.

Syntax



Element	Description	Key Considerations
partition_number	The partition number.	Find the partition numbers in the **partnum** column of the **systables** system catalog table.

Usage

Use this function to adjust index page compression. The pages are merged if the data on those pages totals a set level. To optimize space and transaction processing, you can lower the compression level if your indexes grow quickly. You can increase the level if your indexes have few delete and insert operations or if batch updates are performed.

Use the low argument if you expect an index to grow quickly with frequent splits.

Use the med or default argument if an index has moderate growth or changes.

Use the high argument if an index is 90 percent or more read-only or does not have many changes.

This function is equivalent to the **onmode -C** command and the compression option of the BTSCANNER configuration parameter.

Example

The following example sets index compression for a partition to high:

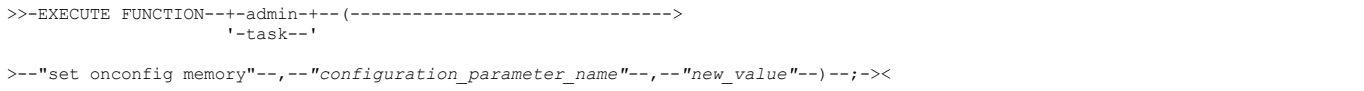
```
EXECUTE FUNCTION task("set index compression","1048611","high");
```

Related reference:
onmode -C: Control the B-tree scanner

set onconfig memory argument: Temporarily change a configuration parameter (SQL administration API)

Use the set onconfig memory argument with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in memory.

Syntax



Element	Description	Key Considerations
configuration_parameter_name	The name of a configuration parameter.	The configuration parameter must be one that you can update dynamically. The list of configuration parameters that you can update dynamically is the same as for the **onmode -wf** command.
new_value	The new value or values for the configuration parameter.	The new value or values must be valid for the configuration parameter. The format of the new value must conform exactly to the syntax for that configuration parameter.

Usage

Use this function to temporarily update the value of a configuration parameter that can be dynamically updated. The new value takes affect immediately. The new value is not written to the **onconfig** file and is lost when the database server is restarted.

This function is equivalent to the **onmode -wm** command.

Example

The following example sets the value of the DYNAMIC_LOGS configuration parameter to 2 for the current session:

```
EXECUTE FUNCTION task("set onconfig memory","DYNAMIC_LOGS","2");
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

set onconfig permanent argument: Permanently change a configuration parameter (SQL administration API)

Use the set onconfig permanent argument with the **admin()** or **task()** function to dynamically update the value of a configuration parameter in the **onconfig** file.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"set onconfig permanent"--,--"configuration_parameter_name"--,--"new_value"--)--;-><
```

Element	Description	Key Considerations
configuration_parameter_name	The name of a configuration parameter.	The configuration parameter must be one that you can update dynamically. The list of configuration parameters that you can update dynamically is the same as for the **onmode -wf** command.
new_value	The new value or values for the configuration parameter.	The new value or values must be valid for the configuration parameter. The format of the new value must conform exactly to the syntax for that configuration parameter.

Usage

Use this function to permanently update the value of a configuration parameter. The new value takes affect immediately and persists in the **onconfig** file after the server restarts.

This function is equivalent to the **onmode -wf** command.

Example

The following example sets the value of the DYNAMIC_LOGS configuration parameter to 2 in the **onconfig** file:

```
EXECUTE FUNCTION task("set onconfig permanent","DYNAMIC_LOGS","2");
```

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

set sbspace accesstime argument: Control access time tracking (SQL administration API)

Use the set sbspace accesstime argument with the **admin()** or **task()** function to start or stop tracking the time of access for all smart large objects stored in the sbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--+-"set sbspace accesstime off"-+--,--"sbspace"--)--;-------><
   '-"set sbspace accesstime on"--'
```

Element	Description	Key Considerations
sbspace	The name of the sbspace.	

Usage

Use the set sbspace accesstime off argument to turn off tracking of access times.

Use the set sbspace accesstime on argument to turn on tracking of access times for all smart large objects stored in the sbspace.

This function is equivalent to the **onspaces -ch** command.

Example

The following example turns off tracking of access times for the sbspace named **sbsp1**:

```
EXECUTE FUNCTION task("set sbspace accesstime off","sbsp1");
```

Related reference:
onspaces -ch: Change sbspace default specifications
create sbspace with accesstime argument: Create an sbspace that tracks access time (SQL administration API)

set sbspace avg_lo_size argument: Set the average size of smart large objects (SQL administration API)

Use the set sbspace avg_lo_size argument with the **admin()** or **task()** function to specify an expected average size of the smart large objects in the specified sbspace so that the database server can calculate the size of the metadata area.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"set sbspace avg_lo_size"--,--"sbspace"--,--"average_size"--->

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
sbspace	The name of the sbspace.	
average_size	The average size, in kilobytes, of the smart large object stored in the sbspace.	Windows: 4 to 2**31 UNIX: 2 to 2**31

Usage

This function is equivalent to the **onspaces -ch** command.

Example

The following example sets the expected average size of smart large objects in the sbspace named **sbsp1** to 8 KB:

```
EXECUTE FUNCTION task("set sbspace avg_lo_size","sbsp1","8");
```

Related reference:
onspaces -ch: Change sbspace default specifications

set sbspace logging argument: Change the logging of an sbspace (SQL administration API)

Use the set sbspace logging argument with the **admin()** or **task()** function to specify whether the database server logs changes to the user data area of the sbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--+-"set sbspace logging on"--+--,--"sbspace"--)--;----------><
   '-"set sbspace logging off"-'
```

Element	Description	Key Considerations
sbspace	The name of the sbspace.	

Usage

Use the set sbspace logging on argument to log changes to the user data area of the sbspace.

Use the set sbspace logging off argument to not log changes to the user data area of the sbspace.

This function is equivalent to the **onspaces -ch** command.

Example

The following example starts sbspace logging for an sbspace named **sbsp1**:

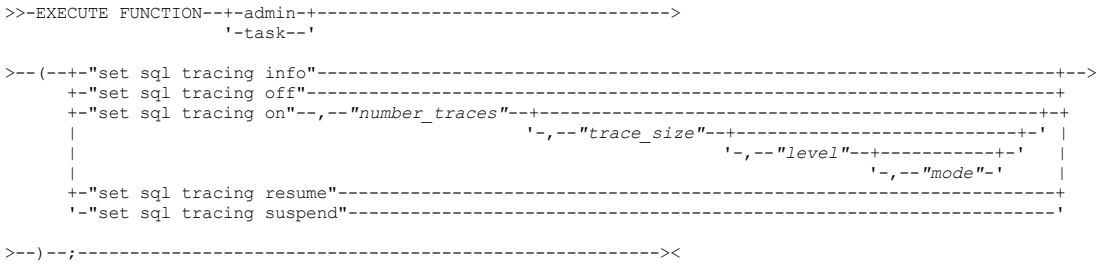
```
EXECUTE FUNCTION task("set sbspace logging on","sbsp1");
```

Related reference:
onspaces -ch: Change sbspace default specifications

set sql tracing argument: Set global SQL tracing (SQL administration API)

Use the set sql tracing argument with the **admin()** or **task()** function to set global SQL tracing.

Syntax



Element	Description	Key Considerations
level	The tracing level. The default is low.	Possible values are: • low • med • high
mode	Whether all or selected users are traced.	Possible modes are: • global • user
number_traces	The number of SQL statements to trace. The default value is 1000.	
trace_size	The number of KB for the size of the trace buffer. If this buffer size is exceeded, the database server discards saved data. The default size is 2 KB.	

Usage

Use this function to reset the value of the SQLTRACE configuration parameter.

Use the set sql tracing info argument to display the state of global SQL tracing.

Use the set sql tracing off argument to turn off global SQL tracing.

Use the set sql tracing on argument to turn on global SQL tracing. Optionally specify the tracing level and mode or change the size of the trace buffer.

- Use the low argument to capture statement statistics, statement text, and statement iterators.
- Use the med argument to capture all of the information included in low-level tracing, plus table names, the database name, and stored procedure stacks.
- Use the high argument to capture all of the information included in medium-level tracing, plus host variables.
- Use the global argument to enable tracing for all users.
- Use the user argument to enable tracing for those users who have tracing enabled by the set sql tracing user argument.

Use the set sql tracing resume argument to restart SQL tracing when it is suspended.

Use the set sql tracing suspend argument to pause SQL tracing without deallocating any resources.

Example

The following example starts a high level of global tracing for 1500 SQL statements into a 4 KB trace buffer:

```
EXECUTE FUNCTION task("set sql tracing on","1500","4","high","global");
```

The following example pauses SQL tracing:

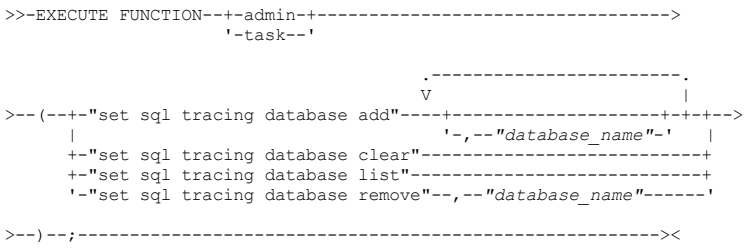
```
EXECUTE FUNCTION task("set sql tracing suspend");
```

Related reference:
SQLTRACE configuration parameter
onstat -g spf: Print prepared statement profiles

set sql tracing database argument: Change database tracing (SQL administration API)

Use the set sql tracing database argument with the **admin()** or **task()** function to start or stop tracing for a database, or list which databases are being traced.

Syntax



Element	Description	Key Considerations
database_name	The name of the database.	Specify one database name.

Usage

Use the set sql tracing database add argument to specify tracing for one or more databases, rather than for all databases. The default is all databases. Specify up to six arguments in a single **admin()** or **task()** function. The maximum number of database names that can be set is 16.

Use the set sql tracing database clear argument to clear all databases from the list of databases being traced. Returns tracing back to the default of all databases.

Use the set sql tracing database list argument to list the databases that are being traced.

Use the set sql tracing database remove argument to remove a single database from the list of databases being traced.

When you use the set sql tracing database argument, you can specify only the name of one database. While you can have a maximum of 16 database names, you must specify each additional database name in separate function calls. Each time you call the function, the function adds another database to the list, until the list contains 16 databases.

Example

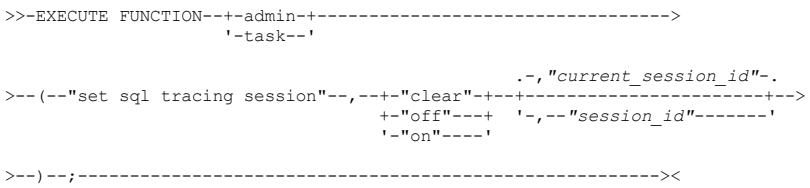
The following example sets SQL tracing for three databases with the names **db1**, **db2** and **db3**:

```
EXECUTE FUNCTION task("set sql tracing database add","db1");
EXECUTE FUNCTION task("set sql tracing database add","db2");
EXECUTE FUNCTION task("set sql tracing database add","db3");
```


set sql tracing session argument: Control tracing for a session (SQL administration API)

Use the set sql tracing session argument with the **admin()** or **task()** function to change SQL tracing for the current session.

Syntax



Element	Description	Key Considerations
current_session_id	The ID of the current session. This is the default session ID.	
session_id	The ID of the session to which this command applies.	

Usage

Use the clear argument to clear any global tracing overrides. The session will conform to the global tracing policy.

Use the off argument to turn off tracing for the session, even if the global tracing policy is set to enable tracing.

Use the on argument to turn on tracing for the session, even if the global tracing policy is set to disable tracing.

Example

The following example stops tracing for the current session:

```
EXECUTE FUNCTION task("set sql tracing session","off");
```


set sql tracing user argument: Control tracing for users (SQL administration API)

Use the set sql tracing user argument with the **admin()** or **task()** function to change SQL tracing for users.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                     '-task--'

>--(--+-"set sql tracing user add"--,--"user_name"----+--)--;--><
      +-"set sql tracing user clear"-------------------+
      +-"set sql tracing user list"--------------------+
      '-"set sql tracing user remove"--,--"user_name"-'
```

Element	Description	Key Considerations
user_name	The name of the user.	

Usage

Use the set sql tracing user add argument to specify tracing for a specific user.

Use the set sql tracing user clear argument to remove all users from the tracing list.

Use the set sql tracing user list argument to list the users that are being traced.

Use the set sql tracing user remove argument to remove a single user from the list of users being traced.

Example

The following example stops tracing SQL statements for the user named **fred**:

```
EXECUTE FUNCTION task("set sql tracing user remove","fred");
```


set sql user tracing argument: Set global SQL tracing for a user session (SQL administration API)

Use the set sql user tracing argument with the **admin()** or **task()** function to set the mode of global SQL tracing for a specified user session.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+----------------------------------->
                     '-task--'

>--(--+-"set sql user tracing clear"-+--,--"session_id"--)--;--><
      +-"set sql user tracing off"---+
      '-"set sql user tracing on"----'
```

Element	Description	Key Considerations
session_id	The ID for the session.	

Usage

Use the set sql user tracing clear to clear user tracing flags for the specified user session so that it adheres to the global tracing policy.

Use the set sql user tracing off to disable SQL tracing for a user session even if the global mode is ON.

Use the set sql user tracing on to enable user SQL tracing for a user session. Even if the global tracing mode is OFF, SQL statements for this user session are traced.

Example

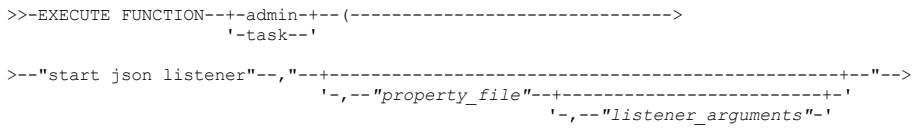
The following example starts tracing for the session with the ID of `18`:

```
EXECUTE FUNCTION task("set sql user tracing on","18");
```


start json listener argument: Start the MongoDB API wire listener (deprecated)

Use the start json listener argument with the **admin()** or **task()** function to start the MongoDB API wire listener.

Syntax



Element	Description	Key Considerations
property_file	The name of the wire listener configuration file to use instead of the default.	The *property_file* is optional. The default wire listener configuration file is in $INFORMIXDIR/etc/jsonListener.properties.
listener_arguments	The command line argument to pass to the wire listener.	

Usage

The start json listener argument starts the MongoDB API wire listener.

Example

In this example, the MongoDB API wire listener is started by using the mycustom.properties file instead of the default jsonListener.properties file:

```
EXECUTE FUNCTION task("start json listener", "mycustom.properties");
```

In this example, the MongoDB API wire listener is started by using mycustom.properties file instead of the default jsonListener.properties, the port is specified as 27018, and the logging level is set as debug:

```
EXECUTE FUNCTION task("start json listener", "mycustom.properties",
   "-port 27018 -loglevel debug");
```

Related information:
Starting the wire listener
Wire listener command line options

start listen argument: Start a listen thread dynamically (SQL administration API)

Use the start listen argument with the **admin()** or **task()** function to start an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.

Syntax

Element	Description	Key Considerations
server_name	The name of the database server for which you want to start a listen thread.	

Usage

The definition of the listen thread must exist in the **sqlhosts** file for the server. If the definition of the listen thread does not exist in the **sqlhosts** file, you must add it before you can start the listen thread dynamically. For information on adding listen threads, see the *IBM® Informix® Administrator's Guide*.

This function does not update the **sqlhosts** file.

This function is equivalent to the **onmode -P start** *server_name* command.

Example

The following command starts a new listen thread for a server named **ids_serv2**:

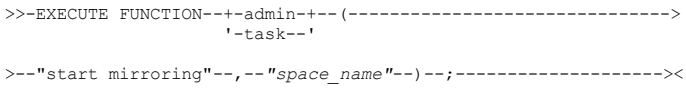
```
EXECUTE FUNCTION task("start listen","ids_serv2");
```

Related reference:
onmode -P: Start, stop, or restart a listen thread dynamically
stop listen argument: Stop a listen thread dynamically (SQL administration API)
restart listen argument: Stop and start a listen thread dynamically (SQL administration API)

start mirroring argument: Starts storage space mirroring (SQL administration API)

Use the start mirroring argument with the **admin()** or **task()** function to start mirroring for a specified dbspace, blobspace, or sbspace.

Syntax



Element	Description	Key Considerations
space_name	The name of the blobspace, dbspace, or sbspace.	

Usage

This function is equivalent to the **onspaces -m** command.

Example

The following example starts mirroring for the dbspace named **dbsp1**:

```
EXECUTE FUNCTION task("start mirroring","dbsp1");
```

Related reference:
onspaces -m: Start mirroring

statement cache enable argument: Enable the SQL statement cache (SQL administration API)

Use the statement cache enable argument with the **admin()** or **task()** function to enable the statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache enable"-+----->
                     '-task--'
>--)--;-------------------------------------------------------------><
```

Usage

This function is equivalent to the **onmode -e enable** command.

statement cache flush argument: Flush the SQL statement cache (SQL administration API)

Use the statement cache flush argument with the **admin()** or **task()** function to flush the statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache flush"-+----->
                     '-task--'
>--)--;-------------------------------------------------------------><
```

Usage

This function is equivalent to the **onmode -e flush** command.

statement cache hits argument: Specify the number of hits in the SQL statement cache (SQL administration API)

Use the statement cache hits argument with the **admin()** or **task()** function to specify the number of hits in the SQL statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache hits", "<numhits>"-+----->
                     '-task--'
>--)--;-------------------------------------------------------------------><
```

Usage

This function is equivalent to the **onmode -W STMT_CACHE_HITS** command.

statement cache nolimit argument: Control whether to insert qualified statements into the SQL statement cache (SQL administration API)

Use the statement cache nolimit argument with the **admin()** or **task()** function to control whether to insert statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache nolimit <0/1>"-+----->
                     '-task--'
>--)--;-------------------------------------------------------------------><
```

Usage

This function is equivalent to the **onmode -W STMT_CACHE_NOLIMIT {0|1}** command.

statement cache off argument: Disable the SQL statement cache (SQL administration API)

Use the statement cache off argument with the **admin()** or **task()** function to turn off the SQL statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache off"-+----->
                    '-task--'
>--)--;-------------------------------------------------------><
```

Usage

This function is equivalent to the **onmode -e off** command.

statement cache save argument: Save the SQL statement cache (SQL administration API)

Use the statement cache save argument with the **admin()** or **task()** function to save the statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache save"-+----->
                    '-task--'
>--)--;-------------------------------------------------------><
```

Usage

statement cache save is used to:

- Lock the Statement Cache
- Drop sysadmin:statement_cache table if it exists
- Create new sysadmin:statement_cache table
- Copy each entry from the Statement Cache and insert into the sysadmin:statement_cache table
  Note: If cache entry is Locked, the query plan will be preserved so that it can persist as-is when being restored.
  Note: If cache entry is Locked and has STMT_CACHE_QUERY_PLAN enabled, the current query plan will be stored in text form so it can persist as-is when being restored.
- Unlock the Statement Cache

statement cache restore argument: Restore the SQL statement cache (SQL administration API)

Use the statement cache restore argument with the **admin()** or **task()** function to restore the statement cache.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(--+-"statement cache restore"-+----->
                    '-task--'
>--)--;-------------------------------------------------------><
```

Usage

statement cache restore is used to:

- Lock the Statement Cache
- Read each row from sysadmin: statement_cache table
- Prepare the statement for the saved row so that statement will be inserted into the cache
  Note: If cache entry is Locked and has STMT_CACHE_QUERY_PLAN enabled, then it will be revived in its saved form.
  Note: If cache entry was Locked and had STMT_CACHE_QUERY_PLAN enabled when it was saved, then the text form of the query plan will be restored.
- Unlock the Statement Cache

stop json listener: Stop the wire listener (deprecated)

Use the stop json listener argument with the **admin()** or **task()** function to stop the wire listener.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
```

```
                    '-task--'

>--"stop json listener"--,"--+-------------------------------------------------+--"-->
                            '-,--"property_file"--+-------------------------+-'
                                                  '-,--"listener_arguments"-'

>--)--;-----------------------------------------------------><
```

Element	Description	Key Considerations
property_file	The name of the wire listener configuration file to use instead of the default.	The *property_file* is optional. The default wire listener configuration file is in $INFORMIXDIR/etc/jsonListener.properties.
listener_arguments	The command line argument to pass to the wire listener.	

Usage

The stop json listener argument stops the wire listener.

Example

In the following example, the wire listener is stopped by using the mycustom.properties file instead of the default jsonListener.properties file:

```
EXECUTE FUNCTION task("stop json listener", "mycustom.properties");
```

In this example, the wire listener is stopped by using mycustom.properties file instead of the default jsonListener.properties, and a command line argument is passed to the wire listener:

```
EXECUTE FUNCTION task("stop json listener", "mycustom.properties", "-port 27018");
```

Related information:
Wire listener command line options
Stopping the wire listener

stop listen argument: Stop a listen thread dynamically (SQL administration API)

Use the stop listen argument with the **admin()** or **task()** function to stop an existing listen thread for a SOCTCP or TLITCP network protocol without interrupting existing connections.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"stop listen"--,--"server_name"--)--;----------------------><
```

Element	Description	Key Considerations
server_name	The name of the database server for which you want to stop a listen thread.	

Usage

The definition of the listen thread must exist in the **sqlhosts** file.

This function does not update the **sqlhosts** file.

This function is equivalent to the **onmode -P stop** *server_name* command.

Example

The following command stops a listen thread for a server named **ids_serv3**:

```
EXECUTE FUNCTION task("stop listen","ids_serv3");
```

Related reference:
onmode -P: Start, stop, or restart a listen thread dynamically
start listen argument: Start a listen thread dynamically (SQL administration API)
restart listen argument: Stop and start a listen thread dynamically (SQL administration API)

stop mirroring argument: Stops storage space mirroring (SQL administration API)

Use the stop mirroring argument with the **admin()** or **task()** function to stop mirroring for a specified dbspace, blobspace, or sbspace.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"stop mirroring"--,--"space_name"--)--;---------------------><
```

Element	Description	Key Considerations
space_name	The name of the blobspace, dbspace, or sbspace.	

Usage

This function is equivalent to the **onspaces -r** command.

Example

The following example stops mirroring for the dbspace with the name of **dbsp1**:

```
EXECUTE FUNCTION task("stop mirroring","dbsp1");
```

Related reference:
onspaces -r: Stop mirroring

storagepool add argument: Add a storage pool entry (SQL administration API)

Use the storagepool add argument with the **admin()** or **task()** function to add an entry to the *storage pool* (a collection of available raw devices, cooked files, or directories that Informix® can use to automatically add space to an existing storage space).

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--storagepool add--"--,--"path_name"------------------------>

>--,--"begin_offset"--,--"total_size"--,--"chunk_size"--,--"priority"-->

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
path_name	The path for the file, directory, or device that the server can use when additional storage space is required.	You do not need to add a final slash (“/”) to a directory name. You can use an environment variable in the path if the variable is in your environment when the **oninit** command runs.
begin_offset	The offset in kilobytes into the device where Informix can begin allocating space.	If you specified a path to a directory, you must specify `0` as the offset.
total_size	The total space available to Informix in this entry. The server can allocate multiple chunks from this amount of space.	Be sure to specify `0` for the total size of a directory. If you specify a value that is not zero for a directory, the SQL administration API command returns an error. If you specify `0` for a file or device, the server will allocate one extendable chunk from the entry.
chunk_size	The minimum size of a chunk that can be allocated from the device, file, or directory.	The smallest chunk that you can create is 1000 K. Therefore, the minimum chunk size that you can specify is 1000 K. See admin() and task() Argument Size Specifications.
priority	The priority of the directory, file, or device when the server searches through the storage pool for space. • 1 = High priority • 2 = Medium priority • 3 = Low priority	The server tries to allocate space from a high-priority entry before it allocates space from a lower priority entry.

Usage

The server uses entries in the storage pool if necessary to add a new chunk to a storage space.

When you add an entry to the storage pool, you might want some control over how that entry is used. For example, to reduce the number of chunks in an instance, you might want only large chunks of space to be allocated from a particular raw device, and might not want the chunks to be extendable. In this case, configure the chunk size for that storage pool entry to be large.

You can add the following types of entries to the storage pool:

- A fixed-length raw device
- A fixed-length cooked file
- An extendable raw device (for extending the size of a chunk)
- An extendable cooked file (for extending the size of a chunk)
- A directory

A storage pool entry that is a directory is always categorized as extendable, because it does not have a total size. If new chunks are automatically created in the directory, the server marks those chunks as extendable. When you add a storage pool entry that is a directory, you might want a small chunk size, because the server can extend any chunk created in the directory and smaller chunk sizes can reduce the amount of wasted space in the instance.

If a storage pool entry is on a High-Availability Data Replication (HDR) primary server, the same path in the entry must be available on all secondary servers in the HDR cluster.

The default units for sizes and offsets are kilobytes. However, you can specify information in any of the ways shown in the following examples:

- "100000"
- "100000 K"
- "100 MB"
- "100 GB"
- "100 TB"

Example: Adding a storage pool entry for a directory

The following command adds a directory named /region2/dbspaces with a beginning offset of 0, a total size of 0, an initial chunk size of 20 megabytes, and a high priority:

```
DATABASE sysadmin;
EXECUTE FUNCTION task("storagepool add", "/region2/dbspaces", "0", "0",
"20000", "1");
```

Example: Adding a storage pool entry for a fixed-length raw device

The following command adds a fixed-length raw device with the path name /dev/raw/device1 and a total of 500 megabytes of space to the storage pool. The command specifies a beginning offset of 50 megabytes, a total size of 10 gigabytes, a minimum of 100 megabytes to allocate to a chunk, and a low priority.

```
EXECUTE FUNCTION task("storagepool add", "/dev/rawdevice1", "50 MB",
 "10 GB", "100 MB", "3");
```

Example: Adding a storage pool entry for a fixed-length cooked file

The following command adds a fixed-length cooked file and 1 gigabyte of space to the storage pool. The command specifies a beginning offset of 0, a total size of 1000000 kilobytes, a minimum of 50000 kilobytes to allocate to a chunk, and a medium priority:

```
EXECUTE FUNCTION task("storagepool add", "/ifmx_filesystem/storage/cooked7",
 "0", "1000000", "50000", "2");
```

When adding this entry, the server tries to increase the size of the cooked7 file to 1 gigabyte. If the server cannot increase the size because the file system is full, the server returns an error message and does not add the entry to the storage pool.

Informix uses part of the cooked file initially, but can use more of the device as necessary as spaces fill.

Example: Adding a storage pool entry for an extendable cooked file

The following command adds a cooked file with the path name /ifmx/CHUNKFILES/cooked2. If the server uses this entry, the server creates one chunk with an initial size of 1 GB, and the server automatically marks the chunk as extendable.

```
EXECUTE FUNCTION task("storagepool add", "/ifmx/CHUNKFILES/cooked2",
 "0", "0", "1 GB", "2");
```

Example: Adding a storage pool entry with an environment variable in the path

The following example includes an environment variable in the path. The variable was present in the server environment when the **oninit** command ran.

```
EXECUTE FUNCTION task("storagepool add", "$DBSDIR/chunk1",
 "0", "100000", "20000", "2");
```

Related reference:
storagepool modify argument: Modify a storage pool entry (SQL administration API)
storagepool delete argument: Delete one storage pool entry (SQL administration API)
storagepool purge argument: Delete storage pool entries (SQL administration API)
Related information:
Automatic space management

storagepool delete argument: Delete one storage pool entry (SQL administration API)

Use the storagepool delete argument with the **admin()** or **task()** function to delete an entry from the storage pool.

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--storagepool delete--"--,--"--entry_id--"--)--;----------><
```

Element	Description	Key Considerations
entry_id	The ID of the storage pool entry.	The **storagepool** table in the **sysadmin** database contains a column that shows the ID of each entry in the storage pool.

Usage

Delete a storage pool entry if you do not want the server to continue to use the entry when expanding a storage space.

To delete all storage pool entries, storage pool entries that have a status of `Full`, or storage pool entries that have a status of `Error`, use the SQL administration API storagepool purge command. (The **storagepool** table in the **sysadmin** database contains a column that shows the status of each entry in the storage pool.)

Example

The following command deletes the storage pool entry with an entry id of 13:

```
EXECUTE FUNCTION task("storagepool delete", "13");
```

Related reference:
storagepool add argument: Add a storage pool entry (SQL administration API)
storagepool modify argument: Modify a storage pool entry (SQL administration API)
storagepool purge argument: Delete storage pool entries (SQL administration API)
Related information:
Automatic space management

storagepool modify argument: Modify a storage pool entry (SQL administration API)

Use the storagepool modify argument with the **admin()** or **task()** function to modify an entry for a directory, cooked file, or raw device that Informix® can use when additional storage space is required.

Syntax

```
>>-EXECUTE FUNCTION--+-admin-+--(------------------------------>
                     '-task--'

>--"--storagepool modify--"--,--"--entry_id--"--,--------------->

>--"new_total_size"--,--"new_chunk_size"--,--"new_priority"----->

>--)--;-------------------------------------------------------><
```

Element	Description	Key Considerations
entry_id	The ID of the storage pool entry.	The **storagepool** table in the **sysadmin** database contains a column that shows the ID of each entry in the storage pool.
new_total_size	The new amount of total space available to Informix in this entry. The server can allocate multiple chunks from this amount of space.	Be sure to specify `0` for the total size of a directory. If you specify a value that is not zero for a directory, the SQL administration API command returns an error. If you specify `0` for a file or device, the server allocates one extendable chunk from the entry.
new_chunk_size	The minimum size of a chunk that can be allocated from the device, file, or directory.	The smallest chunk that you can create is 1000 K. Therefore, the minimum chunk size that you can specify is 1000 K. See admin() and task() Argument Size Specifications.

Element	Description	Key Considerations
new_priority	The priority of the directory, file, or device when the server searches through the storage pool for space. • 1 = High priority • 2 = Medium priority • 3 = Low priority	The server attempts to allocate space from a high-priority entry before it allocates space from a lower priority entry.

Usage

Sometimes you might want to change a storage pool entry. For example, you might want to increase the total size of the storage pool when it runs out of space, or you might want to change the chunk size or the priority. When you change the entry, include the total size, chunk size, and priority even if you do not want to change all of these values.

You cannot modify the path or the beginning offset of a storage pool entry. If you want to change either of those values, you must delete the storage pool entry and add an entry with the new path or beginning offset.

If a storage pool entry is on a High-Availability Data Replication (HDR) primary server, the same path in the entry must be available on all secondary servers in the HDR cluster.

The default units for storage pool sizes and offsets are kilobytes. However, you can specify information in any of the ways shown in the following examples:

- "100000"
- "100000 K"
- "100 MB"
- "100 GB"
- "100 TB"

Examples

The following command changes the total size, chunk size, and priority of the storage pool entry that has an ID of 4 to 10 gigabytes, 10 megabytes, and a medium priority.

```
EXECUTE FUNCTION task("storagepool modify", "4", "10 GB", "10000", "2");
```

Suppose that you add an entry to the storage pool and the entry has a path of (/dev/IDS/chunk2), an offset of 0, a total size of 100 megabytes, a minimum chunk size of 100 megabytes, and a priority of 2. Before Informix allocates any space from this entry, you use **onspaces** to manually add a 50 megabyte chunk with the same path (/dev/IDS/chunk2), and an offset of 50 megabytes. The server only detects the overlap when it attempts to use this entry to automatically create a chunk. At that time, the server marks the entry with an "Error" status and attempts to use another entry to create the chunk.

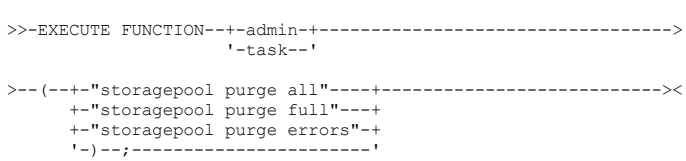
You can correct the problem by changing the total size of the storage pool entry (for example, for entry 2), to 50 megabytes and by changing the minimum chunk size of the entry to 50 megabytes, as follows:

```
EXECUTE FUNCTION task("storagepool modify", "2", "50 MB", "50 MB", "2");
```

Related reference:
storagepool add argument: Add a storage pool entry (SQL administration API)
storagepool delete argument: Delete one storage pool entry (SQL administration API)
storagepool purge argument: Delete storage pool entries (SQL administration API)
Related information:
Automatic space management

storagepool purge argument: Delete storage pool entries (SQL administration API)

Use the storagepool purge argument with the **admin()** or **task()** function to delete all storage pool entries, storage pool entries that have a status of `Full`, or storage pool entries that have a status of `Error`.



Usage

Use the storagepool purge all argument to delete all entries in the storage pool.

Use the storagepool purge full argument to delete all storage pool entries that have a status of `Full`.

Use the storagepool purge errors argument to delete all storage pool entries that have a status of `Error`.

The **storagepool** table in the **sysadmin** database contains a column that shows the status of each entry in the storage pool.

Example

The following command deletes all storage pool entries that have a status of `Full`:

```
EXECUTE FUNCTION task("storagepool purge full");
```

Related reference:
storagepool add argument: Add a storage pool entry (SQL administration API)
storagepool modify argument: Modify a storage pool entry (SQL administration API)
storagepool delete argument: Delete one storage pool entry (SQL administration API)
Related information:
Automatic space management

Table and fragment pfsc_boost argument: Enable or disable a boosted partition free space cache (PFSC)

You can enable and disable a boosted partition free space cache for a table or fragment using SQL administration API **admin()** or **task()** functions and arguments.

The built-in SQL administration API **admin()** or **task()** functions are defined in the **sysadmin** database of each Informix® instance. By default, only user **informix** can invoke these functions. If Connect privilege on the **sysadmin** database is granted to user **root** or to **DBSA** group members, they too can invoke the SQL administration API **admin()** or **task()** functions when they are connected directly or remotely to the **sysadmin** database.

The SQL administration API **admin()** or **task()** command arguments that you can use to enable or disable boosted PFSCs in tables and table fragments are:

- **table pfsc_boost enable**: Use the *"table pfsc_boost enable"* sysadmin task() or admin() command to create a boosted PFSC for a table. For example:

  ```
  execute function task("table pfsc_boost enable", "<table name>", "<database name>");
  ```

- **fragment pfsc_boost enable**: Use the *"fragment pfsc_boost enable"* sysadmin task() or admin() command to create a boosted PFSC for a specific fragment. For example:

  ```
  execute function task("fragment pfsc_boost enable", "<partnum>");
  ```

- **table pfsc_boost disable**: Use the *"table pfsc_boost disable"* sysadmin task() or admin() command to disable a boosted PFSC for a table. For example:

  ```
  execute function task("table pfsc_boost disable", "<table name>", "<database name>");
  ```

- **fragment pfsc_boost disable**: Use the *"fragment pfsc_boost disable"* sysadmin task() or admin() command to disable a boosted PFSC for a specific fragment. For example:

  ```
  execute function task("fragment pfsc_boost disable", "<partnum>");
  ```

Note: Boosted partition free space caches are re-created automatically in shared memory on server restart.
For more information, see Boosted Partition Free Space Caches (PFSC).

Table and fragment compress and uncompress operations (SQL administration API)

You can compress and uncompress the data in a table or in table fragments with SQL administration API **admin()** or **task()** functions and arguments. Compression operations apply only to the contents of data rows and the images of those data rows that appear in logical log records.

The built-in SQL administration API **admin()** or **task()** functions are defined in the **sysadmin** database of each Informix® instance. By default, only user **informix** can invoke these functions. If Connect privilege on the **sysadmin** database is granted to user **root** or to **DBSA** group members, they too can invoke the SQL administration API **admin()** or **task()** functions when they are connected directly or remotely to the **sysadmin** database.

The SQL administration API **admin()** or **task()** command arguments that you can use for compress and uncompress operations in tables and table fragments are:

table *compression parameters*
: Performs various compression operations to all fragments of a specified table. For more information, see table or fragment arguments: Compress data and optimize storage (SQL administration API).

fragment *compression parameters*
: Performs various compression operations to a single fragment or a specified set of fragments that belong to a specific table. For more information, see table or fragment arguments: Compress data and optimize storage (SQL administration API).

compression purge_dictionary
: Deletes all inactive compression dictionaries or all inactive compression dictionaries that were created before a date that you specify. For more information, see purge compression dictionary arguments: Remove compression dictionaries (SQL administration API).

Table and fragment compression operations include creating compression dictionaries, estimating compression ratios, compressing data in tables and table fragments, consolidating free space (repacking), returning free space to a dbspace (shrinking), uncompressing data, and deleting individual table and fragment compression dictionaries.

When you run SQL administration API compression and uncompression commands, you compress and uncompress both row data and simple large objects in dbspaces. You can also specify whether to compress or uncompress only row data or only simple large objects in dbspaces.

An **admin()** command returns an integer; a **task()** command returns a string.

For information on the types of data that you can compress, compression ratios, compression estimates, and compression dictionaries, as well as procedures for using compression command parameters, see Compression in the *IBM® Informix Administrator's Guide*. For information on utilities and the **sysmaster** table and view that display compression information, see syscompdicts_full.

You can also compress, optimize storage, and estimate compression benefits for B-tree indexes. See index compress repack shrink arguments: Optimize the storage of B-tree indexes (SQL administration API) and index estimate_compression argument: Estimate index compression (SQL administration API).

- table or fragment arguments: Compress data and optimize storage (SQL administration API)
  Use SQL administration API functions with **table** or **fragment** arguments to create compression dictionaries, to estimate compression ratios, to compress data in tables and table fragments, to consolidate free space (repack), to return free space to a dbspace (shrink), to uncompress data, and to delete compression dictionaries.
- Output of the estimate compression operation (SQL administration API)
  After you run the command for estimating compression ratios, the database server displays information that shows the estimate of the compression ratio that can be achieved, along with the currently achieved compression ratio (if it exists).
- purge compression dictionary arguments: Remove compression dictionaries (SQL administration API)
  Call the **admin()** or **task()** function with the compression purge_dictionary initial command to delete all inactive compression dictionaries or all inactive compression dictionaries that were created for a compressed table or fragment before a specified date. You must uncompress tables and fragments, which makes the dictionaries inactive, before you delete any compression dictionaries that were created for the tables and fragments.

table or fragment arguments: Compress data and optimize storage (SQL administration API)

Use SQL administration API functions with **table** or **fragment** arguments to create compression dictionaries, to estimate compression ratios, to compress data in tables and table fragments, to consolidate free space (repack), to return free space to a dbspace (shrink), to uncompress data, and to delete compression dictionaries.

When you run SQL administration API compression and uncompression commands, you compress and uncompress row data and simple large objects in dbspaces. You can also specify whether to compress or uncompress only row data or only simple large objects in dbspaces.

```
Syntax: table data compression command arguments

>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"--table--| command arguments |--"------------------------>

>--,--"table_name"--+---------------------------------------+------>
                    '-,--"database_name"--+------------+-'
                                          '-,--"owner"-'

>--)--;-------------------------------------------------------><


Syntax: fragment data compression command arguments

>>-EXECUTE FUNCTION--+-admin-+---------------------------------->
                     '-task--'

>--(--"--fragment--| command arguments |--"---------------------->

>--,--"partition_number"--)--;--------------------------------><


Table and fragment command arguments

|--+-create_dictionary-----------------------------------------------+--|
   +-compress--+-----------+--+--------+--+--------+--+----------+-+
   |           '-+-rows--+-'  '-repack-'  '-shrink-'  '-parallel-' |
   |             '-blobs-'                                         |
   +-repack--+--------+--+----------+-------------------------------+
   |         '-shrink-'  '-parallel-'                              |
   +-shrink----------------------------------------------------------+
   +-estimate_compression--------------------------------------------+
   +-repack_offline--------------------------------------------------+
   +-uncompress--+-----------+--+----------+-----------------------+
   |             '-+-rows--+-'  '-parallel-'                       |
   |               '-blobs-'                                       |
   +-uncompress_offline----------------------------------------------+
   +-purge_dictionary------------------------------------------------+
   '-update_ipa--+----------+---------------------------------------'
                 '-parallel-'
```

Command arguments

The following table describes each argument.

Table 1. Arguments for Compress and Uncompress Operations

Argument	Description
blobs	Specifies that you want to compress or uncompress only simple large objects in dbspaces and not row data.
compress	Compresses all existing rows in-place, without moving them (without repacking the table). This option automatically compresses row data and simple large objects in dbspaces. To compress only row data or only simple large objects in dbspaces, also use the **rows** or **blobs** element. If a compression dictionary for the target table or fragment does not exist, the compress operation also creates the dictionary.
create_dictionary	Builds a compression dictionary, which is a library of frequently occurring patterns and the symbol numbers that replace them in compressed rows. After a dictionary is created, any newly inserted or updated rows will be compressed if they are compressible. Existing rows are not compressed.
estimate_compression	Estimates both a new compression ratio and a current ratio. The current ratio is 0.0 percent if the table is not compressed.
parallel	Runs the compress, repack, update_ipa, or uncompress operation in parallel. A thread is started for each fragment of the table or fragment list and the operation is run in parallel across those fragments.
purge_dictionary	Deletes an inactive compression dictionary after you uncompress a table or fragment.
repack	Consolidates free space by moving data to the front of the fragment or table. Because the repack operation moves rows while the fragment is online, other queries that access the fragment that are using an isolation level below Repeatable Read might occasionally find the same row twice or miss finding a row. To avoid this possibility, use the Repeatable Read isolation level for concurrent queries; or, instead of using the **repack** argument, use the **repack_offline** argument.
repack_offline	Consolidates free space by moving data to the front of the table or fragment, while holding an exclusive lock on the table or fragment. This operation prevents all other access to data until the operation is completed.
rows	Specifies that you want to compress or uncompress only row data and not simple large objects in dbspaces.
shrink	Returns free space at the end of a fragment or table to the dbspace, thus reducing the total size of the fragment or table.
uncompress	Deactivates compression for new INSERT and UPDATE operations, uncompresses all compressed rows, and deactivates the compression dictionary. This operation also allocates new pages for a fragment and moves uncompressed rows that no longer fit on their original pages to the new pages. Because this operation moves rows while the fragment is online, other queries that access the fragment that are using an isolation level below the Repeatable Read isolation level might occasionally find the same row twice or miss finding a row. To avoid this possibility, use the Repeatable Read isolation level for concurrent queries, or instead of using the **uncompress** argument, use the **uncompress_offline** argument. This option automatically uncompresses row data and simple large objects in dbspaces. To compress only row data or only simple large objects in dbspaces, also use the **rows** or **blobs** element.
uncompress_offline	Deactivates compression for new INSERT and UPDATE operations, uncompresses all compressed rows, and deactivates the compression dictionary, while holding an exclusive lock on the fragment. This prevents all other access to the fragment data until the operation is completed. This operation also allocates new pages for a fragment and moves uncompressed rows that no longer fit on their original pages to the new pages.
update_ipa	Removes outstanding in-place alter operations for the specified table or fragments.

Command elements

The following tables show the elements that you can use in commands.

Table 2. Table compression and storage optimization command elements

Element	Description	Key Considerations
database_name	The name of the database that contains the specified table.	Optional. If you do not specify a *database*, Informix® uses the current database. If you enter a database name, you must use the same uppercase or lowercase letters that are in system catalog tables.
owner	The authorization identifier of the owner of the database that contains the specified table.	Optional. If you do not specify an *owner*, Informix uses the current owner. If you enter an owner name, you must use the same uppercase or lowercase letters that are in system catalog tables.
table_name	The name of the table that contains the data.	You must use the same uppercase or lowercase letters that are in system catalog tables.

Table 3. Fragment compression and storage optimization command elements

Element	Description	Key Considerations
partition_number	A space-separated list of partition numbers that belong to the same table.	

Usage

Informix uses the compression dictionary to compress data.

After you run a **compress** command on a table or fragment, Informix automatically compresses any new rows that you add to the table or fragment. If the table or fragment contains more than 2000 rows when you run the **compress** command, a compression dictionary is built and all the rows are compressed. If the table or fragment contains fewer than 2000 rows when you run the compression command, the table or fragment is enabled for automatic compression. After 2000 rows are inserted, a compression dictionary is created and all rows after the initial 2000 rows are compressed. To compress the initial 2000 rows, run the **compress** command again.

If your data changes significantly, the compression dictionary might not be effective. In this situation, uncompress and then compress again.

You can cancel a command with a **compress** or **uncompress** argument, for example, by typing CTRL-C in DB-Access. You can reissue commands with **repack**, **repack_offline**, **uncompress**, and **uncompress_offline** arguments after a prior interrupted command.

You cannot perform a compress, repack, repack_offline, shrink, uncompress, or uncompress_offline operation on a table or fragment while any of these operations is already occurring on the table or fragment.

When you specify multiple operations in a single command, the server performs the operations in this order:

- **create_dictionary**
- **compress**
- **repack**
- **shrink**

Compress, repack, repack_offline, uncompress, and uncompress_offline operations can consume large amounts of log files. Configure your logs to be larger if any workload that you expect to run, including but not limited to these compression operations, consumes log files faster than one every 30 seconds.

Compress, repack, and uncompress operations are logged, but run in small portions.

If you change the fragmentation strategy for a table after you perform a compression operation, the table loses its compression status and will need to be recompressed.

Dropping or disabling indexes before you complete a repack_offline or uncompress_offline operation can decrease the amount of time that it takes the server to complete the operation. Afterward, you can re-create or re-enable the indexes, preferably taking advantage of PDQ. Dropping or disabling the indexes and then creating or enabling them again can be faster than completing a repack_offline or uncompress_offline operation without doing this.

Do not drop a dbspace that Change Data Capture (CDC) API is using, if the dbspace ever contained compressed tables, because this might delete compression dictionaries that CDC still needs.

Repack

The compress operation normally creates a quantity of free space on individual data and remainder pages, but the space is not consolidated at the end of the table or fragment. Instead, the space can be used to hold newly inserted rows, with the table not growing any larger until this space is filled.

A compress operation, which only occurs online, compresses rows of a table in-place. The repack operation moves the rows. You can perform a repack operation online or offline. An online operation allows concurrent activity to occur on a table. However, this can result in *phantom rows*. (Phantom rows are rows that are initially modified or inserted during a transaction that is later rolled back.)

To avoid phantom rows, you might want to repack offline, when you can afford to keep other users from accessing a table or fragment. For example, you could perform a compress operation with concurrent activity during the day, and then perform a repack_offline operation at night, when no concurrent activity is expected on the table.

You cannot perform an offline operation with an online operation. For example, while you can perform a combined compress repack operation, you cannot perform a combined compress repack_offline operation. If you want to repack offline, you must do this in two steps:

1. Perform a compress operation.
2. Perform a repack_offline operation.

Similarly you cannot perform a repack_offline shrink operation.

If light appends (unbuffered, unlogged insert operations) occur in a table or fragment while a repack operation is occurring, the repack operation does not complete the consolidation of space at the end of a table or fragment. The repack operation does not complete because the new extents are added in the location where the repack operation already occurred, so space cannot be returned to the dbspace. To complete the repack process, you must run a second repack operation after light append activity completes. This second repack operation builds on the work of the first repack operation.

Shrink

The shrink operation is typically performed after a repack operation.

You can safely shrink the entire table without compromising the allocation strategy of the table. For example, if you have a fragmented table with one fragment for each day of the week and many fragments pre-allocated for future use, you can shrink the table without compromising this allocation strategy. If the table is empty, Informix shrinks the table to the initial extent size that was specified when the table was created.

When you initiate a shrink operation, Informix shortens extents as follows:

- It shortens all extents except the first extent to as small a size as possible.
- If the table is entirely in the first extent (for example, because the table is an empty table), Informix does not shrink the first extent to a size that was smaller than the extent size that was specified when the table was created with the CREATE TABLE statement.

You can use the MODIFY EXTENT SIZE clause of the ALTER TABLE statement to reduce the current extent size. After you do this, you can rerun the shrink operation to shrink the first extent to the new extent size.

Uncompress

The uncompress operation has no effect on any table or fragment it is applied to that is not compressed.

After you uncompress a table or fragment, you can perform a purge_dictionary operation to delete the dictionary for that table or fragment.

Purge

Before you perform a purge_dictionary operation for tables and fragments, you must:

- Uncompress the tables and fragments.
  When you uncompress a table or fragment, Informix marks the dictionary for the table or fragment as inactive.
- Be sure that Enterprise Replication functions do not need the compression dictionaries for older logs.
- Archive any dbspace that contains a table or fragment with a compression dictionary, even if you have uncompressed data in the table or fragment and the dictionary is no longer active.

You can also delete all compression dictionaries or all compression dictionaries that were created before and on a specified date. For information, see purge compression dictionary arguments: Remove compression dictionaries (SQL administration API).

Examples

The following command compresses, repacks, and shrinks both row data in a table that is named **auto** in the **insurance** database of which **tjones** is the owner and simple large objects in the dbspace.

```
EXECUTE FUNCTION task("table compress repack shrink","auto",
"insurance","tjones");
```

The following command compresses only row data in a table named **dental** in parallel.

```
EXECUTE FUNCTION task("table compress rows parallel","dental");
```

The following command uncompresses the fragment with the partition number 14680071.

```
EXECUTE FUNCTION task("fragment uncompress","14680071");
```

The following command uncompresses only row data in the fragment with the partition number 14680071 in parallel.

```
EXECUTE FUNCTION task("fragment uncompress rows parallel","14680071");
```

The following command estimates the benefit of compressing a table that is named **home** in the **insurance** database of which **fgomez** is the owner.

```
EXECUTE FUNCTION task("table estimate_compression","home",
"insurance","fgomez");
```

The following command removes pending in-place alter operations on a table that is named **auto** in parallel.

```
EXECUTE FUNCTION task("table update_ipa parallel","auto");
```

After you run the command, the database server displays an estimate of the compression ratio that can be achieved, along with the currently achieved compression ratio (if it exists). For information about the output of the command, see Output of the estimate compression operation (SQL administration API).

Related reference:
Output of the estimate compression operation (SQL administration API)
onstat -g ath command: Print information about all threads
Related information:
Compression

Output of the estimate compression operation (SQL administration API)

After you run the command for estimating compression ratios, the database server displays information that shows the estimate of the compression ratio that can be achieved, along with the currently achieved compression ratio (if it exists).

Table 1. Information that an estimate_compression command displays

Column	Information Displayed
`est`	This is the estimate of the compression ratio that can be achieved with a new compression dictionary. The estimate is a percentage of space saved compared to no compression.
`curr`	This is the estimate of the currently achieved compression ratio. This estimate is a percentage of space saved compared to no compression. `0.0%` will always appear for non-compressed fragments or tables.
`change`	This is the estimate of the percentage point gain (or possibly loss, although that should be rare) in the compression ratio that you could achieve by switching to a new compression dictionary. This is just the difference between `est` and `curr`. If the table or fragment is not compressed, you can create a compression dictionary with the compress parameter. If the fragment is compressed, you must perform an uncompress or uncompress_offline operation, before you can compress.
`partnum`	This is the partition number of the fragment.
`coloff`	This value defines whether the estimate is for in-row data or simple large objects in the dbspace, as follows: • -1 indicates that the estimate for in-row data • A positive numeric value indicates that the estimate is for a partition simple large object at the offset identified by the value. The offset is the column offset in the table in bytes.
`table`	This is the full name of the table to which the fragment belongs, in format *`database:owner.tablename`* If you are estimating compression benefits for an index, the full name of the index appears in this column.

Example

The following output shows that a .4 percent increase in saved space can occur if you recompress the first fragment. A 75.7 percent increase can occur if you compress the second fragment, which is not compressed. The value -1 in the `coloff` column indicates that in-row data is compressed.

```
est   curr  change  partnum    coloff  table
----- ----- ------ -------------  ------------------------------
75.7% 75.3%   +0.4  0x00200003     -1  insurance:bwilson.auto
75.7%  0.0%  +75.7  0x00300002     -1  insurance:pchang.home
```

The following output shows compression estimates for in-row data (in the first row) and simple large objects at offsets 4 and 60 (in the second and third rows):

```
est   curr  change  partnum    coloff  table
----- ----- ------ -------------  --------------------
75.4% 71.5%  +3.9 0x00200002       -1 test:mah.table1
 5.0% 75.0%  +0.0 0x00200002        4 test:mah.table1
75.0% 75.0%  +0.0 0x00200002       60 test:mah.table1
```

Output from compression estimates for tables and fragments look the same, except that the output for a table always shows all fragments in the table, while the output for a fragment only shows information for the specified fragments.

Related reference:
index estimate_compression argument: Estimate index compression (SQL administration API)
table or fragment arguments: Compress data and optimize storage (SQL administration API)

purge compression dictionary arguments: Remove compression dictionaries (SQL administration API)

Call the **admin()** or **task()** function with the compression purge_dictionary initial command to delete all inactive compression dictionaries or all inactive compression dictionaries that were created for a compressed table or fragment before a specified date. You must uncompress tables and fragments, which makes the dictionaries inactive, before you delete any compression dictionaries that were created for the tables and fragments.

Syntax: Compression Purge_Dictionary

```
>>-EXECUTE FUNCTION--+-admin-+--(--"--compression purge_dictionary--"--,--"--date--"--)-- ;-><
                     '-task--'
```

Usage

Before you perform a purge_dictionary operation for tables and fragments, you must:

- Uncompress the tables and fragments.
  When you uncompress a table or fragment, Informix® marks the dictionary for the table or fragment as inactive.
- Be sure that Enterprise Replication functions do not need the compression dictionaries.
- Archive any dbspace that contains a table or fragment with a compression dictionary, even if you have uncompressed data in the table or fragment and the dictionary is no longer active.

The compression purge_dictionary command deletes all compression dictionaries.

The compression purge_dictionary command with a date as the second argument deletes all compression dictionaries that were created before and on a specified date. You can use any date in a format that can be converted to a DATE data type based on your locale and environment. For example, you can specify `03/29/2009`, `03/29/09`, or `Mar 29, 2009`.

You can also delete a specific compression dictionary by calling the **admin()** or **task()** function with table or fragment as the initial command and purge_dictionary as the next argument.

You cannot delete compression dictionaries that were created for indexes. The database server removes these compression dictionaries when the indexes are dropped.

The following command tells Informix to remove all dictionaries that were created before and on July 8, 2009:

```
EXECUTE FUNCTION task("compression purge_dictionary", "07/08/2009");
```

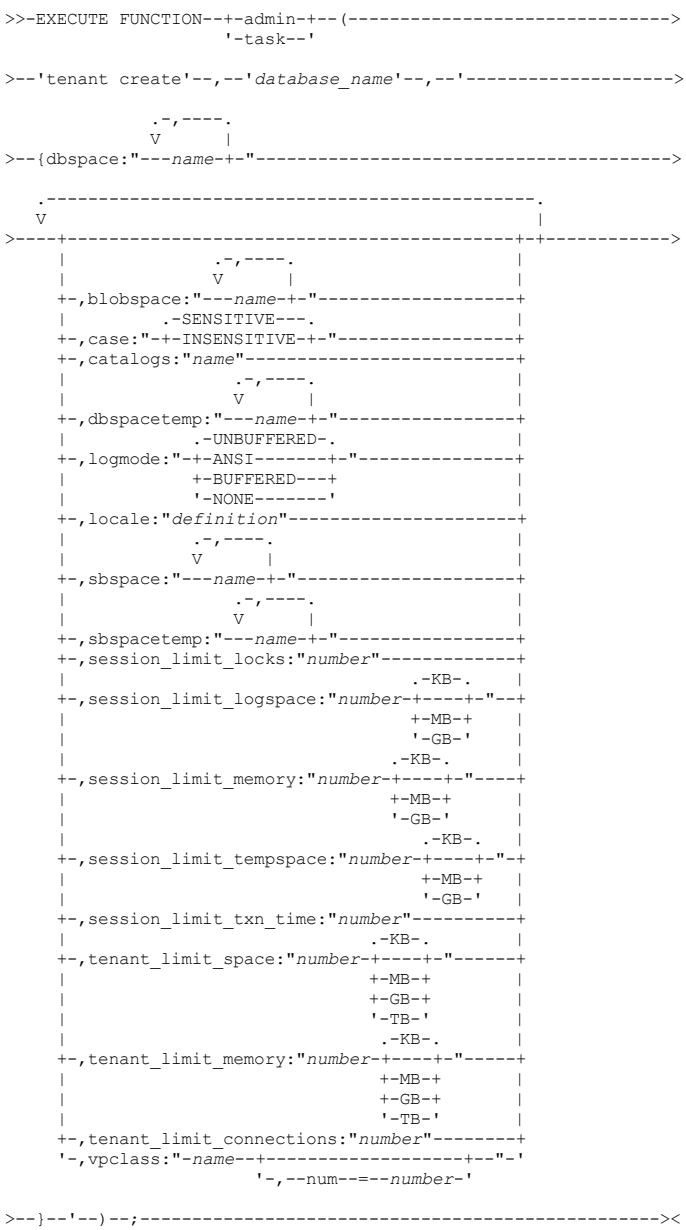
The following command tells Informix to remove the inactive dictionary for a table named **auto** in the **insurance** database of which **tjones** is the owner.

```
EXECUTE FUNCTION task("table purge_dictionary",
"auto", "insurance", "tjones");
```


tenant create argument: Create a tenant database (SQL Administration API)

Use the **tenant create** argument with the **admin()** or **task()** function to create a tenant database.

Syntax



Element	Description	Key Considerations
blobspace	A comma-separated list of one or more blobspaces that are assigned to the tenant database.	At least one blobspace is required if the tenant database contains simple large objects. blobspaces must be empty to be assigned to a tenant database. blobspaces must exist before being assigned to a tenant database. Simple large objects that are created outside of a tenant database cannot be stored in the tenant database's blobspaces.
case	Database sensitivity to uppercase and lowercase letters: INSENSITIVE Case insensitive. SENSITIVE Case sensitive. This is the default value.	If you omit this property, the database is case sensitive.
catalogs	A dbspace to store the tenant database catalogs.	The dbspace must be listed in the **dbspace** property. If you omit this property, the dbspace that is listed as the first value of the **dbspace** property contains the tenant database catalogs.

Element	Description	Key Considerations
database_name	The name of the tenant database.	The database name must be on the database server. An existing non-tenant database cannot become a tenant database.
dbspace	A comma-separated list of one or more dbspaces that are assigned to the tenant database.	dbspaces must be empty to be assigned to a tenant database. dbspaces must exist before being assigned to a tenant database. Objects that are created outside of a tenant database cannot be stored in the tenant database's dbspaces.
dbspacetemp	A comma-separated list of one or more temporary dbspaces that are assigned to a tenant database.	You can override the **dbspacetemp** property for a session by setting the DBSPACETEMP environment variable to a subset of the dbspaces that are specified by the **dbspacetemp** property. If the **dbspacetemp** property is omitted, temporary tables are stored in the temporary dbspaces that are specified by the DBSPACETEMP configuration parameter or environment variable.
locale	The locale of the database.	The values for **locale** are the same as the values for the DB_LOCALE environment variable. The default locale is en_US.819.
logmode	The log mode definition: UNBUFFERED Unbuffered database logging. This is the default. ANSI ANSI-compliant database logging. BUFFERED Buffered database logging. NONE No database logging.	If you omit this property, the logging mode is unbuffered.
sbspace	A comma-separated list of one or more sbspaces that are assigned to the tenant database.	At least one sbspace is required if the tenant database contains smart large objects. Smart large objects can include BLOB or CLOB data, and data and table statistics that are too large to fit in a row. sbspaces must be empty to be assigned to a tenant database. sbspaces must exist before being assigned to a tenant database. Smart large objects that are created outside of a tenant database cannot be stored in the tenant database's sbspaces. Some features, such as Enterprise Replication, spatial data, and basic text searching, require sbspaces.
num	The number of virtual processors to run.	If you do not include the **num** property, 1 virtual processor is started.
sbspacetemp	A comma-separated list of one or more temporary sbspaces that are assigned to the tenant database.	If you omit this property, temporary smart large objects are stored in the temporary sbspaces that are specified by the SBSPACETEMP configuration parameter.
session_limit_locks	The maximum number of locks available to a session.	The value must be 500 - 2147483648. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_locks** property takes precedent over the value of the SESSION_LIMIT_LOCKS configuration parameter. If you omit this property, the number of locks are set by the SESSION_LIMIT_LOCKS configuration parameter. If the SESSION_LIMIT_LOCKS configuration parameter is also not set, the maximum number of locks for a session is 2147483648. You can override the **session_limit_locks** property for a session by setting the IFX_SESSION_LIMIT_LOCKS environment option to a lower value than the **session_limit_locks** property value.
session_limit_logspace	The maximum amount of log space that a session can use for individual transactions.	The value must be 5120 - 2147483648 KB. Values are specified in KB, MB, or GB. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_logspace** property takes precedent over the value of the SESSION_LIMIT_LOGSPACE configuration parameter. If you omit this property, the amount of logspace is set by the SESSION_LIMIT_LOGSPACE configuration parameter. If the SESSION_LIMIT_LOGSPACE configuration parameter is also not set, the maximum amount of log space that a session can use for individual transactions is 2147483648 KB.

Element	Description	Key Considerations
session_limit_memory	The maximum amount of memory that a session can allocate.	The value must be 20480 - 2147483648 KB. Values are specified in KB, MB, or GB. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_memory** property takes precedent over the value of the SESSION_LIMIT_MEMORY configuration parameter. If you omit this property, the amount of memory is set by the SESSION_LIMIT_MEMORY configuration parameter. If the SESSION_LIMIT_MEMORY configuration parameter is also not set, the maximum amount of memory that a session can allocate is 2147483648 KB.
session_limit_tempspace	The maximum amount of temporary table space that a session can allocate.	The value must be 20480 - 2147483648 KB. Values are specified in KB, MB, or GB. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_tempspace** property takes precedent over the value of the SESSION_LIMIT_TEMPSPACE configuration parameter. If you omit this property, the amount of temporary table space is set by the SESSION_LIMIT_TEMPSPACE configuration parameter. If the SESSION_LIMIT_TEMPSPACE configuration parameter is also not set, the maximum amount of temporary table space that a session can allocate is 2147483648 KB.
session_limit_txn_time	The maximum amount of time that a transaction can run in a session.	The value must be 60 - 20000000000. Values are in seconds. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_txn_time** property takes precedent over the value of the SESSION_LIMIT_TXN_TIME configuration parameter. If you omit this property, the amount of time is set by the SESSION_LIMIT_TXN_TIME configuration parameter. If the SESSION_LIMIT_TXN_TIME configuration parameter is also not set, the maximum amount of time that a transaction can run in a session is 20000000000 seconds.
tenant_limit_space	The maximum amount of storage space on disk to a tenant database. When the limit is reached, subsequent operations that require more disk space are rejected.	The value must be 1048576 - 1717986918400 KB (1 GB - 200 TB). Values are specified in KB, MB, GB, or TB. The value of the **tenant_limit_space** property takes precedent over the value of the TENANT_LIMIT_SPACE configuration parameter. If you omit this property, the amount of space is set by the TENANT_LIMIT_SPACE configuration parameter. If the TENANT_LIMIT_SPACE configuration parameter is also not set, the maximum amount of storage space available to a tenant user is 1717986918400 KB.
tenant_limit_memory	The maximum amount of shared memory for all sessions that are connected to the tenant database. When the limit is exceeded, the session that is using the most shared memory is terminated.	The value must be 102400 - 2147483648 KB (100 MB - 2 TB). Values are specified in KB, MB, GB, or TB. The value of the **tenant_limit_memory** property takes precedent over the value of the TENANT_LIMIT_MEMORY configuration parameter. If you omit this property, the amount of memory is set by the TENANT_LIMIT_MEMORY configuration parameter. If the TENANT_LIMIT_MEMORY configuration parameter is also not set, the maximum amount of memory available to a tenant session is 2147483648 KB.
tenant_limit_connections	The maximum number of connections to a tenant database. When the limit is reached, subsequent connection requests are rejected.	The value must be 1 - 65536. The value of the **tenant_limit_connections** property takes precedent over the value of the TENANT_LIMIT_CONNECTIONS configuration parameter. If you omit this property, the number of connections is set by the TENANT_LIMIT_CONNECTIONS configuration parameter. If the TENANT_LIMIT_CONNECTIONS configuration parameter is also not set, the maximum number of connections for a tenant database is 65536.
vpclass	The name of the virtual processor class for running tenant-database session threads.	If you omit this property, session threads are run on CPU virtual processors. Values must be 8 characters or fewer. A maximum of 200 tenant virtual processor classes can be created. If the virtual processor class name is unique, you create a new tenant virtual processor class. If the virtual processor class name exists, the tenant database shares the class with other tenant databases. When a tenant virtual processor is dropped, the virtual processor class ID resources are not freed until the database server is restarted.

Usage

You must have DBA privileges or been granted the TENANT privilege to run this command. Only the first occurrence of each property is valid.

Run the **tenant create** argument with the **admin()** or **task()** to create a tenant database. The user that creates the database is granted DBA privileges. You can view the tenant database properties in the **sysadmin** database's **tenant** table.

The following statement creates a tenant database that is named **company_A**:

```
EXECUTE FUNCTION task('tenant create', 'company_A',
   '{dbspace:"company_A_dbs1,company_A_dbs2,company_A_dbs3",
     sbspace:"company_A_sbs",
     vpclass:"tvp_A,num=6",
     dbspacetemp:"company_A_tdbs",
```

```
     session_limit_locks:"1000",
     session_limit_memory:"100MB",
     session_limit_tempspace:"25MB",
     session_limit_logspace:"30MB",
     session_limit_txn_time:"120",
     tenant_limit_space:"2TB",
     tenant_limit_memory:"1GB",
     tenant_limit_connections:"1000",
     logmode:"ansi",
     locale:"fr_ca.8859-1",
     case:"insensitive",}'
);
```

The tenant database has the following attributes:

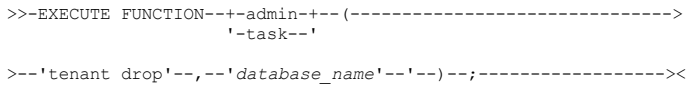
- Three dedicated dbspaces
- A dedicated sbspace
- Six tenant virtual processors
- A dedicated temporary dbspace
- A limit of 1000 locks per session
- A memory allocation limit of 100 MB per session
- A 25 MB limit for temporary table space per session
- A 30 MB limit for log space per session
- A 120 second limit on transaction times
- A limit of 2 TB on the total amount of storage space the tenant database can use
- A limit of 1 GB on the total amount of shared memory for all sessions that are connected to the tenant database
- A limit of 1000 connections
- ANSI logging mode
- French locale
- Case insensitivity
- Temporary smart large objects are stored in the sbspace that is specified by the database server's SBSPACETEMP configuration parameter.
- No blobspaces

Related reference:
create database argument: Create a database (SQL administration API)
TENANT_LIMIT_SPACE configuration parameter
tenant update argument: Modify tenant database properties (SQL Administration API)
tenant drop argument: Drop a tenant database (SQL Administration API)
SESSION_LIMIT_LOCKS configuration parameter
SESSION_LIMIT_MEMORY configuration parameter
SESSION_LIMIT_TEMPSPACE configuration parameter
SESSION_LIMIT_LOGSPACE configuration parameter
SESSION_LIMIT_TXN_TIME configuration parameter
TENANT_LIMIT_MEMORY configuration parameter
TENANT_LIMIT_CONNECTIONS configuration parameter
onstat -g ses command: Print session-related information
Related information:
Multitenancy
DB_LOCALE environment variable

tenant drop argument: Drop a tenant database (SQL Administration API)

Use the tenant drop argument with the **admin()** or **task()** function to drop a tenant database.

Syntax



Element	Description	Key Considerations
database_name	The name of the tenant database.	Must be an existing tenant database.

Usage

You must have DBA privileges or been granted the TENANT privilege to run this command. No other connections to the database can be open.

The tables and data in the database are deleted. The storage spaces that are dedicated to the tenant database are freed. The database tenant properties are removed from the **tenant** table in the **sysadmin** database. The associated tenant virtual processor class is dropped if it is not associated with any other tenant database.

The following statement drops the **companyA** tenant database:

```
EXECUTE FUNCTION task('tenant drop', 'companyA');
```

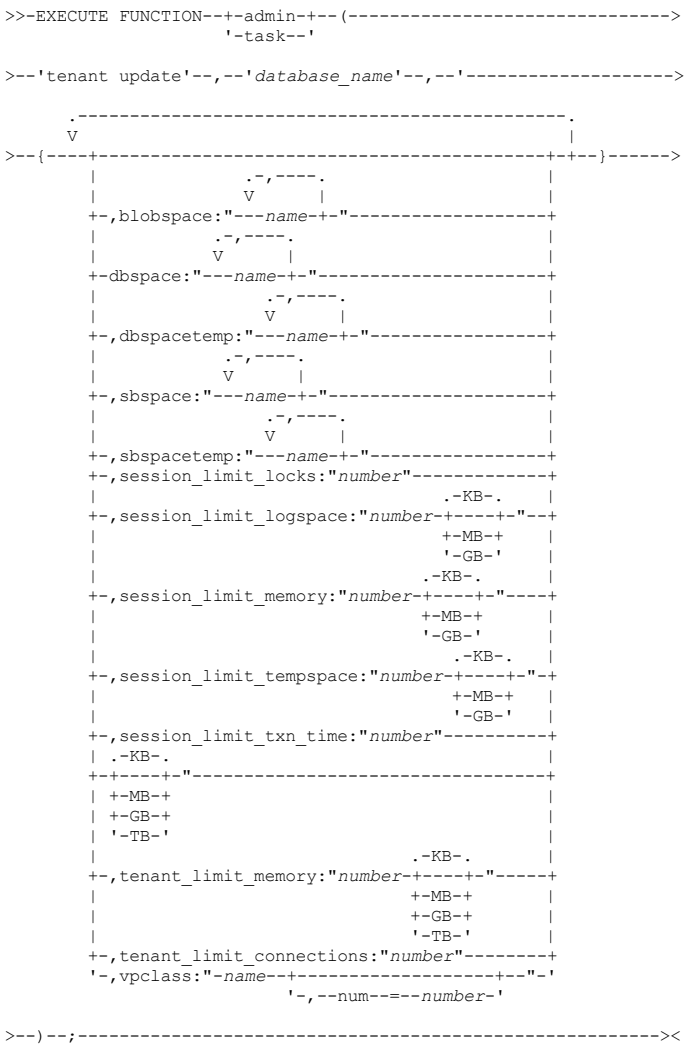
Related reference:

tenant create argument: Create a tenant database (SQL Administration API)
tenant update argument: Modify tenant database properties (SQL Administration API)
Related information:
Multitenancy

tenant update argument: Modify tenant database properties (SQL Administration API)

Use the tenant update argument with the **admin()** or **task()** function to modify the properties of a tenant database.

Syntax



Element	Description	Key Considerations
blobspace	A comma-separated list of one or more blobspaces that are assigned to the tenant database.	Specified blobspaces are appended to the tenant database's existing list of blobspaces. blobspaces must be empty to be assigned to a tenant database. blobspaces must exist before being assigned to a tenant database.
database_name	The name of the tenant database.	The database name must be on the database server.
dbspace	A comma-separated list of one or more dbspaces that are assigned to the tenant database.	Specified dbspaces are appended to the tenant database's existing list of dbspaces. dbspaces must be empty to be assigned to a tenant database. dbspaces must exist before being assigned to a tenant database.

Element	Description	Key Considerations
dbspacetemp	A comma-separated list of one or more temporary dbspaces that are assigned to a tenant database.	The existing **dbspacetemp** property value is replaced. You can override the **dbspacetemp** property for a session by setting the DBSPACETEMP environment variable to a subset of the dbspaces that are specified by the **dbspacetemp** property. If the **dbspacetemp** property is omitted, temporary tables are stored in the temporary dbspaces that are specified by the DBSPACETEMP configuration parameter or environment variable.
sbspace	A comma-separated list of one or more sbspaces that are assigned to the tenant database.	Specified sbspaces are appended to the tenant database's existing list of sbspaces. sbspaces must be empty to be assigned to a tenant database. sbspaces must exist before being assigned to a tenant database.
num	The number of virtual processors to run.	If you do not include the **num** property, 1 virtual processor is started.
sbspacetemp	A comma-separated list of one or more temporary sbspaces that are assigned to the tenant database.	The existing **sbspacetemp** property value is replaced.
session_limit_locks	The maximum number of locks for a session for users who do not have DBA privileges.	The existing **session_limit_locks** property value is replaced. The value must be 500 - 2147483648. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. If this property is not set, the number of locks are set by the SESSION_LIMIT_LOCKS configuration parameter. If the SESSION_LIMIT_LOCKS configuration parameter is also not set, the maximum number of locks for a session is 2147483648. You can override the **session_limit_locks** property for a session by setting the IFX_SESSION_LIMIT_LOCKS environment option to a lower value than the **session_limit_locks** property value.
session_limit_logspace	The maximum amount of log space that a session can use for individual transactions.	The existing **session_limit_logspace** property value is replaced. The value must be 5120 - 2147483648 KB. Values are specified in KB, MB, or GB. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_logspace** property takes precedent over the value of the SESSION_LIMIT_LOGSPACE configuration parameter. If you omit this property, the amount of logspace is set by the SESSION_LIMIT_LOGSPACE configuration parameter. If the SESSION_LIMIT_LOGSPACE configuration parameter is also not set, the maximum amount of log space that a session can use for individual transactions is 2147483648 KB.
session_limit_memory	The maximum amount of memory that a session can allocate.	The existing **session_limit_memory** property value is replaced. The value must be 20480 - 2147483648 KB. Values are specified in KB, MB, or GB. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_memory** property takes precedent over the value of the SESSION_LIMIT_MEMORY configuration parameter. If this property is not set, the amount of memory is set by the SESSION_LIMIT_MEMORY configuration parameter. If the SESSION_LIMIT_MEMORY configuration parameter is also not set, the maximum amount of memory that a session can allocate is 2147483648 KB.
session_limit_tempspace	The maximum amount of temporary table space that a session can allocate.	The existing **session_limit_tempspace** property value is replaced. The value must be 20480 - 2147483648 KB. Values are specified in KB, MB, or GB. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_tempspace** property takes precedent over the value of the SESSION_LIMIT_TEMPSPACE configuration parameter. If this property is not set, the amount of temporary table space is set by the SESSION_LIMIT_TEMPSPACE configuration parameter. If the SESSION_LIMIT_TEMPSPACE configuration parameter is also not set, the maximum amount of temporary table space that a session can allocate is 2147483648 KB.

Element	Description	Key Considerations
session_limit_txn_time	The maximum amount of time that a transaction can run in a session.	The existing **session_limit_txn_time** property value is replaced. The value must be 60 - 20000000000. Values are in seconds. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user. The value of the **session_limit_txn_time** property takes precedent over the value of the SESSION_LIMIT_TXN_TIME configuration parameter. If you omit this property, the amount of time is set by the SESSION_LIMIT_TXN_TIME configuration parameter. If the SESSION_LIMIT_TXN_TIME configuration parameter is also not set, the maximum amount of time that a transaction can run in a session is 20000000000 seconds.
tenant_limit_space	The maximum amount of storage space on disk to a tenant database. When the limit is reached, subsequent operations that require more disk space are rejected.	The existing **tenant_limit_space** property value is replaced. The value must be 1048576 - 1717986918400 KB (1 GB - 200 TB). Values are specified in KB, MB, GB, or TB. The value of the **tenant_limit_space** property takes precedent over the value of the TENANT_LIMIT_SPACE configuration parameter. If you omit this property, the amount of space is set by the TENANT_LIMIT_SPACE configuration parameter. If the TENANT_LIMIT_SPACE configuration parameter is also not set, the maximum amount of storage space available to a tenant user is 1717986918400 KB.
tenant_limit_memory	The maximum amount of shared memory for all sessions that are connected to the tenant database. When the limit is exceeded, the session that is using the most shared memory is terminated.	The existing **tenant_limit_memory** property value is replaced. The value must be 102400 - 2147483648 KB (100 MB - 2 TB). Values are specified in KB, MB, GB, or TB. The value of the **tenant_limit_memory** property takes precedent over the value of the TENANT_LIMIT_MEMORY configuration parameter. If you omit this property, the amount of memory is set by the TENANT_LIMIT_MEMORY configuration parameter. If the TENANT_LIMIT_MEMORY configuration parameter is also not set, the maximum amount of memory available to a tenant session is 2147483648 KB.
tenant_limit_connections	The maximum number of connections to a tenant database. When the limit is reached, subsequent connection requests are rejected.	The existing **tenant_limit_connections** property value is replaced. The value must be 1 - 65536. The value of the **tenant_limit_connections** property takes precedent over the value of the TENANT_LIMIT_CONNECTIONS configuration parameter. If you omit this property, the number of connections is set by the TENANT_LIMIT_CONNECTIONS configuration parameter. If the TENANT_LIMIT_CONNECTIONS configuration parameter is also not set, the maximum number of connections for a tenant database is 65536.
vpclass	The name of the virtual processor class for running tenant-database session threads.	The **vpclass** property value is replaced. If you omit this property, session threads are run on CPU virtual processors. Values must be 8 characters or fewer. A maximum of 200 tenant virtual processor classes can be created. If the virtual processor class name is unique, you create a new tenant virtual processor class. If the virtual processor class name exists, the tenant database shares the class with other tenant databases. When a tenant virtual processor is dropped, the virtual processor class ID resources are not freed until the database server is restarted.

Usage

You must be user **informix** or a DBSA user, or you must have the TENANT privilege to run this command.

The changes to the database properties take effect for new sessions.

The following statement updates the properties of the tenant database that is named **company_A**:

```
EXECUTE FUNCTION task('tenant update', 'company_A',
                      '{dbspace:"company_A_dbs4,company_A_dbs5",
                        sbspace:"company_A_sbs3",
                        vpclass:"tvp_B",
                        session_limit_txn_time:"120"}'
);
```

The tenant database gains two dbspaces and an sbspace, the virtual processor class is changed, and the time limit on transactions becomes 120 seconds.

Related reference:
TENANT_LIMIT_SPACE configuration parameter
tenant create argument: Create a tenant database (SQL Administration API)
tenant drop argument: Drop a tenant database (SQL Administration API)
SESSION_LIMIT_LOCKS configuration parameter
SESSION_LIMIT_MEMORY configuration parameter
SESSION_LIMIT_TEMPSPACE configuration parameter
SESSION_LIMIT_LOGSPACE configuration parameter
SESSION_LIMIT_TXN_TIME configuration parameter
TENANT_LIMIT_CONNECTIONS configuration parameter

onstat -g ses command: Print session-related information
Related information:
Multitenancy
DB_LOCALE environment variable

Appendixes

- Database server files
  Database server files are created in default directories, or in a directory that the relevant configuration parameter specifies. A database administrator might need to edit or examine the content of files that are used by the database server.
- Troubleshooting errors
  Occasionally, a series of events causes the database server to return unexpected error codes.
- Event Alarms
  The database server provides a mechanism for automatically triggering administrative actions based on an event that occurs in the database server environment. This mechanism is the event-alarm feature.
- Messages in the database server log
  Unnumbered messages are printed in the database server message log (online.log). The error messages include corrective actions.
- Limits in Informix
  This topic lists the system-level and table-level parameter limits, the system defaults, and the access capabilities of Informix.

Database server files

Database server files are created in default directories, or in a directory that the relevant configuration parameter specifies. A database administrator might need to edit or examine the content of files that are used by the database server.

- Table 1 lists database server files that you might need to look at, copy, edit, move, or delete (except where noted).
- Table 2 lists database server files that are for internal use only. You must not edit, move, or delete these files.

Table 1. Database server files that you can use. This table lists the files that you might refer to or use when you configure and use the database server.

File name	Directory	Purpose	Created
af.*xxx* *xxx* identifies a specific assertion failure	$INFORMIXDIR/tmp (UNIX) %INFORMIXDIR%\tmp (Windows) Specified by DUMPDIR configuration parameter	Assertion-failure information	By the database server
ac_msg.log	/tmp (UNIX) %INFORMIXDIR%\etc (Windows)	The message log for the **archecker** utility	By the database server
ac_config.std	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Template for **archecker**parameter values	By the database server
bar_act.log	/tmp (UNIX) %INFORMIXDIR%\etc (Windows) Specified by the BAR_ACT_LOG configuration parameter	ON-Bar activity log	By ON-Bar
bar_dbug.log	/usr/informix/ (UNIX) \usr\informix\ (Windows) Specified by the BAR_DEBUG_LOG configuration parameter	ON-Bar debug log	By ON-Bar
bldutil.*process_id* (UNIX) bldutil.out (Windows)	/tmp (UNIX) \tmp (Windows)	Error messages about building the **sysutils** database	By the database server
buildsmi.out (UNIX) buildsmi_out.*%INFORMIXSERVER%* (Windows)	/tmp (UNIX) %INFORMIXDIR%\etc (Windows)	Error messages about building the **sysmaster** database	By the database server
concdr.sh	$INFORMIXDIR /etc/conv (UNIX) %INFORMIXDIR% \etc\conv (Windows)	Converts the **syscdr** database during an upgrade	By the database server
core (UNIX)	Directory from which the database server was started	Core dump	By the database server
gcore.*xxx* (UNIX)	$INFORMIXDIR/tmp (UNIX) %INFORMIXDIR%\tmp (Windows) Specified by DUMPDIR configuration parameter	Assertion failure information	By the database server
.informix (UNIX)	User's home directory	Set personal environment variables	By the user

File name	Directory	Purpose	Created
informix.rc (UNIX)	$INFORMIXDIR/etc	Set default environment variables for all users	By the database administrator
InstallServer.log (Windows)	C:\temp	Database server installation log	By the database server
ixbar.*servernum*	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Emergency boot file that is used in a cold restore	By ON-Bar
jvp.log	/urs/informix Specified by JVPLOGFILE configuration parameter	Messages from the Java™ virtual processor	By the database server
.jvpprops	urs/informix/extend/krakatoa Specified by JVPPROPFILE configuration parameter	Template for Java VP properties	During installation
oncfg_ *servername.servernum*	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Configuration information for whole-system restores by ON-Bar	By the database server
online.log	$INFORMIXDIR/tmp (UNIX) %INFORMIXDIR% (Windows) Specified by the MSGPATH configuration parameter	Database server message log, which contains error messages and status information	By the database server
onconfig	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Configuration information	By the database administrator or the database server administrator
onconfig.std	$INFORMIXDIR/etc (UNIX)	Template for configuration parameter values Important: Do not move, modify, or delete the onconfig.std file unless instructed to do so by Software Support. However, you may make a copy of the onconfig.std file to create a customized configuration file, and then move the copy to another location.	During installation
onsnmp.*servername*	/tmp (UNIX) \tmp (Windows)	Log file that the **onsnmp** subagent uses	By the **onsnmp** utility
onsrvapd.log	/tmp (UNIX) \tmp (Windows)	Log file for the database server daemon **onsrvapd**	By the **onsnmp** utility
psm_act.log	/tmp (UNIX) %INFORMIXDIR%\etc (Windows) Specified by the PSM_ACT_LOG configuration parameter	Log file for Informix® Primary Storage Manager	By ON-Bar
pua.map	$INFORMIXDIR/gls/etc (UNIX) %INFORMIXDIR%\gls\etc\(Windows)	Mapping file for displaying characters in Unicode Private-Use Area (PUA) ranges.	By the user
revcdr.sh (UNIX) revcdr.bat (Windows)	$INFORMIXDIR/etc/conv (UNIX) %INFORMIXDIR%\etc\conv(Windows)	Reverts the **syscdr** database to an earlier format	By the database server
shmem.*xxx*	$INFORMIXDIR/tmp (UNIX) %INFORMIXDIR%\tmp (Windows) Specified by DUMPDIR configuration parameter	Assertion-failure information	By the database server
sm_versions.std	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Identifies storage manager in use	During installation
snmpd.log	/tmp (UNIX) \tmp (Windows)	Log file for the SNMP master agent, **snmpdm**	By **onsnmp**
sqlhosts.*servername*	$INFORMIXDIR/etc %INFORMIXDIR%\etc (Windows)	Connection information	During installation; modified by the database server administrator The file extension is the server name (the default extension is ol_informix*version*)
sqlhosts.std	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Template for connection information	During installation

Table 2. Database server files that are for internal use only. This table lists the files that are required by the database server.
Important: Do not move, modify, or delete these files unless instructed to do so by Software Support.

File name	Directory	Purpose	Created
.conf.*dbservername*	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	The **onsnmp** utility uses this file to obtain the database server configuration	By the database server
illlsrra.*xx*	$INFORMIXDIR/lib (UNIX) %INFORMIXDIR%\lib (Windows)	Shared libraries for the database server and some utilities	By installation procedure
INFORMIXTMP	/INFORMIXTMP (UNIX) \%INFORMIXDIR% (Windows)	Temporary directory for internal files	By the database server
.inf.servicename	/INFORMIXTMP (UNIX) drive:\INFORMIXTMP (Windows)	Connection information	By the database server

File name	Directory	Purpose	Created
.infos.*dbservername*	$INFORMIXDIR/etc (UNIX) %INFORMIXDIR%\etc (Windows)	Connection information	By the database server
.infxdirs	/INFORMIXTMP (UNIX) *drive*:\INFORMIXTMP (Windows)	Database server discovery file that **onsnmp** uses	By the database server
JVM_*vpid*	Specified by JVPLOG configuration parameter	Messages that the Java virtual machine generates	By the Java virtual machine
servicename.exp	/INFORMIXTMP (UNIX) *drive*:\INFORMIXTMP (Windows)	Connection information	By the database server
servicename.str	/INFORMIXTMP (UNIX) *drive*:\INFORMIXTMP (Windows)	Connection information	By the database server
VP.*servername.nnx*	/INFORMIXTMP (UNIX) *drive*:\INFORMIXTMP (Windows)	Connection information	By the database server

Related tasks:
Setting local environment variables for utilities
Related reference:
onstat -c command: Print ONCONFIG file contents
onstat -g cfg command: Print the current values of configuration parameters
onconfig Portal: Configuration parameters by functional category
The oninit utility
Related information:
Database server configuration

Troubleshooting errors

Occasionally, a series of events causes the database server to return unexpected error codes.

You can use the following diagnostic tools to gather information for troubleshooting errors:

- **onmode -I**
- tracepoints
- The **ifxcollect** tool

- Collecting Diagnostics using onmode -I
- Creating Tracepoints
- Collecting data with the ifxcollect tool
  You can use the **ifxcollect** tool to collect diagnostic data if necessary for troubleshooting a specific problem, such as an assertion failure. You can also specify options for transmitting the collected data via the File Transfer Protocol (FTP). .

Collecting Diagnostics using onmode -I

To help collect additional diagnostics, you can use **onmode -I** to instruct the database server to perform the diagnostics collection procedures that the *IBM® Informix® Administrator's Guide* describes. To use **onmode -I** when you encounter an error number, supply the *iserrno* and an optional session ID. For more information about **onmode**, see The onmode utility.

Related reference:
Creating Tracepoints
Collecting data with the ifxcollect tool

Creating Tracepoints

Tracepoints are useful in debugging user-defined routines written in C. You can create a user-defined tracepoint to send special information about the current execution state of a user-defined routine.

Each tracepoint has the following parts:

- A *trace* groups related tracepoints together so that they can be turned on or off at the same time.
  You can either use the built-in trace called **_myErrors** or create your own. To create your own trace, you insert rows into the **systracees** system catalog table.
- A *trace message* is the text that the database server sends to the tracing-output file.
  You can store internationalized trace messages in the **systracemsgs** system catalog table.
- A *tracepoint threshold* determines when the tracepoint executes.

By default, the database server puts all trace messages in the trace-output file in the **tmp** directory with the following filename:

session_num.trc

For more information on tracing user-defined routines, see the *IBM® Informix® DataBlade API Programmer's Guide*.

Related concepts:
Collecting Diagnostics using onmode -I
Related reference:
Collecting data with the ifxcollect tool

Collecting data with the ifxcollect tool

You can use the **ifxcollect** tool to collect diagnostic data if necessary for troubleshooting a specific problem, such as an assertion failure. You can also specify options for transmitting the collected data via the File Transfer Protocol (FTP). .

The **ifxcollect** tool is in the **$INFORMIXDIR/bin** directory. Output files that **ifxcollect** commands generate are in the **$INFORMIXDIR/isa/data** directory.

The type of data that is collected per category and subcategory is in predefined XML files in the **$INFORMIXDIR/isa/** directory. These XML files can be modified to add or remove specific commands.

Important: The XML files can contain commands that override the options that are specified for data collection. For example, an XML file might contain sleep commands that override the **-d** option with a shorter number of seconds; or an XML file might contain a call to **onstat -z**.

Syntax

```
>>-ifxcollect-- -c - category -- -s - subcategory --------------->

>--+-------------------------+--+-----------------+--+-----+------>
   '- -r - number_of_times -'  '- -d - seconds -'  '- -y '

>--+-----------+--+-----+-------------------------------------><
   +- -V-------+  +- -f-+
   '- -version-'  +- -e-+
                  +- -p-+
                  +- -m-+
                  +- -l-+
                  +- -u-+
                  '- -w-'
```

Table 1. Options for data collection

Element	Description	Key Considerations
-c *category*	Tells the server to collect data in the specified category.	You must specify the category of data to collect.
-s *category*	Tells the server to collect data in the specified subcategory.	You must specify the subcategory of data to collect.
-r *number of times*	Specifies the number of times to repeat data collection.	Optional. The default value is 1.
-d *number of seconds*	Specifies the number of times to pause between collection operations.	Optional. The default value is 0.
-y	Causes the database server to automatically respond yes to all prompts.	Optional.
-V	Displays the software version number and the serial number.	Optional. See Obtaining utility version information
-version	Displays the build version, host, operating system, number and date, and the GLS version.	Optional. See Obtaining utility version information

Table 2. FTP options if you also transmitting data

Element	Description	Key Considerations
-f	FTP the entire collection	Required for the transmission of data.
-e *email address*	Email address	Required for the transmission of data.
-p *the PMR number*	PMR number	Required for the transmission of data.
-m *machine name*	Machine to connect to	Required for the transmission of data.
-l *directory*	Directory that contains the data	Required for the transmission of data.
-u *user name*	User name for the FTP	Required for the transmission of data.

Element	Description	Key Considerations
-w *password*	Password for the FTP	Required for the transmission of data.

Usage

The following table shows the combination of categories and subcategories that you can use in your commands.

Table 3. Category and subcategory combinations

Category and subcategory	Explanation
`-c ids -s general`	Collects general data for issues that are related to all Informix products
`-c af -s general`	Collects general data for assertion failures
`-c er -s general`	Collects general data for Enterprise Replication
`-c er -s init`	Collects general data for Enterprise Replication initialization issues
`-c performance -s general`	Collects data for performance issues
`-c performance -s cpu`	Collects data for CPU utilization issues
`-c onbar -s archive_failure`	Collects data for **onbar** archive failures
`-c onbar -s restore_failure`	Collects data for **onbar** restore failures
`-c ontape -s archive_failure`	Collects data for **ontape** archive failures
`-c ontape -s restore_failure`	Collects data for **ontape** restore failures
`-c connection -s failure`	Collects data for connection failures
`-c connection -s hang`	Collects data for connection hangs
`-c cust -s prof`	Collects customer profile information

To view all **ifxcollect** utility command options, type `ifxcollect` at the command prompt.

Examples

To collect information for a general assertion failure, run this command:

```
ifxcollect -c af -s general
```

To collect information for a performance problem that is related to CPU utilization, run this command:

```
ifxcollect -c performance -s cpu
```

To include FTP information, specify the additional information as shown in this example:

```
-f -e user_name@company_name.org -p 9999.999.999
-f -m machine -l /tmp -u user_name -w password
```

Related concepts:
Collecting Diagnostics using onmode -I
Related reference:
Creating Tracepoints

Event Alarms

The database server provides a mechanism for automatically triggering administrative actions based on an event that occurs in the database server environment. This mechanism is the event-alarm feature.

Events can be informative (for example, `Backup Complete`) or can indicate an error condition that requires your attention (for example, `Unable to Allocate Memory`).

- Using ALARMPROGRAM to Capture Events
- Events in the ph_alert Table
  All event alarms that are generated are inserted in the **ph_alert** table in the **sysadmin** database.
- Event Alarm Parameters
  Event alarms have five parameters that describe each event.
- Event alarm IDs
  The class ID for event alarms indicates the type of event. The event ID indicates the specific event.
- Connection Manager event alarm IDs
  The class ID for event alarms indicates the type of event. The event ID indicates the specific event.

Related reference:
ALARMPROGRAM configuration parameter
ALRM_ALL_EVENTS configuration parameter

Using ALARMPROGRAM to Capture Events

On UNIX, use the **alarmprogram.sh** and on Windows, use the **alarmprogram.bat** shell script, for handling event alarms and starting automatic log backups. For the setup instructions, see ALARMPROGRAM configuration parameter.

To automate logical-log backups only, two ready-made scripts are provided: **log_full.**[**sh**|**bat**] and **no_log.**[**sh**|**bat**]. Set ALARMPROGRAM to the full path name of the script. For information, see ALARMPROGRAM configuration parameter.

- Setting ALRM_ALL_EVENTS
- Writing Your Own Alarm Script
- Customizing the ALARMPROGRAM Scripts
  You can customize the ALARMPROGRAM scripts based on your environment.
- Precautions for Foreground Operations in Alarm Scripts
  To ensure continuous server availability, do not run certain foreground operations in an alarm script.
- Interpreting event alarm messages
  Some of the events that the database server reports to the message log trigger the alarm program. The class messages indicate the events that the database server reports.

Setting ALRM_ALL_EVENTS

You can set ALRM_ALL_EVENTS to specify whether ALARMPROGRAM runs for all events that are logged in the MSGPATH or only specified noteworthy events (events greater than severity 1).

Writing Your Own Alarm Script

Alternatively, you can write your own shell script, batch file, or binary program that contains the event-alarm parameters. When an event occurs, the database server invokes this executable file and passes it the event-alarm parameters (see Table 1). For example, your script can use the **_id** and **_msg** parameters to take administrative action when a table failure occurs. Set ALARMPROGRAM to the full pathname of this executable file.

Related reference:
ALARMPROGRAM configuration parameter

Customizing the ALARMPROGRAM Scripts

You can customize the ALARMPROGRAM scripts based on your environment.

The mail utility must already be present.

Follow these steps to customize the **alarmprogram.[sh|bat]** script. You can use **alarmprogram.[sh|bat]** instead of **log_full.[sh|bat]** to automate log backups.

To customize the ALARMPROGRAM scripts:

1. Change the value of ADMINMAIL to the email address of the database server administrator.
2. Change the value of PAGERMAIL to the pager service email address.
3. Set the value of the parameter MAILUTILITY.
   - UNIX: **/usr/bin/mail**
   - Windows: **$INFORMIXDIR/bin/ntmail.exe**
   - Linux: **/usr/lib/sendmail -t**
4. To automatically back up logical logs as they fill, change BACKUP to `yes`. To stop automatic log backups, change BACKUP to any value other than `yes`.
5. In the ONCONFIG file, set ALARMPROGRAM to the full pathname of **alarmprogram.[sh|bat].**
6. Restart the database server.

Alarms with a severity of 1 or 2 do not write any messages to the message log nor send email. Alarms with severity of 3 or greater send email to the database administrator. Alarms with severity of 4 and 5 also notify a pager via email.

Related reference:
ALARMPROGRAM configuration parameter

Precautions for Foreground Operations in Alarm Scripts

To ensure continuous server availability, do not run certain foreground operations in an alarm script.

When the server invokes an alarm script, the server sometimes waits for the script to complete before proceeding. For example:

- When an alarm is invoked because of a fatal error, the server waits for the script to finish writing information to the error log. In certain situations, alarm events 5 and 6 are run in the foreground.
- Some Enterprise Replication event alarms run in the foreground, such as event alarms 31, 34, 37, and 39.

Because the server might need to wait for the alarm program script to finish, do not run the following operations in the foreground in an alarm script:

- An onmode command that forces user connections off the server such as `onmode -u` or `onmode -yuk`. These kinds of onmode commands can cause a deadlock between the server and the alarm script because the server might wait for the alarm script to complete while the alarm script that executed the onmode command waits for the user sessions to shut down, and one of those sessions is running the alarm script itself.
- Operations that might take a long time to complete or that have a highly variable run time. Operations that take a long time to complete can cause the server to appear as if it is not responding while the operation is running.

If you need to run the above operations in an alarm script, run them in the background using one of the following operating system utilities:

On UNIX: Use the nohup utility. For example, `nohup onmode -yuk &` instructs nohup to continue running the command even if its parent terminates and the ampersand, *&*, runs the command in the background so it will not block execution of the alarm program script itself.

On Windows: Use the start utility with the /B flag. For example, `start /B onmode -yuk`.

Interpreting event alarm messages

Some of the events that the database server reports to the message log trigger the alarm program. The class messages indicate the events that the database server reports.

The database server reports a nonzero exit code in the message log. In the alarm program, set the EXIT_STATUS variable to `0` for successful completion and to another number for a failure.

For example, if a thread attempts to acquire a lock, but the maximum number of locks are already in use, the database server writes the following message to the message log:

```
10:37:22  Checkpoint Completed:  duration was 0 seconds.
10:51:08  Lock table overflow - user id 30032, rstcb 10132264
10:51:10  Lock table overflow - user id 30032, rstcb 10132264
10:51:12  Checkpoint Completed:  duration was 1 seconds.
```

When the database server runs the **alarmprogram.sh** or **alarmprogram.bat** program, or your alarm program, the database server generates a message that describes the severity and class of the event. If the severity is greater than 2, the message takes the following format:

Action	Message
A reasonably severe server event	**Severity: 3** **Class ID: 21** **Class msg: Database server resource overflow: 'Locks'.** **Specific msg: Lock table overflow - user id 30032, rstcb 10132264** **See Also: # optional message** **Event ID: 21005**
The message that appears at the end of each e-mailed message	**This e-mail was generated by the server ALARMPROGRAM script on *servername* because something untoward just happened to *eventname*.**

Events in the ph_alert Table

All event alarms that are generated are inserted in the **ph_alert** table in the **sysadmin** database.

You can query the **ph_alert** table on local or remote server to view the recent event alarms for that server. You can write SQL scripts based on the **ph_alert** table to handle event alarms instead of using the scripts controlled by the ALARMPROGRAM configuration parameter.

By default, alerts remain in the **ph_alert** table for 15 days before being purged.

Example

The following example shows an event alarm in the **ph_alert** table:

```
SELECT * FROM ph_alerts WHERE alert_object_type=ALARM;

id                  34
alert_task_id       18
alert_task_seq      10
alert_type          INFO
alert_color         YELLOW
alert_time          2010-03-08 12:05:48
alert_state         NEW
```

```
alert_state_chang+  2010-03-08 12:05:48
alert_object_type   ALARM
alert_object_name   23
alert_message       Logical Log 12 Complete, timestamp: 0x8e6a1.
alert_action_dbs    sysadmin
alert_action
alert_object_info   23001
```

Related reference:
The ph_alert Table
Event Alarm Parameters

Event Alarm Parameters

Event alarms have five parameters that describe each event.

The following table lists the parameters that are part of event alarm.

Table 1. Event Alarm Parameters

Parameter	Description	Data Type
severity	The severity of the event.	integer
class_id	A numeric identifier that classifies the type of event that has occurred.	integer
class_msg	A brief messages that describes the classification of the event.	string
specific_msg	Specific messages that describes the event that occurred.	string
see_also	A reference to a file that contains additional information about the event.	string
uniqueid	A unique event identifier for the specific message.	bigint

Event Severity

An event severity code is a numeric indication of the seriousness of an event. Every event that is included in the message log contains a severity code. The event severity code is the first parameter that is sent to the alarm program. In the **ph_alert** table, the event severity is reflected by a combination of the alert color and the alert type. The event severity codes are listed in the following table.

Table 2. Event Severity Codes

Severity	Description
1	**Not noteworthy**. The event (for example, date change in the message log) is not reported to the alarm program unless ALRM_ALL_EVENTS configuration parameter is enabled. In the **ph_alert** table, the alert color is GREEN and the alert type is INFO.
2	**Information**. No error has occurred, but some routine event completed successfully (for example, checkpoint or log backup completed). In the **ph_alert** table, the alert color is YELLOW and the alert type is INFO.
3	**Attention**. This event does not compromise data or prevent the use of the system; however, the event warrants your attention. For example, one chunk of a mirrored pair goes down. An email is sent to the system administrator. In the **ph_alert** table, the alert color is YELLOW and the alert type is WARNING.
4	**Emergency**. Something unexpected occurred that might compromise data or access to data. For example an assertion failure, or **oncheck** reports data corrupt. Take action immediately. The system administrator is paged when this event severity occurs. In the **ph_alert** table, the alert color is RED and the alert type is ERROR.
5	**Fatal**. Something unexpected occurred and caused the database server to fail. The system administrator is paged when this event severity occurs. In the **ph_alert** table, the alert color is RED and the alert type is ERROR.

Class ID

The class ID is an integer that identifies the event that causes the database server to run your alarm program. The class ID is the second parameter that the database server displays in your alarm program.

The class ID is stored in the **alert_object_name** column in the **ph_alert** table.

Class Message

The class message is a text message briefly describes, or classifies, the event that causes the database server to run your alarm program. The class messages is the third parameter that the database server displays in your alarm program.

Specific Message

The specific message is a text messages the describes in more detail the event that causes the database server to run your alarm program. The specific message is the fourth parameter that the database server displays in your alarm program. For many alarms, the text of this message is the same as the message that is written to the message log for the event.

The specific message is stored in the **alert_message** column in the **ph_alert** table.

See Also Paths

For some events, the database server writes additional information to a file when the event occurs. The path name in this context refers to the path name of the file where the database server writes the additional information.

Event ID

The event ID is a unique number for each specific message. You can use the event ID in custom alarm handling scripts to create responses to specific events.

The event ID is stored in the **alert_object_info** column in the **ph_alert** table.

Related concepts:
Events in the ph_alert Table
Related reference:
STORAGE_FULL_ALARM configuration parameter
SHMVIRT_ALLOCSEG configuration parameter

Event alarm IDs

The class ID for event alarms indicates the type of event. The event ID indicates the specific event.

The following table lists event alarm IDs and messages or where to find more information. Many alarms have additional explanations and user actions. Many of the issues that trigger event alarms also result in messages in the online message log. The location of the message log is specified by the MSGPATH configuration parameter.

Table 1. Event Alarms

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1001	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Page allocation error on '*object*'	The database server detected an inconsistency during the allocation of pages to a table or index. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Dependent upon the nature of the problem found. **User action:** Review the online.log file for appropriate action. You will need to run the **oncheck** utility on the *'dbname:"owner".tabname'* identified in the message. Occasionally, the database server automatically resolves the problem, and this resolution is identified in the online.log file.
Class ID: 1 Event ID: 1002	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Row allocation error on '*object*'	The database server detected an inconsistency during the allocation of a row to a table or index. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Dependent upon the nature of the problem found. **User action:** Review the online.log file for appropriate action. You will need to run the **oncheck** utility on the *'dbname:"owner".tabname'* identified in the message. Occasionally, the database server automatically resolves the problem, and this resolution is identified in the online.log file.
Class ID: 1 Event ID: 1003	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Slot allocation error for '*object*'	The database server detected an inconsistency during row processing. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Dependent upon the nature of the problem found. **User action:** Review the online.log file for appropriate action. You will need to run the **oncheck** utility on the *'dbname:"owner".tabname'* identified in the message. Occasionally, the database server automatically resolves the problem, and this resolution is identified in the online.log file.
Class ID: 1 Event ID: 1004	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error prevented the database server to find the next possible data page in this tblspace.	The database server detected an inconsistency in a bitmap page during the allocation of a row to a table or index. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Dependent upon the nature of the problem found. **User action:** Review the online.log file for appropriate action. Run the **oncheck** utility on the *'dbname:"owner".tabname'* identified in the message.

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1005	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Dropping wrong TBLSpace, requested *tblspace_name* != actual *tblspace_name*	The database server detected a mismatch between the requested and existing tables while attempting to drop a table. No table was dropped. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Online **User action:** Review the online.log file for appropriate action. Run the **oncheck** utility on the *'dbname:"owner".tabname'* identified in the message.
Class ID: 1 Event ID: 1006	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error which may have been caused due to data corruption prevented the database server from altering the bitmap pages for this partition.	The database server encountered a possible data corruption error during a table or index operation to alter bitmap pages. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Online **User action:** Review the online.log file for appropriate action. Run the **oncheck** utility on the *'dbname:"owner".tabname'* identified in the message.
Class ID: 1 Event ID: 1007	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: [3] An internal error which may have been caused due to corrupted bitmap pages as the database server is still in the process of converting them.	The database server encountered an incomplete modification of internal bitmap pages during a table or index operation to alter bitmap pages. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Online **User action:** Note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1008	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error which may have been caused due to unconverted bitmap pages.	The database server encountered a situation where the modification of the internal bitmap pages has not yet completed. during a table or index operation to alter bitmap pages. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Online. **User action:** Note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1009	4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Page Check Error in *object*	The database server detected inconsistencies while checking a page that was being read into internal buffers. **Online log:** Assertion failure or assertion warning with a description of the problem. **Server state:** Online or offline, depending on how serious the problem is. **User action:** Follow the suggestions in the online log. Typically, run the **oncheck -cD** command on the table mentioned in the class message or on the database.
Class ID: 1 Event ID: 1010	4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Bad rowid *rowid*	The database server detected an invalid row ID. **Online log:** Assertion warning with a description of where the problem was found. **Server state:** Online. **User action:** Repair the index by running the **oncheck -cI** command on the table mentioned in the class message or on the database.
Class ID: 1 Event ID: 1011	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Closing TBLSpace *tblspace_name*	The database server determined that the table or index is closed. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Online. **User action:** None. The database server will correct the problem automatically.
Class ID: 1 Event ID: 1012	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Cannot recreate index *index_name* for partnum *partition_number*, iserrno = *error_number*	The database server encountered an error that prevents recreating an index. **Online log:** Assertion Failure or Assertion Warning with problem details. **Server state:** Online. **User action:** Review the online.log file for the index information, and then drop and recreate the index manually.

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1013	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to initialize the type of set read operation.	The database server was unable to initialize internal data structures for a set read operation. **Online log:** Assertion Warning with database and table details. **Server state:** Online. **User action:** Review the online.log file for ISAM error codes, and table and database information. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1014	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to read records from the tblspace's pages	The database server encountered an internal error when reading records from a table or index. **Online log:** Assertion Warning with database and table details. **Server state:** Online. **User action:** Review the online.log file for table and database information. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1015	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to read the current record.	The database server encountered an internal error when reading records from a table or index. **Online log:** Assertion Warning with database and table details. **Server state:** Online. **User action:** Review the online.log file for table and database information. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1016	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to initialize the set read buffer.	An internal error was triggered when the database server attempted to initialize a set read buffer. **Online log:** Assertion Warning with database and table details. **Server state:** Online. **User action:** Review the online.log file for table and database information. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1017	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to set the new mode on the bitmap page.	An internal error occurred during the conversion of a bitmap page from an earlier version of the database server. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** None.
Class ID: 1 Event ID: 1018	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was attempting to convert bitmap pages to the correct format.	The database server was unable to correct an error which occurred during a bitmap page conversion. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** Note all circumstances, review the online.log file for additional information, and contact Software Support.
Class ID: 1 Event ID: 1019	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to modify the bitmap pages during light append operation.	The database server encountered an internal error during a light append operation and could not locate the required bitmap page. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1020	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to perform light scan operation.	The database server encountered an internal error while performing a light scan operation. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1021	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while the database server was trying to perform light scan I/O operation.	The database server encountered an internal error while performing a light scan operation. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1022	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to validate light append buffer.	The database server encountered an internal error while performing a light scan operation. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1023	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to write the next record to the page in the light append buffer.	The database server encountered an internal error while performing a light scan operation. **Online log:** Assertion Warning with problem details. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1024	4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to open a light append for a tblspace.	The database server encountered an internal error during a light append operation on a tblspace. **Online log:** Assertion failure. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1025	4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to load the first bitmap page for a light append operation.	The database server encountered an internal error during a light append operation. **Online log:** `Assertion Failure : Light Append(Redo/Undo): Can't find bitmap page` **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1026	4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to write the cached bitmap pages for a light append operation.	The database server encountered an internal error during a light append operation. **Online log:** Assertion failure. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1027	2	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal deadlock database condition was caught by the Lock Manager in the database server.	The database server detected an internal deadlock database condition. **Online log:** Assertion warning identifying the databases and tables involved in the deadlock. **Server state:** Online. **User action:** None.
Class ID: 1 Event ID: 1028	2	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal deadlock database condition was caught by the Lock Manager in the database server.	The database server detected an internal deadlock database condition. **Online log:** Assertion warning identifying the databases and tables involved in the deadlock. **Server state:** Online. **User action:** None.
Class ID: 1 Event ID: 1029	4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to map the logical page number in the tblspace to its physical location in the chunk.	The database server could not access a table because of an inconsistency between the physical page and its logical page number. **Online log:** Assertion failure with the page information. **Server state:** Online. **User action:** Run the **oncheck -cDI** command on the table mentioned in the class message or on the database, fix any issues reported, and then try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1030	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to allocate the alter information.	The database server encountered an internal error when attempting to read an internal disk structure. **Online log:** Assertion Warning with problem details and table and database information. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1031	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to prepare the list of operations to be performed on a compressed row.	The database server encountered an internal error when it tried to create an internal operations list to transform a compressed version of a row to an uncompressed version of the latest row. **Online log:** Assertion Warning with problem details and table and database information. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1032	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to insert an operation in the list of operations based on the offset in the row where the new operation points to.	The database server encountered an internal error when it tried to create an internal operations list to transform a compressed version of a row to an uncompressed version of the latest row. **Online log:** Assertion Warning with problem details and table and database information. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1033	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it detected an inconsistency with the operation list.	The database server encountered an internal error when it tried to create an internal operations list to transform a compressed version of a row to an uncompressed version of the latest row. **Online log:** Assertion Warning with problem details and table and database information. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1034	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to free the partition header page.	The database server encountered an internal error when attempting to free the header page for a partition. The database server did not free the header page. **Online log:** Assertion Warning with problem details, table and database information, and a specific **oncheck** command to run. **Server state:** Online. **User action:** Review the online.log file for information and run the specified **oncheck** command.
Class ID: 1 Event ID: 1035	3 or 4	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to validate the partition header page.	The database server cannot access a table because of a validation error for the tblspace page. **Online log:** Assertion with details about the table. **Server state:** Online. **User action:** Examine the online log for information about the specified table. Run the **oncheck -pt** command on the table or on the database and correct any errors found. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1036	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to update the special columns list during an alter table command processing.	The database server encountered an internal error when processing the special columns list associated with a table while the table was being altered. **Online log:** Assertion Warning with problem details, table and database information, and a specific **oncheck** command to run. **Server state:** Online. **User action:** Review the online.log file for information and run the specified **oncheck** command.
Class ID: 1 Event ID: 1037	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to log the completion of the alter and remove the associated version information from the tblspace's header page.	The database server encountered an internal error when attempting to alter a table. **Online log:** Assertion Warning with problem details, table and database information, and a specific **oncheck** command to run. **Server state:** Online. **User action:** Review the online.log file for information and run the specified **oncheck** command.

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1038	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it detected a buffer inconsistency.	The database server encountered an internal error during a consistency check of the internal buffers it was manipulating. **Online log:** Assertion Warning with problem details, table and database information, and a specific **oncheck** command to run. **Server state:** Online. **User action:** Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1039	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to construct a forwarded row into a single tuple.	The database server encountered an internal error when processing rows. **Online log:** Assertion Warning with problem details, and table and database information. **Server state:** Online. **User action:** Review the online.log file for more information. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1040	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to read the data from a partition into the set read buffer.	The database server encountered a corrupt record during the process of reading data and was unable to retrieve the data. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file. Some instances of this error require the attention of Software Support.
Class ID: 1 Event ID: 1041	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to read the data row for a given rowid.	The database server encountered a corrupt record during the process of reading data from an index and was unable to retrieve the data. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for information and run the recommended **oncheck** command. Some instances of this error require the attention of Software Support.
Class ID: 1 Event ID: 1042	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to alter the row in memory to the latest schema.	The database server encountered an internal error while trying to convert an old version of a record to the latest version of the record in an altered table. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for more information. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 1 Event ID: 1043	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to undo the alter of a bitmap page.	The database server encountered an internal error while trying to revert an operation that had altered an internal bitmap page. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 1 Event ID: 1044	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to undo the addition of special column descriptors from the tblspace's header page.	The database server encountered an internal error while trying to revert an operation that had added information to the internal structure that tracks tables. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1045	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to undo the addition of the new version to a partition.	The database server encountered an internal error while trying to revert an operation that had added information to the internal structure that tracks tables. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 1 Event ID: 1046	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to allocate the file descriptor for a partition number.	The database server encountered an internal error while trying to create a new file descriptor for a table or index. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 1 Event ID: 1047	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: An internal error was reported by the database server when it tried to free the file descriptor for a partition number.	The database server encountered an internal error while trying to release an internal data structure associated with a table or index. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** None. The database server will internally correct the issue.
Class ID: 1 Event ID: 1048	3	Class message: Table failure: *'dbname:"owner".tabname'* Specific message: Error updating table record.	The database server was unable to update a database record for a table that has in-place alters. It was unable to write the new version of the record. **Online log:** Assertion Warning with details of the error encountered and the *database:table* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2001	3 or 4	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: Fragid *fragment_id*, Rowid *rowid* not found for delete in partnum *partition_number*	The database server did not delete a record because it could not find it in the index. **Online log:** Assertion indicating that a DELETE operation failed and details of the table and index where the problem occurred. **Server state:** Online. **User action:** Run the **oncheck -cI** command on the specified table and index, or on the database, and correct any errors found. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 2 Event ID: 2002	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error was raised due to an inconsistency in the index which is preventing the database server to position on the first record in that index.	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2003	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error was raised due to an inconsistency in the index which is preventing the database server to read ahead pages in that index.	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2004	4	Class message: Index failure: *'dbname:"owner".tabname-idxname'* Specific message: Page Check Error in *object*	The database server detected inconsistencies with an index. **Online log:** Various messages depending on where the issue was detected. For example: `Possible inconsistencies in a DBSpace TBLSpace Run 'oncheck -cD' on all DBSpace TBLSpaces` **Server state:** Online. **User action:** Review the online.log file for more information and run the recommended **oncheck -cD** command on the database.

ID	Severity	Messages	Explanation
Class ID: 2 Event ID: 2005	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error occurred during batched index read because the database server had an invalid index key item.	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2006	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: *index_page* log record too large to fit into the logical log buffer. Recommended minimum value for LOGBUFF is *number*.	The server detected that a log record for an index page is too large for the configured logical log buffer size. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and update the onconfig value for LOGBUFF to the recommended value.
Class ID: 2 Event ID: 2007	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: Comparison based on locale '*locale_name*' failed	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2008	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: Comparison failed	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2009	4	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error occurred while the database server was trying to add a new item to the index.	The database server could not insert a record into an index. **Online log:** Assertion specifying the index and the recommended **oncheck** command to run. **Server state:** Online. **User action:** Examine the online log file and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2010	4	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error was raised due to an inconsistency in the index which is preventing the database server to position at the correct item in the index.	The database server could not retrieve the correct item in the index because of an inconsistency in the index. **Online log:** Assertion specifying the index and the recommended **oncheck** command to run. **Server state:** Online. **User action:** Examine the online log file and run the recommended **oncheck** command.
Class ID: 2 Event ID: 2011	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: Cannot drop index *index_name* for partnum *partition_number*, iserrno = *error_number*	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information and run the recommended **oncheck** command. Retry the original operation and if it fails again contact Software Support.
Class ID: 2 Event ID: 2012	3	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error occurred while the database server was trying to mark an index key descriptor as bad.	The database server detected an inconsistent index and marked it as unusable. **Online log:** Assertion Warning with details of the error encountered and the *database:table:index* involved. **Server state:** Online. **User Action:** Review the online.log file for more information, run the recommended **oncheck** command, fix any problems detected, then re-enable the index.

ID	Severity	Messages	Explanation
Class ID: 2 Event ID: 2013	4	Class message: Index failure: *'dbname:"owner".tabname:idxname'* Specific message: An internal error occurred while the database server was trying to delete an item from the index.	The database server could not delete a record from an index. **Online log:** Assertion specifying the index and the recommended **oncheck** command to run. **Server state:** Online. **User action:** Examine the online log file and run the recommended **oncheck** command.
Class ID: 3 Event ID: 3001	3	Class message: Blob failure: *'dbname:"owner".tabname'* Specific message: tb_sockid in blob descriptor is corrupted. Current® table is *'dbname:"owner".tabname'*	
Class ID: 3 Event ID: 3002	3	Class message: Blob failure: *'dbname:"owner".tabname'* Specific message: Incorrect BLOB stamps.	
Class ID: 3 Event ID: 3003	4	Class message: Blob failure: *'dbname:"owner".tabname'* Specific message: BLOB Page Check error at *dbspace_name*	The database server performed a check on pages that are moving between disk and memory and the check failed. **Online log:** Assertion describing the error. **Server state:** Online. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 3 Event ID: 3004	3	Class message: Blob failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while trying to read a blob from a table.	
Class ID: 3 Event ID: 3005	3	Class message: Blob failure: *'dbname:"owner".tabname'* Specific message: An internal error occurred while trying to copy a blob from a table.	
Class ID: 4 Event ID: 4001	4	Class message: Chunk is offline, mirror is active: *chunk_number* Specific message: I/O error, *error_number* Chunk '*chunk_number*' -- Offline	An error has occurred reading from or writing to a chunk. The database server has taken the chunk offline and switched to performing all I/O operations on the active mirrored chunk. **Online log:** Assertion describing the error that occurred. **Server state:** Online. **User action:** Examine the online log for information and fix the error. Run the **onspaces -s** command to recover the offline chunk. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 4 Event ID: 4002	3 or 4	Class message: Chunk is offline, mirror is active: *chunk_number* Specific message: An internal error occurred during physical I/O because the chunk was not opened.	The database server cannot access a chunk and switched to performing all I/O operations on the active mirrored chunk. **Online log:** Assertion describing the error and information about the chunk where the problem occurred. **Server state:** Online. **User action:** Examine the online log, fix any errors, and recover the mirror by using the **onspaces** utility. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 4 Event ID: 4003	3	Class message: Chunk is offline, mirror is active: *chunk_number* Specific message: I/O error, *error_number* Chunk '*chunk_number*' -- Offline (sanity)	

ID	Severity	Messages	Explanation
Class ID: 4 Event ID: 4004	3	Class message: Chunk is offline, mirror is active: *chunk_number* Specific message: Chunk failed sanity check	
Class ID: 4 Event ID: 4005	3	Class message: Chunk is offline, mirror is active: *chunk_number* Specific message: Mirror Chunk *chunk_number* added to space '*space_number*'. Perform manual recovery.	
Class ID: 5 Event ID: 5001	4	Class message: Dbspace is offline: *'dbspace_name'* Specific message: Chunk *chunk_number* is being taken OFFLINE.	The database server took a dbspace offline because of an error in accessing a chunk. **Online log:** Assertion failure if the dbspace was a critical dbspace, such as the rootdbs. Assertion warning if the dbspace is not critical. Both provide information about the chunk and dbspace being taken offline. **Server state:** Online if a non-critical media failure. Offline if a critical media failure. **User action:** Examine the online log file and fix the underlying problem that caused the dbspace to be taken offline. You might need to restore the dbspace.
Class ID: 5 Event ID: 5002	4	Class message: Dbspace is offline: *'dbspace_name'* Specific message: WARNING! Chunk *chunk_number* is being taken OFFLINE for testing.	The database server has taken a dbspace offline as a result of an **onmode** command. **Online log:** Assertion warning indicating that the dbspace has been taken offline. **Server state:** Online. **User action:** None.
Class ID: 6 Event ID: 6016	3	Class message: Internal subsystem failure: *'message'* Specific message: Pool not freed. pool name:*pool_name*, address:*address*	
Class ID: 6 Event ID: 6017	4	Class message: Internal subsystem failure: *'message'* Specific message: CDR Grouper FanOut thread is aborting	A problem occurred with Enterprise Replication. **Online log:** Assertion describing the problem. **Server state:** Online. **User action:** Follow the instructions in the online log.
Class ID: 6 Event ID: 6018	4	Class message: Internal subsystem failure: *'message'* Specific message: CDR Pager: Paging File full: Waiting for additional space in CDR_QDATA_SBSPACE	The storage space of an Enterprise Replication queue is full. **Online log:** Assertion describing the problem. **Server state:** Online. **User action:** Add a chunk to one or more of the sbspaces specified by the CDR_QDATA_SBSPACE configuration parameter.
Class ID: 6 Event ID: 6021	3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was reported by the database server during conversion when it found some indices in the old format.	
Class ID: 6 Event ID: 6022	3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was reported by the database server when it checks for any new in-place alter pending in the current server during reversion.	
Class ID: 6 Event ID: 6023	3	Class message: Internal subsystem failure: *'message'* Specific message: Cannot open index '*dbname*:*index_name*', iserrno = *error_number*	

ID	Severity	Messages	Explanation
Class ID: 6 Event ID: 6024	3	Class message: Internal subsystem failure: *'message'* Specific message: Cannot drop index '*dbname*:*index_name*', iserrno = *error_number*	
Class ID: 6 Event ID: 6025	3	Class message: Internal subsystem failure: *'message'* Specific message: Cannot open table '*dbname*:*table_name*', iserrno = *error_number*	
Class ID: 6 Event ID: 6026	3	Class message: Internal subsystem failure: *'message'* Specific message: Cannot drop table '*dbname*:*table_name*', iserrno = *error_number*	
Class ID: 6 Event ID: 6027	3	Class message: Internal subsystem failure: *'message'* Specific message: An error was reported by the database server when it tried to drop the sysmaster database during reversion	
Class ID: 6 Event ID: 6030	3	Class message: Internal subsystem failure: *'message'* Specific message: Invalid or missing name for Subsystem Staging BLOBspace	
Class ID: 6 Event ID: 6033	5	Class message: Internal subsystem failure: *'message'* Specific message: Cache read error	The database server shut down after encountering an error while reading an internal cache. **Online log:** Assertion Failure. **Server State:** Offline. **User action:** Start the database server and try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6034	3	Class message: Internal subsystem failure: *'message'* Specific message: Could not start remote server	
Class ID: 6 Event ID: 6035	3	Class message: Internal subsystem failure: *'message'* Specific message: An error was reported by the database server during the handling of audit trail files.	
Class ID: 6 Event ID: 6036	3	Class message: Internal subsystem failure: *'message'* Specific message: Archive on *dbspaces_list* ABORTED	
Class ID: 6 Event ID: 6037	3	Class message: Internal subsystem failure: *'message'* Specific message: Waiting on BLOBSpace to appear for Logical Recovery	
Class ID: 6 Event ID: 6038	3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error reported by the database server. Users may need to look at the specific message which accompanies with this id.	

ID	Severity	Messages	Explanation
Class ID: 6 Event ID: 6039	3	Class message: Internal subsystem failure: *'message'* Specific message: Wrong page for cleaning deleted items	
Class ID: 6 Event ID: 6040	3	Class message: Internal subsystem failure: *'message'* Specific message: Buffer in wrong state for cleaning deleted items	
Class ID: 6 Event ID: 6041	5 or 3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was detected by the Buffer Manager in the database server.	For severity 5, the database server buffer manager encountered an internal error and either shut down or corrected the problem. **Online log:** Assertion warning or an assertion failure with a description of the operation being performed at the time of the error. Typically, an assertion warning shows that the error was internally corrected. **Server State:** Offline if the error was unrecoverable. Online if the error was corrected. **User action:** If the error was unrecoverable, start the database server and try the operation again. If the operation fails again, note all circumstances and contact Software Support. No action is required if the error was internally corrected by the database server.
Class ID: 6 Event ID: 6042	5 or 2	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was reported by the database server when it detected an inconsistency with the internal buffer queues.	For severity 5, the database server detected an inconsistency during the processing of internal buffer queues and either shut down or corrected the problem. **Online log:** Assertion warning or an assertion failure with a description of the operation being performed at the time of the error. Typically, an assertion warning shows that the error was internally corrected. **Server State:** Offline if the error was unrecoverable. Online if the error was corrected. **User action:** If the error was unrecoverable, start the database server and try the operation again. If the operation fails again, note all circumstances and contact Software Support. No action is required if the error was internally corrected by the database server.
Class ID: 6 Event ID: 6043	3	Class message: Internal subsystem failure: *'message'* Specific message: Internal file error	
Class ID: 6 Event ID: 6044	3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was corrected automatically by the database server when it tried to save the log buffer into a system log buffer.	
Class ID: 6 Event ID: 6045	5	Class message: Internal subsystem failure: *'message'* Specific message: Logical logging error for '*object*' in *'space'*	The database server shut down because of an error while processing logical logs. **Online log:** Assertion failure with a description of the operation and logical log information. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6046	4	Class message: Internal subsystem failure: *'message'* Specific message: Page Check Error in *object*	The database server detected inconsistencies in the data. **Online log:** Various outputs depending upon where the issue was detected. For example: `Possible inconsistencies in a DBSpace TBLSpace Run 'oncheck -cD' on all DBSpace TBLSpaces` **Server state:** Online. **User action:** Examine the online log file and run the recommended **oncheck -cD** command on the database.

ID	Severity	Messages	Explanation
Class ID: 6 Event ID: 6047	3	Class message: Internal subsystem failure: *'message'* Specific message: Errors occurred while recreating indexes	
Class ID: 6 Event ID: 6049	5	Class message: Internal subsystem failure: *'message'* Specific message: Lock types *lock_type* and *lock_type* should never be merged	The database server shut down after attempting to merge incompatible locks. **Online log:** Assertion failure with the lock types that the database server was attempting to merge. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6050	5	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was reported by the database server when it detected some corruption in the lock free list chain.	The database server shut down after detecting corruption of an internal structure that manages an internal list of free locks. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6051	3	Class message: Internal subsystem failure: *'message'* Specific message: ERROR - NO 'waitfor' locks in Critical Section!!!	
Class ID: 6 Event ID: 6052	3	Class message: Internal subsystem failure: *'message'* Specific message: Internal Tblspace error	
Class ID: 6 Event ID: 6053	3	Class message: Internal subsystem failure: *'message'* Specific message: Session does not have exclusive access to partition *partition_name*. Request to drop the partition ignored.	
Class ID: 6 Event ID: 6054	3	Class message: Internal subsystem failure: *'message'* Specific message: Error building 'sysmaster' database.	
Class ID: 6 Event ID: 6055	3	Class message: Internal subsystem failure: *'message'* Specific message: Setread error on SMI Table, partnum *partition_number*	
Class ID: 6 Event ID: 6056	3	Class message: Internal subsystem failure: *'message'* Specific message: Comparison based on locale '*locale_name*' failed	
Class ID: 6 Event ID: 6057	2	Class message: Internal subsystem failure: *'message'* Specific message: DBSPACETEMP internal list not initialized, using default	The database server did not create the necessary structures for holding the DBSPACETEMP information. **Online log:** Message stating that the internal DBSPACETEMP list was not initialized. **Server state:** Online. **User action:** None.

ID	Severity	Messages	Explanation
Class ID: 6 Event ID: 6058	3	Class message: Internal subsystem failure: *'message'* Specific message: A data source accessed using a gateway (*gateway_name*) might be in an inconsistent state	
Class ID: 6 Event ID: 6059	3	Class message: Internal subsystem failure: *'message'* Specific message: Prepared participant site *site_name* not responding	
Class ID: 6 Event ID: 6060	5	Class message: Internal subsystem failure: *'message'* Specific message: Thread exited with *number* buffers held	The database server shut down after detecting that a thread is holding one or more buffers. **Online log:** Assertion failure with the number of buffers being held by the thread. **Server State:** Offline. **User action:** Bring the database server online. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6061	3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was automatically corrected by the database server when it detected that the undo log for the transaction was not applicable.	
Class ID: 6 Event ID: 6062	3	Class message: Internal subsystem failure: *'message'* Specific message: Internal Error - Freeing transaction entry that still holds locks!	While freeing resources associated with a transaction, the database server detected that the transaction is holding locks. In most circumstances the database server can release these locks. **Online log:** Assertion warning with the transaction and a statement that the database server internally corrected the problem. **Server State:** Online. **User action:** If the database server shut down, start the database server.
Class ID: 6 Event ID: 6063	3	Class message: Internal subsystem failure: *'message'* Specific message: User thread not on TX wait list	
Class ID: 6 Event ID: 6064	3	Class message: Internal subsystem failure: *'message'* Specific message: Due to a heuristic decision, the work done on behalf of the specified transaction branch might have been heuristically completed or committed or rolled back or partially committed and partially rolled back.	
Class ID: 6 Event ID: 6065	3	Class message: Internal subsystem failure: *'message'* Specific message: Errors occurred while recreating indexes	
Class ID: 6 Event ID: 6066	3	Class message: Internal subsystem failure: *'message'* Specific message: An internal error is reported by the database server when it has checked all sites to see if a heuristic rollback was the reason for the failure.	

ID	Severity	Messages	Explanation
Class ID: 6 Event ID: 6067	5	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (Recursive exception) has caused the database server processes to terminate unexpectedly.	The database server detected recursive calls to exception handling and immediately shut down to avoid an infinite loop. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6068	5	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (Internal exception) has caused the database server processes to terminate unexpectedly.	The database server shut down due to an unrecoverable internal error. **Online log:** Assertion failure with information about the exception that caused the problem. **Server State:** Offline. **User action:** Start the database server. Look at the exception information in the assertion failure file. If the exception relates to a user-defined routine, investigate and correct the user-defined routine. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6069	5	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (Master daemon died) has caused the database server processes to terminate unexpectedly.	The master daemon **oninit** process stopped and the database server shut down. This error can be caused by the termination of operating system processes. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. Be careful when terminating operating system processes.
Class ID: 6 Event ID: 6070	5	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (VP died) has caused the database server processes to terminate unexpectedly.	An **oninit** process stopped and the database server shut down. This error can be caused by the termination of operating system processes. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. Be careful when terminating operating system processes.
Class ID: 6 Event ID: 6071	5	Class message: Internal subsystem failure: *'message'* Specific message: ERROR: can not fork secondary Server thread (MACH11 Shutdown)	The secondary server shut down but was unable to create a thread to shut down normally. **Online log:** `DR: Shutting down the server. ERROR: can not fork secondary Server thread (MACH11 Shutdown) Can not run onmode -ky PANIC: Attempting to bring system down.` **Server State:** Offline. **User action:** None.
Class ID: 6 Event ID: 6072	3	Class message: Internal subsystem failure: *'message'* Specific message: Generic unique event id when the server failed to fork a new thread.	
Class ID: 6 Event ID: 6073	3	Class message: Internal subsystem failure: *'message'* Specific message: An error was reported by the database server when it could not initialize GLS for starting a session.	
Class ID: 6 Event ID: 6074	3	Class message: Internal subsystem failure: *'message'* Specific message: WARNING: mt_aio_wait: errno == EINVAL	
Class ID: 6 Event ID: 6075	5	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (KAIO) has caused the database server processes to terminate unexpectedly.	The database server shut down because of an error in the KAIO subsystem. **Online log:** Assertion failure with the specific operation that failed. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.

ID	Severity	Messages	Explanation
Class ID: 6 Event ID: 6100		Generic event for when the database server implicitly raises an assert warning.	A generic internal error occurred. **Online log:** Assertion warning with problem details. **Server State:** Online. **User action:** Look at the online log and take any recommended corrective action. The database server might correct the problem automatically. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6300		Generic event for when the database server implicitly raises an assert failure.	A generic internal error occurred. **Online log:** Assertion failure with problem details. **Server State:** Online. **User action:** Look at the online log and take any recommended corrective action. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6500		Generic event for when the database server terminates unexpectedly due to an internal error condition.	An internal error occurred and the database server shut down. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. Examine the assertion failure file for more information about what happened. If possible, fix any problems identified and try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7001	3	Class message: Database server initialization failure Specific message: TABLOCKS log record too large to fit into the logical log buffer. Recommended minimum value for LOGBUFF is *size*. I-STAR(C) begins prepare log record too large to fit into the logical log buffer. Recommended minimum value for LOGBUFF is *size*. Partition blob log record too large to fit into the logical log buffer. Recommended minimum value for LOGBUFF is *size*. Alter table special column desc log record too large to fit into the logical log buffer. Recommended minimum value for LOGBUFF is *size*.	
Class ID: 7 Event ID: 7002	4	Class message: Database server initialization failure Specific message: Unable to extend *number* reserved pages for checkpoint in ROOT chunk. Unable to extend *number* reserved pages for log in ROOT chunk.	The database server could not start because it could not allocate more space for internal structures in the initial root chunk. **Online log:** Assertion. **Server state:** Offline. **User action:** Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7003	4	Class message: Database server initialization failure Specific message: An internal error occurred during conversion. Users may need to take a look at the specific messages for further action.	The database server could not start during an upgrade because an internal error occurred during the conversion process. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Look at the online log and the specific message and take the necessary corrective action. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7004	4	Class message: Database server initialization failure Specific message: An internal error occurred while trying to convert the database tblspace.	The database server cannot start because of an internal error when trying to convert the database tblspace, which holds information about the databases in the instance. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Contact Software Support.
Class ID: 7 Event ID: 7005	4	Class message: Database server initialization failure Specific message: An internal error occurred while trying to convert blob free map pages.	The database server cannot start because of an internal error when trying to convert blobspace free-map pages. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Contact Software Support.

ID	Severity	Messages	Explanation
Class ID: 7 Event ID: 7006	4	Class message: Database server initialization failure Specific message: Cannot Open Logical Log.	The database server cannot start because it is still restoring physical or logical logs. This situation can occur if the **onmode -m** or **onmode -s** command is run before the restore is complete. **Online log:** Assertion describing the error. **Server state:** Recovering and starting. **User action:** Run the **onmode -m** or **onmode -s** command after the restore is complete.
Class ID: 7 Event ID: 7007	4	Class message: Database server initialization failure Specific message: Logical Log File not found.	The database server cannot start because a logical log file is missing. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Restore the database server from a backup.
Class ID: 7 Event ID: 7008	3	Class message: Database server initialization failure Specific message: WARNING! LTXHWM is set to 100%. This long transaction high water mark will never be reached. Transactions will not be aborted automatically by the server, regardless of their length.	
Class ID: 7 Event ID: 7009	4	Class message: Database server initialization failure Specific message: A Physical or Logical Restore is active.	The database server cannot start because it is still restoring physical or logical logs. This situation can occur if the **onmode -m** or **onmode -s** command is run before the restore is complete. **Online log:** Assertion describing the error. **Server state:** Recovering and starting. **User action:** Run the **onmode -m** or **onmode -s** command after the restore is complete.
Class ID: 7 Event ID: 7010	4	Class message: Database server initialization failure Specific message: *root_dbspace* has not been physically recovered.	The database server cannot start because the restore was interrupted before the rootdbs was physically restored. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Restore the rootdbs.
Class ID: 7 Event ID: 7011	4	Class message: Database server initialization failure Specific message: *dbspace* has not been physically recovered.	The database cannot start because a dbspace is not physically restored. This situation can occur if the database server is attempted to be started before a restore is complete. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Wait until the restore is complete before starting the database server.
Class ID: 7 Event ID: 7012	4	Class message: Database server initialization failure Specific message: *dbspace* not recovered from same archive backup as *dbspace*.	The database server cannot start because a dbspace was not restored successfully. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Recover the dbspace from a backup and roll forward the necessary logs to bring the dbspace to the correct point in time.
Class ID: 7 Event ID: 7013	4	Class message: Database server initialization failure Specific message: Log *log_number* not found.	The database server cannot start because a restore is not complete. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Wait until the restore is complete before starting the database server.
Class ID: 7 Event ID: 7014	4	Class message: Database server initialization failure Specific message: Logical restore cannot be skipped. Perform a logical restore.	The database server cannot start because a logical restore is not complete. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Perform a logical restore (for example, by using the **onbar -r -l** command) and start the database server in quiescent or online mode.

ID	Severity	Messages	Explanation
Class ID: 7 Event ID: 7015	4	Class message: Database server initialization failure Specific message: Cannot change to On-Line or Quiescent mode.	The database server cannot start because of an error during fast or full recovery. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Examine the online log for more information. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7016	4	Class message: Database server initialization failure Specific message: Cannot Open Primary Chunk '*chunk_number*'.	The database server cannot start because it could not access a primary chunk. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Examine the online log for more information. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7017	4	Class message: Database server initialization failure Specific message: The chunk '*chunk_number*' must have owner-ID "owner_id" and group-ID "*group_id*".	The database server cannot start because the owner and group of a chunk path are not correct. **Online log:** Assertion describing the problem. **Server state:** Offline. **User action:** Correct the permissions on the chunk path mentioned in the specific message. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7018	4	Class message: Database server initialization failure Specific message: The chunk '*chunk_number*' must have READ/WRITE permissions for owner and group (660).	The database server cannot start because the permissions on a chunk path are not correct. **Online log:** Assertion describing the problem. **Server state:** Offline. **User action:** Correct the permissions on the chunk path mentioned in the specific message. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7019	4	Class message: Database server initialization failure Specific message: Memory allocation error.	The database server cannot start because it failed to allocate enough memory. **Online log:** **Server state:** Offline. **User action:** Ensure that enough memory is available for the configuration you have specified for the database server. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7020	4	Class message: Database server initialization failure Specific message: The chunk '*chunk_number*' will not fit in the space specified.	The database server cannot start because there is insufficient space to create the specified chunk. **Online log:** Assertion describing the problem. **Server state:** Offline. **User action:** Specify a smaller size for the chunk or free additional space for the chunk.
Class ID: 7 Event ID: 7021	4	Class message: Database server initialization failure Specific message: *device_name*: write failed, file system is full.	The database server cannot start because the file system does not have free space. **Online log:** Assertion describing the problem. **Server state:** Offline. **User action:** Ensure the file system mentioned in the specific message has enough space. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7022	3	Class message: Database server initialization failure Specific message: An error occurred while the database server was creating the SMI database.	

ID	Severity	Messages	Explanation
Class ID: 7 Event ID: 7023	4	Class message: Database server initialization failure Specific message: Unable to create boot strap config file - '*file_name*'	The database server cannot start because it could not create a configuration file. **Online log:** Assertion describing the error. **Server state:** Offline. **User action:** Examine the online log for more information and fix the problem. The problem might be incorrect permissions on a directory. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 7 Event ID: 7024	3	Class message: Database server initialization failure Specific message: 'sysmaster' database will not be built/checked	
Class ID: 7 Event ID: 7025	3	Class message: Database server initialization failure Specific message: WARNING! Physical Log size *size* is too small. Physical Log overflows may occur during peak activity. Recommended minimum Physical Log size is *number* times maximum concurrent user threads.	
Class ID: 7 Event ID: 7026	3	Class message: Database server initialization failure Specific message: WARNING! Logical log layout may cause __ISN__ to get into a locked state. Recommended smallest logical log size is *number* times maximum concurrent user threads.	
Class ID: 7 Event ID: 7027	3	Class message: Database server initialization failure Specific message: WARNING! Buffer pool size may cause __ISN__ to get into a locked state. Recommended minimum buffer pool size is *number* times maximum concurrent user threads.	
Class ID: 7 Event ID: 7028	3	Class message: Database server initialization failure Specific message: Checkpoint log record may not fit into the logical log buffer. Recommended minimum value for LOGBUFF is *size*.	
Class ID: 7 Event ID: 7029	3	Class message: Database server initialization failure Specific message: Temp transaction not NULL.	
Class ID: 9 Event ID: 9001	4	Class message: Physical recovery failure Specific message: Physical log recovery error	The physical recovery of the database server failed. **Online log:** Assertion failure with a description of the problem. **Server state:** Online. **User action:** Retry the operation or restore from a backup.
Class ID: 10 Event ID: 10001	3 or 4	Class message: Logical recovery failure Specific message: Rollback error *error_number*	Logical recovery failed because the database server could not roll back a transaction. **Online log:** Assertion with details of the error and the log or log record where the problem occurred. **Server state:** Online or offline, depending on the error. **User action:** Examine the online log file for more information and run any recommended commands, such as an **oncheck** command. Retry the original operation. If the operation fails again, note all circumstances and contact Software Support.

ID	Severity	Messages	Explanation
Class ID: 10 Event ID: 10002	4	Class message: Logical recovery failure Specific message: Logical Recovery ABORTED.	The logical recovery of the database server failed. **Online log:** Assertion warning with information about the log record. Assertion failure with information about the log record if the failure is associated with a critical dbspace. **Server state:** Online if the dbspace is not critical. Offline if the dbspace is critical. **User action:** Examine the online log to determine the appropriate action, for example, you might need to restart the warm restore.
Class ID: 10 Event ID: 10003	4	Class message: Logical recovery failure Specific message: Log record (*log_subsystem*:*log_type*) in log *log_number*, offset *log_position* was not rolled back	Logical recovery encountered an internal error while rolling back a transaction. **Online log:** Message describing the log record. **Server state:** Online. **User action:** Examine the online log and determine the appropriate action, for example, resubmit the transaction.
Class ID: 10 Event ID: 10004	3	Class message: Logical recovery failure Specific message: Logical Logging error for '*log_subsystem*:*log_type*' in '*object*'	
Class ID: 10 Event ID: 10005	4	Class message: Logical recovery failure Specific message: An internal error occurred while trying to apply the log records during logical log recovery.	Logical recovery failed. **Online log:** Assertion warning with information about the log record. **Server state:** Depends on the failure. **User action:** Examine the online log and determine the appropriate action, for example, restart the warm restore.
Class ID: 10 Event ID: 10006	3 or 4	Class message: Logical recovery failure Specific message: An internal error occurred when the database server tried to find the file descriptor for the tblspace.	Logical recovery failed because the database server could not find an internal file descriptor for a partition. **Online log:** Assertion indicating for which table the error occurred and instructions to run the **oncheck** command. **Server state:** Online. **User action:** Run the **oncheck -cDI** command for the table mentioned in the online log or for the database.
Class ID: 11 Event ID: 11001	3	Class message: Cannot open chunk: *'pathname'* Specific message: Cannot Open Mirror Chunk '*chunk_number*', errno = *error_number*	
Class ID: 11 Event ID: 11002	3	Class message: Cannot open chunk: *'pathname'* Specific message: Cannot Open Primary Chunk '*chunk_number*', errno = *error_number*	
Class ID: 12 Event ID: 12001	3	Class message: Cannot open dbspace: *'dbspace_name'* Specific message: ERROR: DBspace *dbspace_name* not found among table *table_name* fragments.	
Class ID: 13 Event ID: 13001	2	Class message: Performance improvement possible Specific message: The number of configured CPU poll threads exceeds number of CPU VPs specified in 'VPCLASS cpu'. NETTYPE '*protocol*' poll threads started on NET VPs.	The database server detected a configuration mismatch between the number of CPU virtual processors and the number of requested CPU poll threads during server initialization. **Online log:** Performance warning about the configuration mismatch. The database server uses NET virtual processors instead. **Server state:** Online. **User action:** Check the configuration of the server.

ID	Severity	Messages	Explanation
Class ID: 13 Event ID: 13002	2	Class message: Performance improvement possible Specific message: Transaction table overflow due to parallel recovery.	An internal structure is not large enough to process the logical log. The database server will postpone the log processing until more space exists within the structure. **Online log:** Warning message indicating that the transaction processing was delayed. **Server state:** Online. **User action:** None.
Class ID: 14 Event ID: 14001	3	Class message: Database failure. *'dbname'* Specific message: ''*dbname*" - Error *error_number* during logging mode change.	
Class ID: 15 Event ID: 15001	3	Class message: High-Availability Data-Replication failure Specific message: DR: Turned off on secondary server	
Class ID: 15 Event ID: 15002	3	Class message: High-Availability Data-Replication failure Specific message: DR: Turned off on primary server	
Class ID: 15 Event ID: 15003	3	Class message: High-Availability Data-Replication failure Specific message: DR: Cannot connect to secondary server	
Class ID: 15 Event ID: 15004	3	Class message: High-Availability Data-Replication failure Specific message: DR: Received connection request from remote server when DR is not Off [Local type: *type*, Current state: *state*] [Remote type: *type*]	
Class ID: 15 Event ID: 15005	3	Class message: High-Availability Data-Replication failure Specific message: DR: Received connection request before physical recovery completed.	
Class ID: 15 Event ID: 15006	3	Class message: High-Availability Data-Replication failure Specific message: DR: Local and Remote server type and/or last change (LC) incompatible [Local type: *type*, LC: *type*] [Remote type: *type*, LC: *type*]	
Class ID: 16 Event ID: 16001	2	Class message: Backup completed: *'dbspace_list'* Specific message: Archive on *dbspace_list* completed without being recorded.	An archive completed, but the server detected corrupted pages during the archive. **Online log:** Message indicating that the backup is complete but that corrupted pages have been detected. **Server state:** Online. **User action:** Do not use this backup. Use an earlier backup to immediately restore the bad chunks.

ID	Severity	Messages	Explanation
Class ID: 16 Event ID: 16002	2	Class message: Backup completed: *'dbspace_list'* Specific message: Archive on *dbspace_list* Completed with *number* corrupted pages detected.	An archive completed, but the server detected corrupted pages during the archive. **Online log:** Message indicating that the backup is complete but that corrupted pages have been detected. **Server state:** Online. **User action:** Do not use this backup. Use an earlier backup with 0 bad pages to immediately restore the bad chunks.
Class ID: 16 Event ID: 16003	2	Class message: Backup completed: *'dbspace_list'* Specific message: Archive on *dbspace_list* Completed	An archive completed for the dbspaces listed. **Online log:** Message indicating that the backup is complete for the dbspaces listed. **Server state:** Online. **User action:** None.
Class ID: 17 Event ID: 17001	4	Class message: Backup aborted: *'dbspace_list'* Specific message: Archive detects that page *chunk_number*:*page_offset* is corrupt.	The database server detected corruption and stopped the backup. **Online log:** Assertion describing the problem. **Server state:** Online. **User action:** Examine the online log for information about the corruption. Try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 17 Event ID: 17002	3	Class message: Backup aborted: *'dbspace_list'* Specific message: Page %d:%d of partition *partition_number* not archived.	
Class ID: 18 Event ID: 18001	2	Class message: Log backup completed: *log_number* Specific message: Logical Log *log_number* - Backup Completed	The logical log was backed up. **Online log:** Message identifying the log number of the backed up logical log. **Server state:** Online. **User action:** None.
Class ID: 19 Event ID: 19001	3	Class message: Log backup aborted: *log_number* Specific message: Logical Log *log_number* - Backup Aborted *message*	
Class ID: 20 Event ID: 20001	3	Class message: Logical logs are full—backup is needed Specific message: Logical Log Files are Full -- Backup is Needed	
Class ID: 20 Event ID: 20002	3	Class message: Logical logs are full—backup is needed Specific message: Waiting for Next Logical Log File to be Freed	
Class ID: 20 Event ID: 20003	3	Class message: Logical logs are full—backup is needed Specific message: Logical Log Files are almost Full -- Backup is Needed. In Data replication scenario, this could block failure-recovery of the paired server.	
Class ID: 21 Event ID: 21001	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Archive arcbu_next_tbuf() – Buffer Overflow	

ID	Severity	Messages	Explanation
Class ID: 21 Event ID: 21002	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Archive tcp_logbu_hdr() – Buffer Overflow	
Class ID: 21 Event ID: 21003	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Archive tcp_logbu_trl() – Buffer Overflow	
Class ID: 21 Event ID: 21004	2 or 5	Class message: Database server resource overflow: *'resource_name'* Specific message: Physical log file overflow	For severity 5, the physical log file is full and needs to overflow. If this happens during recovery, the database server attempts to extend the physical log. **Online log:** Assertion failure if the database server is either not in recovery or is unable to extend the physical log. Assertion warning if the database server is in recovery and extends the physical log. **Server State:** Offline. **User action:** None.
Class ID: 21 Event ID: 21005	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Lock table overflow - user id %d, session id %d	
Class ID: 21 Event ID: 21006	5	Class message: Database server resource overflow: *'resource_name'* Specific message: Logical log buffer overflow detected	The database server shut down because the logical log buffer is full. **Online log:** Assertion failure with the log record size and the buffer size. **Server State:** Offline. **User action:** Increase the value of the LOGBUFF configuration parameter in the onconfig file. Start the database server.
Class ID: 21 Event ID: 21007	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Llog logbu_logfile() – Buffer Overflow	
Class ID: 21 Event ID: 21008	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Llog logbu_bpage() – Buffer Overflow	
Class ID: 21 Event ID: 21009	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Unable to allocate a user thread for user id *user_ID*	
Class ID: 21 Event ID: 21010	3	Class message: Database server resource overflow: *'resource_name'* Specific message: Unable to allocate a transaction for user id *user_ID*, session id *session_ID*	
Class ID: 21 Event ID: 21014	3	Class message: OnLine resource overflow: Locks; Specific message: SID='*session id*' User ID='*uid*' NAME='*user name*' PID='*process id*'	The lock limit within this session is reached. The current transaction is terminated. **Online log:** Session *session id* has exceeded the session limit of *number* locks. **Server state:** Online **User action:** Reduce your transaction size.

ID	Severity	Messages	Explanation
Class ID: 21 Event ID: 21016	3	Class message: OnLine resource overflow: Memory; Specific message: Session info: '*session id username*@*hostname*' pid '*pid*'	The memory limit within this session is reached. The current session is terminated. **Online log:** Session *session id* has exceeded the session limit of *size* of memory. **Server state:** Online **User action:** Reduce session memory consumption.
Class ID: 21 Event ID: 21017	3	Class message: OnLine resource overflow: Tempspace; Specific message: Session info: '*session id username*@*hostname*' pid '*pid*'	The temporary space limit within this session is reached. The current session is terminated. **Online log:** Session *session id* has exceeded the session limit of *size* of temporary space. **Server state:** Online **User action:** Reduce your data size.
Class ID: 21 Event ID: 21018	3	Class message: OnLine resource overflow: Log Space; Specific message: Session info: '*session id username*@*hostname*' pid '*pid*'	The log space limit within this session is reached. The current transaction is terminated. **Online log:** Session *session id* has exceeded the session limit of *size* of log space. **Server state:** Online **User action:** Reduce your transaction size.
Class ID: 21 Event ID: 21019	3	Class message: OnLine resource overflow: Transaction Time; Specific message: Session info: '*session id username*@*hostname*' pid '*pid*'	The transaction time limit within this session is reached. The current session is terminated. **Online log:** Session *session id* has exceeded the session limit of *seconds* of transaction time. **Server state:** Online **User action:** Reduce idle time for open transactions.
Class ID: 22 Event ID: 22001	3	Class message: Long transaction detected Specific message: Blocking on XA transaction, tx *transaction_number*, till it is cleaned up.	
Class ID: 22 Event ID: 22002	3	Class message: Long transaction detected Specific message: Continuing Long Transaction (for COMMIT): tx:	
Class ID: 22 Event ID: 22003	3	Class message: Long transaction detected Specific message: Aborting Long Transaction: tx:	
Class ID: 23 Event ID: 23001	2	Class message: Logical log *'number'* complete Specific message: Logical Log *log_number* Complete, timestamp: *timestamp*.	The logical log is full, and no more transactions can be written to it. **Online log:** Message indicating that the logical log is full. **Server state:** Online. **User action:** None.
Class ID: 24 Event ID: 24001	3	Class message: Unable to allocate memory Specific message: Generic unique event id when the server failed to allocate memory for starting a new thread.	
Class ID: 24 Event ID: 24002	3	Class message: Unable to allocate memory Specific message: Warning: unable to allocate requested big buffer of size *size*	

ID	Severity	Messages	Explanation
Class ID: 24 Event ID: 24003	3	Class message: Unable to allocate memory Specific message: The database server tried to allocate a shared memory virtual segment before it was actually needed, in accordance with the setting of the SHMVIRT_ALLOCSEG configuration parameter - but the segment could not be added. Next failure message will be printed in 30 minutes.	
Class ID: 24 Event ID: 24004	3	Class message: Unable to allocate memory Specific message: out of message shared memory	
Class ID: 24 Event ID: 24005	3	Class message: Unable to allocate memory Specific message: out of message shared memory	
Class ID: 24 Event ID: 24006	3	Class message: Unable to allocate memory Specific message: out of virtual shared memory	
Class ID: 24 Event ID: 24007	3	Class message: Unable to allocate memory Specific message: No memory available for page cleaners	
Class ID: 24 Event ID: 24008	3	Class message: Unable to allocate memory Specific message: kysearch(): Memory allocation error	
Class ID: 24 Event ID: 24009	3	Class message: Unable to allocate memory Specific message: Lock table overflow - user id *user_ID*, session id *session_ID*	
Class ID: 24 Event ID: 24010	3	Class message: Unable to allocate memory Specific message: Unable to allocate a user thread for user id *user_ID*	
Class ID: 24 Event ID: 24011	3	Class message: Unable to allocate memory Specific message: Unable to allocate a transaction for user id *user_ID*, session id *session_ID*	
Class ID: 26 Event ID: 26001	3	Class message: Dynamically added log file *logid* Specific message: Dynamically added log file *logid* to DBspace *dbspace_name*	

ID	Severity	Messages	Explanation
Class ID: 27 Event ID: 27001	4	Class message: Log file required Specific message: ALERT: The oldest logical log (*log_number*) contains records from an open transaction (*transaction_number*). Logical logging will remain blocked until a log file is added. Add the log file with the onparams -a command, using the -i (insert) option, as in: onparams -a -d *dbspace* -s *size* -i Then complete the transaction as soon as possible.	The database server needs an additional log file to continue processing. **Online log:** `ALERT: The oldest logical log (log_number) contains records from an open transaction (transaction_number). Logical logging will remain blocked until a log file is added. Add the log file with the onparams -a command, using the -i (insert) option, as in: onparams -a -d dbspace -s size -i Then complete the transaction as soon as possible.` **Server state:** Online. **User action:** Add a new logical log.
Class ID: 28 Event ID: 28001	4	Class message: No space for log file Specific message: ALERT: Because the oldest logical log (*log_number*) contains records from an open transaction (*transaction_number*), the server is attempting to dynamically add a log file. But there is no space available. Please add a DBspace or chunk. Then complete the transaction as soon as possible.	The database server cannot dynamically add an additional logical log file because not enough space is available. **Online log:** Assertion warning indicating that there is not enough space available for an additional logical log file. **Server state:** Online. **User action:** Add a new logical log file or additional space.
Class ID: 28 Event ID: 28002	4	Class message: No space for log file Specific message: Warning - Enterprise Replication is attempting to dynamically add a log file. But there is no space available. The replay position may overrun.	The database server cannot dynamically add an additional logical log file because not enough space is available. **Online log:** Assertion warning indicating that there is not enough space available for an additional logical log file. **Server state:** Online. **User action:** Add a new logical log file or additional space.
Class ID: 29 Event ID: 29001	2	Class message: Internal subsystem: *subsystem* Specific message: Skipped existing audit trail files *file_name* to *file_name*.	The auditing subsystem needs to change to a new output file. **Online log:** Message indicating that the audit file changed, skipping over existing files. **Server state:** Online. **User action:** None.
Class ID: 30 - 39	2, 3, or 4	Enterprise Replication events. See Enterprise Replication Event Alarms.	
Class ID: 40 Event ID: 40001	3	Class message: RSS alarm Specific message: RSS *server_name* added	
Class ID: 40 Event ID: 40002	3	Class message: RSS alarm Specific message: Password for RSS Source *server_name* changed	
Class ID: 40 Event ID: 40003	3	Class message: RSS alarm Specific message: RSS *server_name* deleted	
Class ID: 40 Event ID: 40004	3	Class message: RSS alarm Specific message: RSS *server_name* log replay position is falling too far behind RSS Source	
Class ID: 40 Event ID: 40005	3	Class message: RSS alarm Specific message: RSS *server_name* is not acknowledging log transmission	

ID	Severity	Messages	Explanation
Class ID: 40 Event ID: 40006	3	Class message: RSS alarm Specific message: Error receiving a buffer from RSS *server_name* - shutting down	
Class ID: 40 Event ID: 40007	3	Class message: RSS alarm Specific message: Delay or Stop Apply: I/O write error: *error_number error_description*.	
Class ID: 40 Event ID: 40008	3	Class message: RSS alarm Specific message: Delay or Stop Apply: Thread exiting due to error.	
Class ID: 41 Event ID: 41001	3	Class message: SDS alarm Specific message: ERROR: Removing SDS Node *server_name* has timed out - removing	
Class ID: 42 Event ID: 42001	1	Class message: Event occurred	The database server encountered an error while validating the tablespace page. **Online log:** Assertion warning with details of the table. **Server state:** Online. **User action:** Examine the online log to determine which *database.owner.tablename* the issue occurred on. Run the **oncheck -pt** command on the table. Correct any errors identified by the **oncheck** utility and retry the operation. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 43 Event ID: 43001	3	Class message: Connection Manager alarm Specific message: CM:Session for Connection manager *name* terminated abnormally	
Class ID: 43 Event ID: 43002	3	Class message: Connection Manager alarm Specific message: The FOC setting *FOC_String* for Connection Manager *CM_Name* does not match the FOC setting for the other Connection Managers that are configured to arbitrate failover for the cluster. If this Connection Manager becomes the active arbitrator, its FOC will not match the previous FOC policy.	
Class ID: 44 Event ID: 44001	3	Class message: DBSpace is full: *dbspace_name* Specific message: WARNING: *dbspace_type dbspace_name* is full	
Class ID: 45 Event ID: 45001	3	Class message: partition *'partition_name'*: no more extents Specific message: Partition *'partition_name'*: No more extents	
Class ID: 46 Event ID: 46001	3	Class message: partition *'partition_name'*: no more pages Specific message: Partition *'partition_name*: No more pages	

ID	Severity	Messages	Explanation
Class ID: 47 - 71	3 or 4	Enterprise Replication events. See Enterprise Replication Event Alarms.	
Class ID: 72 Event ID: 72001	2	Class message: Audit trail is switched to a new file. Specific message: Audit trail switched to *file_name*	The auditing subsystem is switching to a new output file. **Online log:** Message providing the file name of the new output file. **Server state:** Online. **User action:** None.
Class ID: 73-77	3 or 4	Enterprise Replication events. See Enterprise Replication Event Alarms.	
Class ID: 78 Event ID: 78001	3	Class message: The storage pool is empty. Specific message: Warning: The storage pool is out of space.	
Class ID: 79 Event ID: 79001	3	Class message: Dynamically added chunk *chunk_name* to space Specific message: Dynamically added chunk *chunk_name* to space '*space_name*' Path: *path*, offset *offset_number* kilobytes Size: *size* kilobytes	
Class ID: 80 80001	2	Class message: A new fragment for table *table_name* has been added in DBspace *dbspace_name*.	A new fragment was automatically added to a table because the table grew larger than the size of its existing fragments. **Online log:** Message providing the table name and dbspace name. **Server state:** Online. **User action:** None.
Class ID: 81 Event ID: 81001	4	Class message: Logical log file or dbspace corruption detected during backup. Loguniq or Dbspace id: *ID*. Specific message: Log Backup detected a corrupted logical log file. Expected loguniq:pagenum *log_number*:*page_number* Actual loguniq:pagenum *log_number*:*page_number* Log backup continuing but the log backup cannot be used to restore a server. You should run **oncheck** and take a level 0 archive.	A backup failed because the database server detected corruption in the logical log file or dbspace. **Online log:** Assertion warning. **Server state:** Online. **User action:** Perform a new level-0 backup.
Class ID: 82 Event ID: 82001	3	Class message: session *ID* (*thread*) network write operation has been blocked for at least 30 minutes, which might indicate an operating system problem Specific message: session *ID* (*thread*) network write operation has been blocked for at least 30 minutes, which might indicate an operating system problem	
Class ID: 83 Event ID: 83001	3	Class message: SDS: Failover aborted - detected primary server is still active. Specific message: SDS: Failover aborted - detected primary server is still active.	

ID	Severity	Messages	Explanation
Event ID: 84001	3	Class message: Generic network failure alarm Specific message: Unable to bind to the port (port number or service name) on the host (IP address or host name) for the server (*dbservername*).	The host name or IP address, the service name, or the port number might be incorrect. The port might already be in use. **Server state:** Online, during server startup. **Online log:** Assertion warning. **User action:** Check the host name or IP address, the service name, and the port number entries in the sqlhosts file. Make that sure that the port is not already in use. Make the necessary changes and restart the server.
Event ID: 86001	3	Class message: Space has reached its maximum configured size. Specific msg: Warning: Space *space_name* has reached its maximum configured size (*size* MB).	The extendable storage space is at the configured maximum size and cannot expand further. **Server state:** Online **Online log:** **User action:** No action needed. If you want to increase the maximum size of the storage space, run the **admin()** or **task()** SQL administration API function with the modify space sp_sizes argument and specify a new maximum size.
Event ID: 87001	3	Message: **ifxguard** utility connected	The **ifxguard** agent connected to the database server. **Server state:** Online **Online log:** Message that identifies the agent name. **User action:** No action needed.
Event ID: 87002	3	Message: **ifxguard** utility disconnected	The **ifxguard** agent closed the connected to the database server. **Server state:** Online **Online log:** Message that identifies the agent name. **User action:** No action needed. The user session that the agent audited ended.
Event ID: 87003	3	Message: **ifxguard** utility has not responded	The **ifxguard** agent did not connect to the database server during the timeout period. **Server state:** Depends on the action set by the IFXGUARD configuration parameter. **Online log:** Message that identifies the agent name. **User action:** No action needed unless the database server shut down. If the database server shut down, examine the ifxguard log file to discover why the **ifxguard** agent failed to connect in time.

- Severity 5 event alarms
  Severity 5 event alarms indicate that the database server has failed.

Severity 5 event alarms

Severity 5 event alarms indicate that the database server has failed.

Table 1. Severity 5 event alarms

ID	Messages	Explanation
Class ID: 6 Event ID: 6033	Class message: Internal subsystem failure: *'message'* Specific message: Cache read error	The database server shut down after encountering an error while reading an internal cache. **Online log:** Assertion Failure. **Server State:** Offline. **User action:** Start the database server and try the operation again. If the operation fails again, note all circumstances and contact Software Support.

ID	Messages	Explanation
Class ID: 6 Event ID: 6041	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was detected by the Buffer Manager in the database server.	The database server buffer manager encountered an internal error and either shut down or corrected the problem. **Online log:** Assertion warning or an assertion failure with a description of the operation being performed at the time of the error. Typically, an assertion warning shows that the error was internally corrected. **Server State:** Offline if the error was unrecoverable. Online if the error was corrected. **User action:** If the error was unrecoverable, start the database server and try the operation again. If the operation fails again, note all circumstances and contact Software Support. No action is required if the error was internally corrected by the database server.
Class ID: 6 Event ID: 6042	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was reported by the database server when it detected an inconsistency with the internal buffer queues.	The database server detected an inconsistency during the processing of internal buffer queues and either shut down or corrected the problem. **Online log:** Assertion warning or an assertion failure with a description of the operation being performed at the time of the error. Typically, an assertion warning shows that the error was internally corrected. **Server State:** Offline if the error was unrecoverable. Online if the error was corrected. **User action:** If the error was unrecoverable, start the database server and try the operation again. If the operation fails again, note all circumstances and contact Software Support. No action is required if the error was internally corrected by the database server.
Class ID: 6 Event ID: 6045	Class message: Internal subsystem failure: *'message'* Specific message: Logical logging error for '*object*' in *'space'*	The database server shut down because of an error while processing logical logs. **Online log:** Assertion failure with a description of the operation and logical log information. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6049	Class message: Internal subsystem failure: *'message'* Specific message: Lock types *lock_type* and *lock_type* should never be merged	The database server shut down after attempting to merge incompatible locks. **Online log:** Assertion failure with the lock types that the database server was attempting to merge. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6050	Class message: Internal subsystem failure: *'message'* Specific message: An internal error was reported by the database server when it detected some corruption in the lock free list chain.	The database server shut down after detecting corruption of an internal structure that manages an internal list of free locks. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6060	Class message: Internal subsystem failure: *'message'* Specific message: Thread exited with *number* buffers held	The database server shut down after detecting that a thread is holding one or more buffers. **Online log:** Assertion failure with the number of buffers being held by the thread. **Server State:** Offline. **User action:** Bring the database server online. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6067	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (Recursive exception) has caused the database server processes to terminate unexpectedly.	The database server detected recursive calls to exception handling and immediately shut down to avoid an infinite loop. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.

ID	Messages	Explanation
Class ID: 6 Event ID: 6068	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (Internal exception) has caused the database server processes to terminate unexpectedly.	The database server shut down due to an unrecoverable internal error. **Online log:** Assertion failure with information about the exception that caused the problem. **Server State:** Offline. **User action:** Start the database server. Look at the exception information in the assertion failure file. If the exception relates to a user-defined routine, investigate and correct the user-defined routine. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6069	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (Master daemon died) has caused the database server processes to terminate unexpectedly.	The master daemon **oninit** process stopped and the database server shut down. This error can be caused by the termination of operating system processes. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. Be careful when terminating operating system processes.
Class ID: 6 Event ID: 6070	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (VP died) has caused the database server processes to terminate unexpectedly.	An **oninit** process stopped and the database server shut down. This error can be caused by the termination of operating system processes. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. Be careful when terminating operating system processes.
Class ID: 6 Event ID: 6071	Class message: Internal subsystem failure: *'message'* Specific message: ERROR: cannot fork secondary Server thread (MACH11 Shutdown)	The secondary server shut down but was unable to create a thread to shut down normally. **Online log:** `DR: Shutting down the server. ERROR: can not fork secondary Server thread (MACH11 Shutdown) Can not run onmode -ky PANIC: Attempting to bring system down.` **Server State:** Offline. **User action:** None.
Class ID: 6 Event ID: 6075	Class message: Internal subsystem failure: *'message'* Specific message: A fatal internal error (KAIO) has caused the database server processes to terminate unexpectedly.	The database server shut down because of an error in the KAIO subsystem. **Online log:** Assertion failure with the specific operation that failed. **Server State:** Offline. **User action:** Start the database server. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 6 Event ID: 6500	Generic event for when the database server terminates unexpectedly due to an internal error condition.	An internal error occurred and the database server shut down. **Online log:** Assertion failure. **Server State:** Offline. **User action:** Start the database server. Examine the assertion failure file for more information about what happened. If possible, fix any problems identified and try the operation again. If the operation fails again, note all circumstances and contact Software Support.
Class ID: 21 Event ID: 21004	Class message: Database server resource overflow: *'resource_name'* Specific message: Physical log file overflow	The physical log file is full and needs to overflow. If this happens during recovery, the database server attempts to extend the physical log. **Online log:** Assertion failure if the database server is either not in recovery or is unable to extend the physical log. Assertion warning if the database server is in recovery and extends the physical log. **Server State:** Offline. **User action:** None.
Class ID: 21 Event ID: 21006	Class message: Database server resource overflow: *'resource_name'* Specific message: Logical log buffer overflow detected	The database server shut down because the logical log buffer is full. **Online log:** Assertion failure with the log record size and the buffer size. **Server State:** Offline. **User action:** Increase the value of the LOGBUFF configuration parameter in the onconfig file. Start the database server.

Connection Manager event alarm IDs

The class ID for event alarms indicates the type of event. The event ID indicates the specific event.

The following table lists event alarm IDs and messages for the Connection Manager.

You can set your alarm program script to capture the Connection Manager class ID and message and initiate corrective actions or notifications.

You can use the values set in the INFORMIXCMNAME and INFORMIXCMCONUNITNAME environment variables when writing an alarm handler for the Connection Manager. If the Connection Manager raises an event alarm, the Connection Manager instance name is stored in the INFORMIXCMNAME environment variable, and the Connection Manager connection unit name is stored in the INFORMIXCMCONUNITNAME environment variable.

Event alarm messages are written to the Connection Manager log file.

Table 1. Connection Manager event alarms

ID	Severity	Messages	Explanation
Class ID: 1 Event ID: 1001	3	Class message: Connection Manager generic alarm Specific message: Connection Manager stopped	The Connection Manager stopped running. **Online log message:** Connection Manager shut down successfully. **User action:** Restart the Connection Manager, if necessary.
Class ID: 1 Event ID: 1002	3	Class message: Connection Manager generic alarm Specific message: Connection Manager fatal error	The Connection Manager failed to initialize. **Online log message:** Failed to switch to daemon mode, Connection Manager stopped. Error: Initialize failed, Connection Manager stopped. Error: SLA listener failed, Connection Manager can not start. **User action:** Check the message file for failure details. Correct any errors and then restart the Connection Manager.
Class ID: 1 Event ID: 1003	3	Class message: Connection Manager generic alarm Specific message: Connection Manager received signal	The Connection Manager stopped or crashed. **Online log message:** Connection Manager process received signal, shutting down **User action:** If the Connection Manager was killed by signal 9, no action is required. Otherwise, report the problem to the System Administrator.
Class ID: 2 Event ID: 2001	3	Class message: Failover Arbitrator alarm Specific message: Failover in progress	The Connection Manager Failover Arbitrator initiated a failover event. **Online log message:** Failover Arbitrator automated failover in progress.
Class ID: 2 Event ID: 2002	3	Class message: Failover Arbitrator alarm Specific message: Failover completed	The Connection Manager Failover Arbitrator has completed failover. **Online log message:** Failover Arbitrator automated failover completed.
Class ID: 2 Event ID: 2003	3	Class message: Failover Arbitrator alarm Specific message: Failover disabled	Automated failover for the Connection Manager is disabled. **Online log message:** Failover Arbitrator automated failover is disabled. **User action:** N/A
Class ID: 2 Event ID: 2004	3	Class message: Failover Arbitrator alarm Specific message: Failover Arbitrator aborting automated failover	Failover processing has failed. **Online log message:** Failover Arbitrator aborting automated failover. **User action:** Check the message log file, and then manually start the primary server or manually perform failover.
Class ID: 2 Event ID: 2005	3	Class message: Failover Arbitrator alarm Specific message: Failover processing is in manual mode	Failover processing is not in automatic mode. **Online log message:** Failover processing is in manual mode
Class ID: 3 Event ID: 3001	3	Class message: Connection to the primary Specific message: Cannot connect to primary server	The Connection Manager cannot connect to the primary server. **Online log message:** Unable to connect to Informix® server. **User action:** Correct the setup problem. Connection Manager can then connect to the server automatically.
Class ID: 3 Event ID: 3002	3	Class message: Connection to the primary Specific message: Lost connection to primary server	The Connection Manager is disconnected from the primary server. **Online log message:** Detected lost connection to Informix server. **User action:** Correct the setup or network problem. The Connection Manager can then connect to the primary server automatically.

ID	Severity	Messages	Explanation
Class ID: 3 Event ID: 3003	4	Class message: Connection to the primary Specific message: CM detected multiple primary servers in the cluster	Connection Manager is connected to multiple primary servers. **Online log message:** Detected multiple primary servers in the cluster. **User action:** Correct the setup or network problem so that Connection Manager is connected to only one primary server.
Class ID: 4 Event ID: 4001	3	Class message: Connection to ER node Specific message: Cannot connect to ER node	The Connection Manager cannot connect to an Enterprise Replication server. **User action:** Correct the setup problem. The Connection Manager can then connect to the Enterprise Replication server automatically.
Class ID: 4 Event ID: 4002	3	Class message: Connection to ER node Specific message: Lost connection to ER node	The Connection Manager has disconnected from an Enterprise Replication server. **Online log message:** Detected lost connection to Informix server **User action:** Correct the setup or network problem. The Connection Manager can then connect to the Enterprise Replication server automatically.
Class ID: 5 Event ID: 5001	3	Class message: Connection to generic server Specific message: Cannot connect to server	The Connection Manager cannot connect to a server in a high-availability cluster. **Online log message:** Unable to connect to Informix server. **User action:** Correct the setup problem. Connection Manager can then connect to the high-availability server automatically.
Class ID: 5 Event ID: 5002	3	Class message: Connection to generic server Specific message: Lost connection to server	The Connection Manager is disconnected from a secondary server. **Online log message:** Detected lost connection to Informix server. **User action:** Correct the setup or network problem. Connection Manager can then connect to the secondary server automatically.

Related information:
INFORMIXCMNAME environment variable
INFORMIXCMCONUNITNAME environment variable

Messages in the database server log

Unnumbered messages are printed in the database server message log (online.log). The error messages include corrective actions.

For a description of an error message, use the **finderr** utility or go to Error messages.

Some of the messages might require you to contact Software Support.

- How the Messages Are Ordered in This Chapter
- Messages: A-B
- Messages: C
- Messages: D-E-F
- Messages: G-H-I
- Messages: J-K-L-M
- Messages: N-O-P
- Messages: Q-R-S
- Messages: T-U-V
- Messages: W-X-Y-Z
- Messages: Symbols
- Conversion and reversion error messages
  If conversion or reversion is not successful, error messages are stored in the online.log file to help you identify what failed and what actions to take to fix the problem.
- Conversion and Reversion Messages for Enterprise Replication
  During conversion and reversion, specific messages are logged for Enterprise Replication by the concdr, revcdr, and revtestcdr scripts.
- Dynamic Log Messages
- Sbspace Metadata Messages
- Truncate Table Messages

Related reference:
MSGPATH configuration parameter
Related information:
ON-Bar messages and return codes

How the Messages Are Ordered in This Chapter

Database server message-log messages are arranged in this chapter in alphabetical order, sorted with the following additional rules:

- The time stamp that precedes each message is ignored.
- Letter case is ignored in alphabetization.
- Spaces are ignored.
- Quotation marks are ignored.
- Leading ellipses are ignored.
- The word *the* is ignored if it is the first word in the message.
- Messages that begin with numbers or punctuation symbols appear toward the end of the list in a special section labeled Messages: Symbols.
- Certain related messages are grouped together, as follows:
  - Conversion and reversion error messages
  - Conversion and Reversion Messages for Enterprise Replication
  - Dynamic Log Messages
  - Sbspace Metadata Messages
  - Truncate Table Messages

A cause and suggested corrective action for a message or group of messages follow the message text.

- How to view these messages
- Message Categories

How to view these messages

Use one of the following methods to view these messages:

- Online message log
  To see the messages displayed as they occur, use the **tail -f** *online.log* command.
- **onstat -m** command
  For more information, see onstat -l command: Print physical and logical log information.

To see the error number associated with these unnumbered messages, view the **logmessage** table in the **sysmaster** database:

```
SELECT * FROM logmessage;
```


Message Categories

Four general categories of unnumbered messages exist, although some messages fall into more than one category:

- Routine information
- Assertion-failed messages
- Administrative action needed
- Unrecoverable error detected

Technical Support uses the assertion-failed messages to assist in troubleshooting and diagnostics. The information that they report often falls into the category of *unexpected events* that might or might not develop into problems caught by other error codes. Moreover, the messages are terse and often extremely technical. They might report on one or two isolated statistics without providing an overall picture of what is happening. This information can suggest to technical support possible research paths.

Messages: A-B

Aborting Long Transaction: *tx 0xn.*

Cause

The transaction spans the log space specified by transaction high-watermark (LTXHWM), and the offending long transaction is rolling back.

Action

No additional action is needed. The address of the transaction structure in shared memory is displayed as a hexadecimal value.

Affinitied VP *mm* to phys proc *nn*.

Cause

The database server successfully bound a CPU virtual processor to a physical processor.

Action

None required.

Affinity not enabled for this server.

Cause

You tried to bind your CPU virtual processors to physical processors, but the database server that you are running does not support process affinity.

Action

Remove the affinity setting from the VPCLASS configuration parameter.

Assert Failed: Error from SBSpace cleanup thread.

Cause

The sbspace cleanup thread encountered an error while cleaning up stray smart large objects.

Action

See the action suggested in the message log file.

Most of the time, running **onspaces -cl sbspacename** on the failed sbspace succeeds in cleaning up any stray smart large objects. If you encounter an unrecoverable error, contact Technical Support.

Assert Failed: Short description of what failed Who: Description of user/session/thread running at the time Result: State of the affected database server entity Action: What action the database administrator should take See Also: DUMPDIR/af.uniqid containing more diagnostics.

Cause

This message indicates an internal error.

Action

The **af.uniqid** file in the directory specified by the ONCONFIG parameter DUMPDIR contains a copy of the assertion-failure message that was sent to the message log, as well as the contents of the current, relevant structures and/or data buffers. The information included in this message is intended for Technical Support.

Begin re-creating indexes deferred during recovery.

Cause

During recovery, indexes to be created are deferred until after recovery completes. This message indicates that the database server deferred re-creating indexes and that it is now creating the indexes. During the time that the database server re-creates the indexes, it locks the affected tables with a shared lock.

Action

None required.

Building 'sysmaster' database requires ~*mm* pages of logical log. Currently there are *nn* pages available. Prepare to back up your logs soon.

Cause

You do not currently have the approximate amount of free log space necessary to complete a build of the sysmaster database.

Action

Back up your logs.

Building 'sysmaster' database...

Cause

The database server is building the **sysmaster** database.

Action

None required.

Messages: C

Cannot Allocate Physical-log File, *mm* wanted, *nn* available.

Cause

The database server attempted to increase the size of the physical log, but it needed more contiguous space than was available in the dbspace. The quantities of space are expressed as kilobytes.

Action

You must either specify a smaller size for the physical log (use the PHYSFILE configuration parameter), or change the location of the physical log to a dbspace that contains adequate contiguous space to accommodate the larger physical log.

Cannot alter a table which has associated violations table.

Cause

The user tried to add, drop, or modify a column in a table that has a violations table associated with it.

Action

Do not change the columns in the user table.

Cannot change to mode.

Cause

Some error during fast or full recovery has prevented the system from changing to online or quiescent mode.

Action

See previous messages in the log file for information.

Cannot Commit Partially Complete Transactions.

Cause

Transactions that drop tables or indexes do not perform the drop until a COMMIT statement is processed (with a few exceptions). In these cases, a *beginning commit* log record is written, followed by the usual commit log record. If the database server fails in between the two, the fast recovery process attempts to complete the commit the next time that you initialize the database server.

If this completion of the commit fails, the database server generates the preceding message.

Action

To determine if you need to take action, examine the logical log as described in Interpreting Logical-Log Records.

Cannot create a user-defined VP class with 'SINGLE_CPU_VP' non-zero.

Cause

SINGLE_CPU_VP is set to nonzero, and **onmode** was used to create a user-defined VP class.

Action

If user-defined VP classes are necessary, stop the database server, change SINGLE_CPU_VP to zero, and restart the database server.

Cannot create violations/diagnostics table.

Cause

The user issued a START VIOLATIONS TABLE statement for a target table. The database server cannot create the violations table for this target table. Any of the following situations might be the reason for this failure:

- The target table already has a violations table.
- You specified an invalid name for the violations table in the START VIOLATIONS TABLE statement. For example, if you omit the USING clause from the statement and if the number of characters in the target table plus four characters is longer than the maximum identifier length, the generated names of the violations table exceed the maximum identifier length.
- You specified a name for the violations table in the START VIOLATIONS TABLE statement that match the names of existing tables in the database.
- The target table contains columns with the names **informix_tupleid**, **informix_optype**, or **informix_recowner**. Because these column names duplicate the **informix_tupleid**, **informix_optype**, or **informix_recowner** columns in the violations table, the database server cannot create the violations table.
- The target table is a temporary table.
- The target table is serving as a violations table for some other table.
- The target table is a system catalog table.

Action

To resolve this error, perform one of the following actions:

- If the violations table name was invalid, specify a unique name for the violations table in the USING clause of the START VIOLATIONS TABLE statement.
- If the target table contains columns with the names **informix_tupleid**, **informix_optype**, or **informix_recowner**, rename them to something else.
- Choose a permanent target table that is not a system catalog table or a violations table for some other table.

Cannot insert from the violations table to the target table.

Cause

The user has issued a statement that attempts to insert rows from the violations table into the target table. For example, the user enters the following invalid statement:

```
INSERT INTO mytable SELECT * FROM mytable_vio;
```

Also, if the target table has filtering-mode constraints, you receive this error.

Action

To recover from this error, perform the following actions:

- Do not use filtering constraints.
- Stop the violations table.
- Insert rows from the violations table into a temporary table, and then insert rows from the temporary table into the target table.

Cannot modify/drop a violations/diagnostics table.

Cause

The user has tried to alter or drop a table that is serving as a violations table for another table.

Action

Do not alter or drop the violations table.

Cannot Open Dbspace *nnn*.

Cause

The database server is unable to access the specified dbspace. This message indicates a problem opening the tblspace or corruption in the initial chunk of the dbspace.

Action

Verify that the device or devices that make up the chunks of this dbspace are functioning properly and that you assigned them the correct operating-system permissions (rw-rw----). You might be required to perform a data restore.

Cannot Open Logical Log.

Cause

The database server is unable to access the logical-log files. Because the database server cannot operate without access to the logical log, you must resolve this problem.

Action

Verify that the chunk device where the logical-log files reside is functioning and has the correct operating-system permissions (rw-rw----).

Cannot Open Mirror Chunk *pathname*, errorno = *nn*.

Cause

The database server cannot open the mirrored chunk of a mirrored pair. The chunk *pathname* and the operating-system error are returned.

Action

For more information about corrective actions, see your operating-system documentation.

Cannot Open Primary Chunk *pathname*, errorno = *nnn*.

Cause

The primary chunk of a mirrored pair cannot be opened. The chunk *pathname* and the operating-system error are returned.

Action

For more information about corrective actions, see your operating-system documentation.

Cannot Open Primary Chunk *chunkname*.

Cause

The *initial* chunk of the dbspace cannot be opened.

Action

Verify that the chunk device is running properly and has the correct operating-system permissions (rw-rw----).

Cannot open sysams in database *name*, iserrno *number*.

Cause

An error occurred when the database server opened the **sysams** system table.

Action

Note the error *number* and contact Technical Support.

Cannot open sysdistrib in database *name*, iserrno *number*.

Cause

An error occurred when the database server accessed the **sysdistrib** system table.

Action

Note the error *number* and contact Technical Support.

Cannot open *system_table* in database *name*, iserrno *number*.

Cause

An error occurred when the database server opened the specified system table.

Action

Note the error *number* and contact Technical Support.

Cannot open systrigbody in database *name*, iserrno *number*.

Cause

An error occurred when the database server accessed the **systrigbody** system table.

Action

Note the error *number* and contact Technical Support.

Cannot open systriggers in database *name*, iserrno *number*.

Cause

An error occurred when the database server accessed the **systriggers** system table.

Action

Note the error *number* and contact Technical Support.

Cannot open sysxtdtypes in database *name*, iserrno *number*.

Cause

An error occurred while accessing the **sysxtdtypes** system table.

Action

Note the error *number* and contact Technical Support.

Cannot Perform Checkpoint, shut system down.

Cause

A thread that is attempting to restore a mirrored chunk has requested a checkpoint, but the checkpoint cannot be performed.

Action

Shut down the database server.

Cannot Restore to Checkpoint.

Cause

The database server is unable to recover the physical log and thus unable to perform fast recovery.

Action

If the database server does not come online, perform a data restore from dbspace backup.

Cannot Rollback Incomplete Transactions.

Cause

Within the fast-recovery or data-restore procedure, the logical-log records are first rolled forward. Then, open transactions that have not committed are rolled back. An open transaction could fail during the rollback, leaving some of the modifications from the open transaction in place. This error does not prevent the database server from moving to quiescent or online mode, but it might indicate an inconsistent database.

Action

To determine if any action is needed, use the onlog utility to examine the logical log.

Cannot update pagezero.

Cause

A failure occurred while the database server was trying to rewrite a reserved page during the reversion process.

Action

See previous messages in the log file for information, or contact Technical Support.

Cannot update syscasts in database *name*. Iserrno *number*.

Cause

An internal error occurred while inserting data into the **syscasts** system table.

Action

Contact Technical Support..

Can’t affinity VP *mm* to phys proc *nn*.

Cause

The database server supports process affinity, but the system call to bind the virtual processor to a physical processor failed.

Action

See your operating-system documentation.

Changing the sbspace minimum extent value: old value *value1*, new value *value2*.

Cause

This informational message occurs when you issue the following command:

```
onspaces -ch sbspace -Df "MIN_EXT_SIZE=value1" -y
```

Action

None. For more information, see onspaces -ch: Change sbspace default specifications.

Checkpoint blocked by down space, waiting for override or shutdown.

Cause

A dbspace has gone down during a checkpoint interval. The database server is configured to wait for an override when this situation occurs.

Action

Either shut down the database server or issue an **onmode -O** command to override the down dbspace. For more information on the **onmode** utility, see The onmode utility.

Checkpoint Completed: duration was *n* seconds.

Cause

A checkpoint completed successfully.

Action

None required.

Checkpoint Page Write Error.

Cause

The database server detected an error in an attempt to write checkpoint information to disk.

Action

For additional assistance in resolving this situation, contact Technical Support.

Checkpoint Record Not Found in Logical Log.

Cause

The logical log or the chunk that contains the logical log is corrupted. The database server cannot initialize.

Action

Perform a data restore from dbspace backup.

Chunk *chunkname* added to space *spacename.*

Cause

The variables in this message have the following values:

chunkname
: is the name of the chunk that the database server administrator is adding.

spacename
: is the name of the storage space to which the database server administrator is adding the chunk.

Action

None required.

Chunk *chunkname* dropped from space *spacename.*

Cause

The database server administrator dropped chunk *chunkname* from space *spacename*.

Action

None required.

Chunk *number nn pathname* -- Offline.

Cause

The indicated chunk in a mirrored pair has been marked with status `D` and taken offline. The other chunk in the mirrored pair is operating successfully.

Action

Take steps now to repair the chunk device and restore the chunk. The chunk *number* and chunk device *pathname* are displayed.

Chunk *number nn pathname* -- Online.

Cause

The indicated chunk in a mirrored pair has been recovered and is online (marked with status `O`). The chunk *number* and chunk device *pathname* are displayed.

Action

None required.

The chunk *pathname* must have READ/WRITE permissions for owner and group.

Cause

The chunk *pathname*does not have the correct owner and group permissions.

Action

Make sure that you assigned the correct permissions (-rw-rw---) to the device on which the chunk is located.

The chunk *pathname* must have *owner-ID* and *group-ID* set to informix.

Cause

The chunk *chunkname* does not have the correct owner and group ID.

Action

Make sure the device on which the chunk is located has the ownership. On UNIX, both owner and group should be **informix**. On Windows, the owner must be a member of the **Informix-Admin** group.

The chunk *pathname* will not fit in the space specified.

Cause

The chunk *pathname* does not fit in the space that you specified.

Action

Choose a smaller size for the chunk, or free space where the chunk is to be created.

Cleaning stray LOs in sbspace *sbspacename*.

Cause

The database server administrator is running **onspaces -cl sbspacename**.

Action

None required.

Completed re-creating indexes.

Cause

The database server finished re-creating the deferred indexes.

Action

None required.

Configuration has been grown to handle up to *integer* chunks.

Cause

The database server administrator increased the number of chunks to the specified value by changing CONFIGSIZE or setting MAX_CHUNKS to a higher value.

Action

None required. The change was successful.

Configuration has been grown to handle up to *integer* dbslices.

Cause

The database server administrator increased the number of dbslices to the specified value by changing CONFIGSIZE or setting MAX_DBSLICES to a higher value.

Action

None required. The change was successful.

Configuration has been grown to handle up to *integer* dbspaces.

Cause

The database server administrator increased the number of dbspaces to the specified value by changing CONFIGSIZE or setting MAX_DBSPACES to a higher value.

Action

None required. The change was successful.

Continuing Long Transaction (for COMMIT): *tx 0xn*.

Cause

The logical log has filled beyond the long-transaction high-watermark (LTXHWM), but the offending long transaction is in the process of committing. In this case, the transaction is permitted to continue writing to the logical log and is not rolled back. The address of the transaction structure in shared memory is displayed as hexadecimal value *tx 0xn*.

Action

None required.

Could not disable priority aging: errno = *number*.

Cause

An operating-system call failed while it was trying to disable priority aging for the CPU virtual processor. The system error *number* associated with the failure is returned.

Action

See your operating-system documentation.

Could not fork a virtual processor: errno = *number*.

Cause

The fork of a virtual processor failed. The database server returns the operating-system error *number* associated with the failure.

Action

For information on determining the maximum number of processes available per user and for the system as a whole, refer to your operating-system documentation.

Create_vp: cannot allocate memory.

Cause

The database server cannot allocate new shared memory.

Action

The database server administrator must make more shared memory available. This situation might require increasing SHMTOTAL or reconfiguring the operating system. This message is usually accompanied by other messages that give additional information.

Messages: D-E-F

Dataskip is OFF for all dbspaces.

Cause

Informational.

Action

None required.

Dataskip is ON for all dbspaces.

Cause

Informational.

Action

None required.

Dataskip is ON for dbspaces: *dbspacelist*.

Cause

Informational; DATASKIP is ON for the specified dbspaces.

Action

None required.

Dataskip will be turned {ON|OFF} for *dbspacename*.

Cause

Informational; DATASKIP is ON or OFF for the specified dbspace.

Action

None required.

DBSERVERALIASES exceeded the maximum limit of 32

Cause

The limit of 32 aliases was reached.

Action

Nothing. Only the first 32 will be used.

DBSPACETEMP internal list not initialized, using default.

Cause

An error occurred while initializing a user-specified DBSPACETEMP list. Typically this condition is due to a memory-allocation failure.

Action

Check for accompanying error messages.

The DBspace/BLOBspace *spacename* is now mirrored.

Cause

You successfully added mirroring to the indicated storage space.

Action

None required.

The DBspace/BLOBspace *spacename* is no longer mirrored.

Cause

You have ended mirroring for the indicated storage space.

Action

None required.

devname: write failed, file system is full.

Cause

Because the file system *devname* is full, the write failed.

Action

Free some space in *devname*.

Dropping temporary tblspace *0xn*, recovering *nn* pages.

Cause

During shared-memory initialization, the database server routinely searches for temporary tables that are left without proper cleanup. If the database server finds a temporary table, it drops the table and recovers the space. The database server located the specified temporary tblspace and dropped it. The value *0xn* is the hexadecimal representation of the tblspace number.

Action

None required.

Dynamically allocated new shared memory segment (size *nnnn*).

Cause

This status message informs you that the database server successfully allocated a new shared-memory segment of size *nnnn*.

Action

None required.

ERROR: NO "wait for" locks in Critical Section.

Cause

The database server does not permit a thread to own locks that might have to wait while that thread is within a critical section. Any such lock request is denied, and an ISAM error message is returned to the user.

Action

The error reported is an internal error. Contact IBM® Informix® Technical Support.

Error building sysmaster database. See *outfile*.

Cause

Errors were encountered in building the sysmaster database. The file *outfile* contains the result of running the script buildsmi.

Action

See the file *outfile*.

Error in dropping system defined type.

Cause

An internal error occurred while updating either the **sysxtdtypes**, **sysctddesc**, or **sysxtdtypeauth** system table.

Action

Contact Technical Support.

Error in renaming systdist.

Cause

An internal error occurred while trying to find and rename the **Informix®.systdist** SPL routine.

Action

Contact Technical Support.

Error removing sysdistrib row for tabid = *tabid*, colid = *colid* in database *name*. iserrno = *number*

Cause

An error occurred while updating the **sysdistrib** system table.

Action

Note the error *number* and contact Technical Support.

Error writing *pathname* errno = *number*.

Cause

The operating system cannot write to *pathname. Number* is the number of the operating-system error that was returned.

Action

Investigate the cause of the operating-system error. Usually it means that no space is available for the file. It might also mean that the directory does not exist or that no write permissions exist.

Error writing shmem to file *filename* (*error*). Unable to create output file *filename* errno=*mm*.Error writing *filename* errno=*nn*.

Cause

The database server detected an error in an attempt to write shared memory to *filename*. The first message is followed by one of the next two. Either the attempt failed because the output file could not be created or because the contents of shared memory could not be written. The error refers to the operating-system error that prompted the attempted write of shared memory to a file. The value of *nn* is the operating-system error.

Action

See your operating-system documentation.

Fail to extend physical log space.

Cause

The attempt to extend the physical log space failed. Either the path does not exist or the permissions are incorrect.

Action

Use a path that exists. Check permissions on the current working directory. You or the system administrator must give your group execute permission on the current working directory. After your group has been given permission, retry the operation that generated this message.

Fatal error initializing CWD string. Check permissions on current working directory. Group *groupname* must have at least execute permission on '.'.

Cause

Group *groupname* does not have execute permission for the current working directory.

Action

Check permissions on the current working directory. You or the system administrator must give your group execute permission on the current working directory. After your group has been given permission, retry the operation that generated this message.

Fragments *dbspacename1 dbspacename2* of table *tablename* set to non-resident.

Cause

The specified fragments of *tablename* either have been set to nonresident by the SET TABLE statement.

Action

None required.

Forced-resident shared memory not available.

Cause

The database server port for your computer does not support forced-resident shared memory.

Action

None required.

Freed *mm* shared-memory segment(s) *number* bytes.

Cause

The database server sends this message to the message log after you run the **-F** option of the **onmode** utility to free unused memory. The message informs you of the number of segments and bytes that the database server successfully freed.

Action

None required.

Messages: G-H-I

gcore *pid*; mv core.*pid* dir/core.*pid*.ABORT.

Cause

This status message during a database server failure provides the name and place of each core file associated with the virtual processors.

Action

None required.

I/O *function* chunk *mm*, pagenum *nn*, pagecnt *aa* --> errno = *bb*.

Cause

An operating-system error occurred during an attempt to access data from disk space. The operating-system function that failed is defined by *function*. The chunk number and physical address of the page where the error occurred are displayed as integers. The *pagecnt* value refers to the number of pages that the thread was attempting to read or write. If an *errno* value is displayed, it is the number of the operating-system error and might explain the failure. If *function* is specified as *bad request*, some unexpected event caused the I/O attempt on an invalid chunk or page.

Action

If the chunk status changes to `D`, or down, restore the chunk from its mirror or repair the chunk. Otherwise, perform a data restore.

I/O error, *primary/mirror* Chunk *pathname* -- Offline (*sanity*).

Cause

The database server detected an I/O error on a primary or mirror chunk with *pathname*. The chunk was taken offline.

Action

Check that the device on which the chunk was stored is functioning as intended.

Deleted Indexes idx1 and idx 2 error message

Informix® *database_server* Initialized - Complete Disk Initialized.

Cause

Disk space and shared memory have been initialized. Any databases that existed on the disk before the initialization are now inaccessible.

Action

None required.

Informix® *database_server* Initialized - Shared Memory Initialized.

Cause

Shared memory has been initialized.

Action

None required.

Informix® *database_server* Stopped.

Cause

The database server has moved from quiescent mode to offline mode. The database server is offline.

Action

None required.

In-Place Alter Table. Perform EXECUTE FUNCTION sysadmin:task('table update_ipa', 'table_name','database');

Cause

Reversion to a previous version of the database server was attempted while an in-place alter operation is in progress on a table. The previous versions of the database server cannot handle tables that have multiple schemas of rows in them.

Action

Force any in-place alter operations to complete by updating the rows in the affected tables before you attempt to revert to a previous version of the database server. Run the SQL administration API **task()** or **admin()** command with the **table update_ipa** argument to resolve all pending in-place alter operations on the table.

ERROR: Insufficient available disk in the root dbspace to increase the entire Configuration save area.

Cause

The user attempted to increase the number of storage objects to a specific value by changing CONFIGSIZE or setting MAX_DBSPACES, MAX_DBSLICES, or MAX_CHUNKS to a higher value, but the database server did not have enough rootspace for the increased number of storage objects. A storage object might be a dbspace, dbslice, or chunk.

Action

Increase the size of the root dbspace or reset CONFIGSIZE, MAX_DBSPACES, MAX_DBSLICES, or MAX_DBSLICES to a lower value and restart the database server. For example, if you set MAX_CHUNKS to `32,768`, but the root dbspace did not have enough space, set MAX_CHUNKS to a lower value.

Insufficient available disk in the root dbspace for the CM save area. Increase the size of the root dbspace in the ONCONFIG file and reinitialize the server.

Cause

The cause might be one of the following:

- The user attempted to increase the number of storage objects to a specific value by changing CONFIGSIZE or setting MAX_DBSPACES, MAX_DBSLICES, or MAX_CHUNKS to a higher value, but the database server did not have enough rootspace for the increased number of storage objects. A storage object might be a dbspace, dbslice, or chunk.
- The user converted to a database server version that requires slightly more rootspace, but it is not available (this case is unlikely).

Action

Take one of the following actions:

- Increase the size of the root dbspace or reset CONFIGSIZE, MAX_DBSPACES, MAX_DBSLICES, or MAX_DBSLICES to a lower value and restart the database server. For example, if you set MAX_DBSPACES to `32,768` but the root dbspace did not have enough space, set MAX_DBSPACES to a lower value.
- Increase the size of the root dbspace and reinitialize the database server.

Internal overflow of shmid's, increase system max shared memory segment size.

Cause

The database server was initializing shared memory when it ran out of internal storage for the shared-memory IDs associated with this segment.

Action

Increase the value of your maximum kernel shared-memory segment size, usually SHMMAX. For more information, see your operating-system documentation.

Messages: J-K-L-M

- Logical Recovery Started.
- Maximum server connections number.
- Memory allocation error.
- Mirror Chunk chunkname added to space spacename. Perform manual recovery.
- Mixed transaction result. (pid=nn user=userid).
- mt_shm_free_pool: pool 0xn has blocks still used (id nn).
- mt_shm_init: can't create resident/virtual segment.
- mt_shm_remove: WARNING: may not have removed all/correct segments.

Listener-thread err = *error_number*: *error_message.*

Cause

A listener thread has encountered an error. This message displays the error number and message text.

Action

For a description of an error message, use the **finderr** utility or go to Error messages.

Lock table overflow - user id *mm* session id *nn.*

Cause

A thread attempted to acquire a lock when no locks were available. The user ID and session ID are displayed.

Action

Increase the LOCKS configuration parameter, and initialize shared memory.

Logical-log File not found.

Cause

The checkpoint record in the root dbspace reserved page is corrupted.

Action

Perform a data restore from dbspace backup.

Logical Log *nn* Complete.

Cause

The logical-log file identified by log-ID number *nn* is full. The database server automatically switches to the next logical-log file in the sequence.

Action

None required.

Logical logging *vberror* for *type:subtype* in (*failed_system*).

Cause

Logging failed. The log record that caused the error is identified as follows:

type
: Is the logical-log record type.

subtype
: Is the logging subsystem.

failed_system
: Is the name of an internal function that indicates what system failed to log.

Action

Contact Technical Support.

Log Record: log = *ll*, pos = *0xn*, type = *type:subtype(snum),* trans = *xx*

Cause

The database server detected an error during the rollforward portion of fast recovery or logical-log restore.

The log record that caused the error is identified as follows:

ll
: Is the logical-log ID where the record is stored.

0xn
: Is the hexadecimal address position within the log.

type
: Is the logical-log record type.

subtype
: Is the logging subsystem.

snum
: Is the subsystem number.

xx
: Is the transaction number that appears in the logical log.

Action

Contact Technical Support.

Log record (*type:subtype*) at log *nn*, *0xn* was not undone.

Cause

A log undo failed because a log is corrupt.

The log record that caused the error is identified as follows:

type
: Is the logical-log record type.

subtype
: Is the logging subsystem.

nn
: Is the logical-log ID where the record is stored.

0xn
: Is the hexadecimal address position within the log.

Action

To determine if any action is needed, use the onlog utility to examine the logical log. Contact Technical Support.

Log record (*type:subtype*) failed, partnum *pnum row rid* iserrno *num*.

Cause

A logging failure occurred.

The log record that caused the error is identified as follows:

type
: Is the logical-log record type.

subtype
: Is the logging subsystem.

pnum
: Is the part number.

rid
: Is the row ID.

num
: Is the iserror number.

Action

Contact Technical Support.

Log record (*type:subtype*) in log *nn*, offset *0xn* was not rolled back.

Cause

A log undo failed because a log is corrupt.

The log record that caused the error is identified as follows:

type
: Is the logical-log record type.

subtype
: Is the logging subsystem.

log
: Is the logical-log ID where the record is stored.

offset
: Is the hexadecimal address position within the log.

Action

To determine if any action is needed, use the onlog utility to examine the logical log. Contact Technical Support.

Logical Recovery allocating *nn* worker threads *thread_type*.

Cause

The database server determined the number of worker threads that will be used for parallel recovery. The variable *thread_type* can assume the values ON_RECVRY_THREADS or OFF_RECVRY_THREADS.

Action

This status message requires no action. If you want a different number of worker threads allocated for parallel recovery, change the value of the ONCONFIG configuration parameter ON_RECVRY_THREADS or OFF_RECVRY_THREADS.

Logical Recovery Started.

Cause

Logical recovery began.

Action

This status message requires no action.

Maximum server connections *number*.

Cause

Outputs with each checkpoint message to indicate the maximum number of concurrent connections to the database server since the last restart.

Action

This message helps the customer track license usage to determine when more licenses need to be purchased. For assistance, Contact Technical Support.

Memory allocation error.

Cause

The database server ran out of shared memory.

Action

Take one of the following actions:

1. Increase swap space on the computer.
2. Check kernel shared-memory parameters for limits on shared memory.
3. Decrease the size of the memory allocated, with the **buffers** field in the BUFFERPOOL configuration parameter.
4. Increase the virtual-memory size (SHMVIRTSIZE), the size of the added segments, (SHMADD), or your total shared-memory size (SHMTOTAL).

Mirror Chunk *chunkname* added to space *spacename*. Perform manual recovery.

Cause

Fast recovery, full recovery, or an HDR secondary has recovered the add of a mirror chunk. It does not perform automatic mirror recovery, however. The administrator must do this.

Action

Use the **onspaces** utility to attempt to recover the mirror chunks.

Mixed transaction result. (*pid=nn* user=*userid*).

Cause

You receive this message only when more than one database server is involved in a transaction. This message indicates that a database server, after preparing a transaction for commit, heuristically rolled back the transaction, and the global transaction completed inconsistently. The *pid* value is the user-process identification number of the coordinator process. The value of *user* is the user ID associated with the coordinator process.

Action

See the information on recovering manually from failed two-phase commit in your *IBM® Informix® Administrator's Guide*.

mt_shm_free_pool: pool *0xn* has blocks still used (id *nn*).

Cause

An internal error occurred during a pool deallocation because blocks are still associated with the pool.

Action

Contact Technical Support.

mt_shm_init: can't create *resident/virtual* segment.

Cause

The causes for the failure to create the resident or virtual segment are as follows: (1) the segment size is less than the minimum segment size; (2) the segment size is larger than the maximum segment size; (3) allocating another segment would exceed the allowable total shared-memory size; or (4) a failure occurred while the database server was trying to allocate the segment.

Action

If you suspect that this error was generated because of item 1 or 2 in the preceding paragraph, Contact Technical Support. To correct item 3, increase the SHMTOTAL value in your ONCONFIG configuration file. For additional information about errors generated because of item 4, see your logical-log file.

mt_shm_remove: WARNING: may not have removed all/correct segments.

Cause

When the operating system tried to remove the shared-memory segments associated with the database server, the last segment did not equal the last segment registered internally. This situation is probably due to the unexpected failure of the database server.

Action

Remove any segments that were not cleaned up.

Messages: N-O-P

- Physical recovery started at page (chunk:offset).
- Portions of partition partnum of table tablename in database dbname were not logged. This partition cannot be rolled forward.
- Possible mixed transaction result.
- Prepared participant site server_name did not respond.
- Prepared participant site server_name not responding.

Newly specified value of *value* for the pagesize in the configuration file does not match older value of *value*. Using the older value.

Cause

This message displays upon database server restart. The PAGESIZE value changed in the ONCONFIG file after the database server was initialized.

Action

The database server uses the older PAGESIZE value.

Not enough main memory.

Cause

The database server detected an error in an attempt to acquire more memory space from the operating system.

Action

For more information about shared-memory configuration and management, refer to your operating-system documentation.

Not enough logical-log files, Increase LOGFILES.

Cause

During a data restore, the value of the LOGFILES configuration must always be greater than or equal to the total number of logical-log files. At some point during the restore, the number of logical-log files exceeded the value of LOGFILES.

Action

Increase the value of LOGFILES in ONCONFIG.

The number of configured inline poll threads exceeds the number of CPU virtual processors.

Cause

This message is generated when the number of inline poll threads specified by the NETTYPE configuration parameter exceeds the number of CPU virtual processors configured by the VPCLASS configuration parameter. Poll threads that are configured to run on CPU virtual processors are referred to as inline poll threads.

Action

Either modify the VPCLASS configuration parameter to increase the number of CPU virtual processors, or modify the NETTYPE configuration parameter to decrease the number of inline poll threads.

Related reference:
NETTYPE configuration parameter
VPCLASS configuration parameter

onconfig parameter *parameter* modified from *old_value* to *new_value.*

Cause

When the database server shared memory is reinitialized, this message documents any changes that occurred since the last initialization.

Action

None required.

oninit: Cannot have SINGLE_CPU_VP non-zero and number of CPU VPs greater than 1.

Cause

The ONCONFIG file contains VPCLASS cpu with a `num=` value greater than `1` and a nonzero value for SINGLE_CPU_VP. SINGLE_CPU_VP must be 0 (or omitted) when there are more than 1 CPU VPs.

Action

Correct the ONCONFIG file and restart the database server.

oninit: Cannot have SINGLE_CPU_VP non-zero and user-defined VP classes.

Cause

The ONCONFIG file contains a user-defined VPCLASS as well as a nonzero value for SINGLE_CPU_VP. SINGLE_CPU_VP must be `0` (or omitted) when the ONCONFIG file contains a user-defined VPCLASS.

Action

Correct the ONCONFIG file and restart the database server.

oninit: Fatal error in initializing ASF with 'ASF_INIT_DATA' flags asfcode = '25507'.

Cause

The **nettype** value specified in the **sqlhosts** file or registry for the database server is invalid or unsupported, or the **servicename** specified in the **sqlhosts** file or registry for the database server is invalid.

Action

Check the **nettype** and **servicename** values in the **sqlhosts** file or registry for each DBSERVERNAME and for the DBSERVERALIASES. Check the **nettype** value in each NETTYPE parameter in the ONCONFIG file.

Cannot alter a table which has associated violations table.

Cause

The user tried to add, drop, or modify a column in a table that has a violations table associated with it.

Action

Do not change the columns in the user table.

oninit: Too many VPCLASS parameters specified.

Cause

Too many VPCLASS parameter lines have been specified in the ONCONFIG file.

Action

Reduce the number of VPCLASS lines, if possible. If not possible, contact Technical Support.

oninit: VPCLASS *classname* bad affinity specification.

Cause

The affinity specification for the VPCLASS line is incorrect. Affinity is specified as a range:

For *m,* use processor *m*.

For *m* to *n,* use processors in the range *m* to *n* inclusive, where *m* <=*n*, *m* >= 0, and *n* >= 0.

Action

Correct the VPCLASS parameter in the ONCONFIG file and restart the database server.

oninit: VPCLASS *classname* duplicate class *name*.

Cause

The VPCLASS *classname* in the ONCONFIG file has a duplicate name. VP class names must be unique.

Action

Correct the duplicate name and restart the database server.

oninit: VPCLASS *classname* illegal option.

Cause

One of the fields in the VPCLASS *classname* parameter is illegal.

Action

Correct the parameter in the ONCONFIG file and restart the database server.

oninit: VPCLASS *classname* maximum number of VPs is out of the range 0-10000.

Cause

The maximum number of VPs specified by a VPCLASS parameter line must be in the range 1 to 10,000.

Action

Correct the value and restart the database server.

oninit: VPCLASS *classname* name is too long. Maximum length is *maxlength*.

Cause

The length of the name field in VPCLASS *classname* is too long.

Action

Choose a shorter class name, correct the ONCONFIG file, and restart the database server.

oninit: VPCLASS *classname* number of VPs is greater than the maximum specified.

Cause

The initial number of VPs specified by a VPCLASS parameter is greater than the maximum specified by the same VPCLASS parameter.

Action

Correct the VPCLASS parameter and restart the database server.

oninit: VPCLASS *classname* number of VPs is out of the range 0-10000.

Cause

The initial number of VPs specified by a VPCLASS parameter line must be in the range 1 to 10,000.

Action

Correct the value and restart the database server.

onmode: VPCLASS *classname* name is too long. Maximum length is *maxlength*.

Cause

The name of a dynamically added VP class that **onmode -p** specifies is too long.

Action

Choose a shorter name, and retry the **onmode -p** command.

Online Mode.

Cause

The database server is in online mode. Users can access all databases

Action

This status message requires no action.

onspaces: unable to reset dataskip.

Cause

This error message comes from the **onspaces** utility. For some reason, the utility cannot change the specification of DATASKIP (ON or OFF) across all dbspaces in the database server instance.

Action

You are unlikely to receive this message. If the error persists after you restart the database server, Contact Technical Support.

Open transaction detected when changing log versions.

Cause

The database server detected an open transaction while it was trying to convert the data from a previous version of the database server.

Action

Conversion is not allowed unless the last record in the log is a checkpoint. You must restore the previous version of the database server, force a checkpoint, and then retry conversion.

Out of message shared memory.

Cause

The database server could not allocate more memory for the specified segment.

Action

For additional information, see the log file.

Out of resident shared memory.

Cause

The database server could not allocate more memory for the specified segment.

Action

For additional information, see the log file.

Out of virtual shared memory.

Cause

The database server could not allocate more memory for the specified segment.

Action

For additional information, see the log file.

PANIC: Attempting to bring system down.

Cause

A fatal database server error occurred.

Action

See the error that caused the panic and attempt the corrective action suggested by the error message. For additional information that might explain the failure, refer also to other messages in the message-log file.

Participant site *database_server* heuristically rolled back.

Cause

A remote site rolled back a transaction after it reached the prepared-for-commit phase.

Action

You might need to roll back the transaction on other sites and then restart it.

Physical recovery complete: *number* pages examined, *number* pages restored.

Cause

This message displays during fast recovery. The *number of pages examined* indicates the number of page images that exist in the physical log. The *number of pages restored* indicates the actual number of pages that are restored from the physical log. The number of pages restored is always less than or equal to the number examined.

The database server might physically log a page image multiple times between checkpoints. Physical recovery restores only the first logged page image.

If a page stays in the memory buffer pool, the database server physically logs it once per checkpoint, and stores one page image in the physical log. If the buffer pool is too small, a page that is being updated many times might get forced out of the buffer pool to disk and then brought back into memory for the next update. Each time the page is brought into memory, it is physically logged again, resulting in duplicate page images in the physical log.

Action

If the *number of pages examined* is much larger than the *number of pages restored,* increase the size of the buffer pool to reduce the number of duplicate before-images. For more information, see the *IBM® Informix® Performance Guide*.

Physical recovery started at page (*chunk*:*offset*).

Cause

This message displays during fast recovery. *Chunk* is the number of the chunk that contains the physical log. *Offset* is the page offset of the start of the physical log entries. Physical recovery begins restoring pages from that point.

Action

No action required. For information on fast recovery, see the *IBM® Informix® Administrator's Guide*.

Portions of partition partnum of table tablename in database dbname were not logged. This partition cannot be rolled forward.

Cause

Light appends occurred to the operational table since the last backup.

Action

If you want full access to data in this table, you need to alter the table to raw and then to the desired table type. This alter operation removes inconsistencies in the table that resulted from replaying non-logged operations such as light appends.

Possible mixed transaction result.

Cause

This message indicates that error -716 has been returned. Associated with this message is a list of the database servers where the result of a transaction is unknown.

Action

For information on determining if a transaction was implemented inconsistently, see the *IBM® Informix® Administrator's Guide*.

Prepared participant site *server_name* did not respond.

Cause

Too many attempts were made to contact remote site *server_name.* After several timeout intervals were met, the site was determined to be down.

Action

Verify that the remote site is online and that it is correctly configured for distributed transactions. Once the remote site is ready, reinitiate the transaction.

Prepared participant site *server_name* not responding.

Cause

The database server is attempting to contact remote site *server_name*. For some unknown reason, the database server cannot contact the remote site.

Action

Verify that the remote site is online and that it is correctly configured for distributed transactions.

Messages: Q-R-S

Quiescent Mode.

Cause

The database server has entered quiescent mode from some other state. On UNIX, only users logged in as **informix** or as **root** can interact with the database server. On Windows, only members of the **Informix-Admin** group can interact with the database server. No user can access a database.

Action

None required.

Read failed. Table *name*, Database *name*, iserrno = *number*

Cause

An error occurred reading the specified system table.

Action

Note the error number and contact Technical Support.

Recovery Mode.

Cause

The database server entered the recovery mode. No user can access a database until recovery is complete.

Action

None required.

Recreating index: '*dbsname*:"*owner*".*tabname-idxname*'.

Cause

After DDL statements implicitly or explicitly create one or more new indexes, but the database server terminates abnormally before the next checkpoint, re-creation of the new indexes is deferred until after logical recovery, instead of adding each index item row by row. After logical recovery ends, the server begins a parallel index build to re-create them. This message indicates when re-creation commences for each deferred index. (But if an index was dropped before the abnormal shutdown, it will not be re-created after logical recovery, and no message referencing that index will be printed.)

Action

None required.

Rollforward of log record failed, iserrno = *nn*.

Cause

The message appears if, during fast recovery or a data restore, the database server cannot roll forward a specific logical-log record. The database server might be able to change to quiescent or online mode, but some inconsistency could result. For further information, see the message that immediately precedes this one. The *iserrno* value is the error number.

Action

Contact IBM® Informix® Technical Support.

Root chunk is full and no additional pages could be allocated to chunk descriptor page.

Cause

The root chunk is full.

Action

To free space in the root chunk, take one of the following actions:

- Drop and re-create the **sysmaster** database.
- Move user tables from the root dbspace to another dbspace.
- Refragment tables.

scan_logundo: subsys *ss*, type *tt*, iserrno *ee*.

Cause

A log undo failed because log type *tt* is corrupt.

The variables in this message have the following values:

ss
: Is the subsystem name.

tt
: Is the logical-log record type.

ee
: Is the iserror number.

Action

Examine the logical log with the onlog utility to determine if any action is needed. Contact Technical Support.

Session completed abnormally. Committing *tx* id *0xm*, flags *0xn*.

Cause

Abnormal session completion occurs only when the database server is attempting to commit a transaction that has no current owner, and the transaction develops into a long transaction. The database server forked a thread to complete the commit.

Action

None required.

Session completed abnormally. Rolling back *tx* id *0xm*, flags *0xn*.

Cause

Abnormal session completion occurs only when the database server is attempting to commit a distributed transaction that has no current owner, and the transaction develops into a long transaction. The database server forked a thread that rolled back the transaction.

Action

None required.

semctl: errno = *nn*.

Cause

When the database server initialized a semaphore, an error occurred. The operating-system error is returned.

Action

See your operating-system documentation.

semget: errno = *nn*.

Cause

An allocation of a semaphore set failed. The operating-system error is returned.

Action

See your operating-system documentation.

shmat: *some_string os_errno*: *os_err_text*.

Cause

An attempt to attach to a shared-memory segment failed. The system error number and the suggested corrective action are returned.

Action

Review the corrective action (if given), and determine if it is reasonable to try. For more information, refer to your operating-system documentation.

shmctl: errno = *nn.*

Cause

An error occurred while the database server tried to remove or lock a shared-memory segment. The operating-system error number is returned.

Action

See your operating-system documentation.

shmdt: errno = *nn.*

Cause

An error occurred while the database server was trying to detach from a shared-memory segment. The operating-system error number is returned.

Action

See your operating-system documentation.

shmem sent to *filename.*

Cause

The database server wrote a copy of shared memory to the specified file as a consequence of an assertion failure.

Action

None.

shmget: *some_str os_errno*: key *shmkey*: *some_string.*

Cause

Either the creation of a shared-memory segment failed, or an attempt to get the shared-memory ID associated with a certain key failed. The system error number and the suggested corrective action are returned.

Action

Consult your operating-system documentation.

Shutdown (onmode -k) or override (onmode -O).

Cause

A dbspace has gone down during a checkpoint interval. The database server is configured to wait for an override when this situation occurs.

When the checkpoint actually happens, the following message appears: `Checkpoint blocked by down space, waiting for override or shutdown.`

Action

Either shut down the database server or issue an **onmode -O** command to override the down dbspace. For more information on the **onmode** utility, see The onmode utility.

Shutdown Mode.

Cause

The database server is in the process of moving from online mode to quiescent mode.

Action

None required.

Space *spacename* added.

Cause

The database server administrator added a new storage space *spacename* to the database server.

Action

None required.

Space *spacename* dropped.

Cause

The database server administrator dropped a storage space *spacename* from the database server.

Action

None required.

Space *spacename* -- Recovery Begins(*addr*).

Cause

This informational message indicates that the database server is attempting to recover the storage space.

The variables in this message have the following values:

spacename
: Is the name of the storage space that the database server is recovering.

addr
: Is the address of the control block.

Action

None required.

Space *spacename* -- Recovery Complete(*addr*).

Cause

This informational message indicates that the database server recovered the storage space.

The variables in this message have the following values:

spacename
: Is the name of the storage space that the database server has recovered.

addr
: Is the address of the control block.

Action

None required.

Space *spacename* -- Recovery Failed(*addr*).

Cause

This informational message indicates that the database server was unable to recover the storage space.

The variables in this message have the following values:

spacename
: Is the name of the storage space that the database server failed to recover.

addr
: Is the address of the control block.

Action

None required.

sysmaster database built successfully.

Cause

The database server successfully built the sysmaster database.

Action

None required.

Successfully extend physical log space

Cause

The physical log space was successfully extended to the file plog_extend.*servernum* under the designated path.

Action

None required.

Messages: T-U-V

This ddl operation is not allowed due to deferred constraints pending on this table and dependent tables.

Cause

This error is returned when you attempt to start a violations table and constraints are in deferred mode.

Note: No error is returned if you start a violations table and then later set the constraints to deferred. However, the violations get undone immediately rather than written into the deferred constraint buffer. For more information, see the *IBM® Informix® Guide to SQL: Syntax*.

Action

If you would like to start a violations table, you must either change the constraint mode to immediate or commit the transaction.

This type of space does not accept log files.

Cause

Adding a logical-log file to a blobspace or sbspace is not allowed.

Action

Add the logical-log file to a dbspace. For more information, see onparams -a -d dbspace: Add a logical-log file.

TIMER VP: Could not redirect I/O in initialization, errno = *nn*.

Cause

The operating system could not open the null device or duplicate the file descriptor associated with the opening of that device. The system error number is returned.

Action

See your operating-system documentation.

Too Many Active Transactions.

Cause

During a data restore, there were too many active transactions. At some point during the restore, the number of active transactions exceeded 32 kilobytes.

Action

None.

Too many violations.

Cause

The number of violations in the diagnostics table exceeds the limit that is specified in the MAX VIOLATIONS clause of the START VIOLATIONS TABLE statement. When a single statement on the target table (such as an INSERT or UPDATE statement) inserts more records into the violations table than the limit that is specified by the MAX VIOLATIONS clause, this error is returned to the user who issued the statement on the target table.

Action

To resolve this error, perform one of the following actions:

- Omit the MAX VIOLATIONS clause in the START VIOLATIONS TABLE statement when you start a violations table. Here, you are specifying no limit to the number of rows in the violations table.
- Set MAX VIOLATIONS to a high value.

Transaction Not Found.

Cause

The logical log is corrupt. This situation can occur when a new transaction is started, but the first logical-log record for the transaction is not a BEGWORK record.

Action

Contact Technical Support.

Transaction heuristically rolled back.

Cause

A heuristic decision occurred to roll back a transaction after it completed the first phase of a two-phase commit.

Action

None required.

Transaction table overflow - user id *nn*, process id *nn*.

Cause

A thread attempted to allocate an entry in the transaction table when no entries in the shared-memory table were available. The user ID and process ID of the requesting thread are displayed.

Action

Try again later.

Unable to create output file *filename* errno = *nn*.

Cause

The operating system cannot create output file *filename*. The *errno* is the number of the operating-system error returned.

Action

Verify that the directory exists and has write permissions.

Unable to extend *nn* reserved pages for *purpose* in root chunk.

Cause

The operating system cannot extend to *nn* reserved pages for *purpose* in root chunk. (The value *purpose* can be either Checkpoint/Log, DBSpace, Chunk, or Mirror Chunk.)

Action

Reduce the ONCONFIG parameter for the resource cited; bring the database server up and free some space in the primary root chunk. Then reattempt the same operation.

Unable to start SQL engine.

Cause

The database server encountered an out-of-memory condition.

Action

No action is necessary.

Unable to open tblspace *nn*, iserrno = *nn*.

Cause

The database server cannot open the specified tblspace. (The value *nn* is the hexadecimal representation of the tblspace number.)

Action

See the ISAM error message number *nn*, which should explain why the tblspace cannot be accessed. The error message appears in *IBM® Informix® Error Messages*.

The value of pagesize *pagesize* specified in the config file is not a valid pagesize. Use 2048, 4096 or 8192 as the value for PAGESIZE in the onconfig file and restart the server.

Cause

This message displays upon disk initialization. The value of PAGESIZE that was specified in the ONCONFIG file is not a valid value.

Action

Restart the database server with a valid PAGESIZE value.

Violations table is not started for the target table.

Cause

If you issue a STOP VIOLATIONS TABLE statement for which no violations table is started, you receive this message.

Action

To recover from this error, you must start a violations table for the target table.

Violations table reversion test completed successfully.

Cause

This message is recorded in the **logmessage** table in the **sysmaster** database when the **revtestviolations.sh** script has completed successfully (no open violations tables were found).

Action

No action is necessary.

Violations table reversion test failed.

Cause

When the database server finds an open violations table, it reports errors 16992 and 16993 in the **logmessage** table in the **sysmaster** database and aborts the reversion process.

Action

When this message appears, you must issue the STOP VIOLATIONS TABLE FOR *table_name* command for each open violations table. After you close all open violations tables, you can restart the reversion process.

Violations table reversion test start.

Cause

This message is recorded in the **logmessage** table in the **sysmaster** database when the **revtestviolations.sh** script is executed.

Action

No action is necessary.

Violations tables still exist.

Cause

This message is recorded in the **logmessage** table in the **sysmaster** database when an open violations table is found.

Action

When this message appears, you must issue the STOP VIOLATIONS TABLE FOR *table_name* command for each open violations table. After you close all open violations tables, you can restart the reversion process.

Virtual processor limit exceeded.

Cause

You configured the database server with more than the maximum number of virtual processors allowed (1000).

Action

Modify the value of the VPCLASS configuration parameter, the NETTYPE configuration parameter, or both.

Related reference:
NETTYPE configuration parameter
VPCLASS configuration parameter

VPCLASS *classname* name is too long. Maximum length is *maxlength*.

Cause

This message indicates an internal error.

Action

Contact Technical Support.

VPCLASS *classname* duplicate class name.

Cause

This message indicates an internal error.

Action

Contact Technical Support.

VPCLASS *classname* Not enough physical procs for affinity.

Cause

The physical processors in the affinity specification for the VP class *classname* do not exist or are offline.

Action

Make sure the named processors are online. Correct the affinity specification for the named VP class. Restart the database server.

Messages: W-X-Y-Z

WARNING: aio_wait: errno = *nn.*

Cause

While the database server was waiting for an I/O request to complete, it generated error number *nn* on an operation that it was attempting to execute.

Action

Contact Technical Support for assistance.

WARNING: Buffer pool size may cause database server to get into a locked state. Recommended minimum buffer pool size is *num* times maximum concurrent user threads.

Cause

There are not enough buffers in the buffer pool. The database server could use all available buffers and cause a deadlock to occur.

Action

Change the **buffers** field in the BUFFERPOOL parameter in the ONCONFIG file to the number that this message recommends. For more information on the BUFFERPOOL parameter, see BUFFERPOOL configuration parameter..

warning: Chunk time stamps are invalid.

Cause

A sanity check is performed on chunks when they are first opened at system initialization. The chunk specified did not pass the check and will be brought offline.

Action

Restore the chunk from a dbspace backup or its mirror.

Warning: *name_old* is a deprecated onconfig parameter. Use *name_new* instead. See the release notes and the Informix® Administrator's Reference for more information.

Cause

A deprecated ONCONFIG parameter was used. This message displays the first time that you use a deprecated parameter. The shorter form of the message displays thereafter.

Action

Use the suggested alternative ONCONFIG parameter.

Warning: *name_old* is a deprecated onconfig parameter. Use *name_new* instead.

Cause

A deprecated ONCONFIG parameter was used.

Action

Use the suggested alternative ONCONFIG parameter.

Warning: Unable to allocate requested big buffer of size *nn*.

Cause

The internal memory allocation for a big buffer failed.

Action

Increase either virtual memory size (SHMVIRTSIZE), the size of the added segments (SHMADD), or your total shared-memory size (SHMTOTAL).

You are turning off smart large object logging.

Cause

These changes will become the new sbspace default values. Changes have been made to the sbspace. The onspaces utility will read and update 100 smart large objects at a time and commit each block of 100 smart large objects as a single transaction. This utility might take a long time to complete.

Action

This informational message occurs when you issue the following command:

```
onspaces -ch sbspace -Df "LOGGING=OFF" -y
```

For more information, see onspaces -ch: Change sbspace default specifications.

Messages: Symbols

- HH:MM:SS Informix database server Version R.VV.PPPPP Software Serial Number RDS#XYYYYYY.
- argument: invalid argument.
- function_name: cannot allocate memory.

HH:MM:SS Informix® database server Version *R.VV.PPPPP* Software Serial Number RDS#*XYYYYYY*.

Cause

This message indicate the start-up of the database server, after the initialization of shared memory.

Action

No action is required.

argument: invalid argument.

Cause

This internal error indicates that an invalid argument was passed to an internal routine.

Action

Contact Technical Support.

function_name: cannot allocate memory.

Cause

The database server cannot allocate memory from internal shared-memory pool.

Action

Increase either virtual-memory size (SHMVIRTSIZE), the size of the added segments (SHMADD), or your total shared-memory size (SHMTOTAL).

Conversion and reversion error messages

If conversion or reversion is not successful, error messages are stored in the online.log file to help you identify what failed and what actions to take to fix the problem.

- Cannot revert new fragment expression for index index, tabid id.
- Cannot revert new table fragment expression for table with id id.
- The conversion of the database name has failed.
- Database name is not revertible...
- Database name: Must drop trigger (id = id_number) before attempting reversion.
- The dummy updates failed while converting database name. This may imply data corruption in the database. If so, restore the original database with the tape backup. For more information, see output_file.
- Error in slow altering a system table.
- Internal server error.
- Must drop long identifiers in table name in database name
- Must drop new database (name) before attempting reversion. Iserrno error_number
- Must drop new user defined statistics in database name, iserrno number
- The pload database contains load/unload jobs referring to long table names, column names, or database names. These jobs will not work as expected until they are redefined.
- Reversion canceled.
- There is a semi-detached index in this table, which cannot be reverted.
- WARNING: Target server version must have a certified Storage Manager installed after conversion/reversion and before bringing up server.

Cannot revert new fragment expression for index *index*, tabid *id*.

Cause

The index fragmentation was defined in a version more recent than the one to which you are reverting.

Action

Drop the problem index-fragmentation scheme and retry reversion.

Cannot revert new table fragment expression for *table* with id *id*.

Cause

The fragmentation of this table was defined in a version more recent than the one to which you are reverting.

Action

Drop the problem table fragmentation scheme and retry reversion.

The conversion of the database *name* has failed.

Cause

Indicates that the conversion of the specified database has failed.

Action

Connect to the database. This action triggers conversion of the database. If it fails, the relevant error message appears. Contact Technical Support.

Database *name* is not revertible...

Cause

The database has failed one of the reversion checks and is not revertible.

Action

Take action to correct the error displayed as a separate message.

Database *name*: Must drop trigger (id = *id_number*) before attempting reversion.

Cause

The database contains a trigger that was created in a version more recent than the one to which you are converting.

Action

Drop the trigger with the specified trigger identification number and then attempt reversion.

The dummy updates failed while converting database *name*. This may imply data corruption in the database. If so, restore the original database with the tape backup. For more information, see *output_file*.

Cause

During conversion of a database from a version earlier than version 9.2, dummy update statements are run against the system tables in the database being converted. This message indicates failure in running one of these update statements.

Action

To retry the dummy updates, run the dummy update script for your old database server version. For instructions, refer to the *IBM® Informix® Migration Guide*.

If data corruption occurred, restore the original database with the tape backup. For more information, see the *IBM Informix Backup and Restore Guide*.

Error in slow altering a system table.

Cause

An internal error occurred while performing reversion.

Action

Contact Technical Support.

Internal server error.

Cause

An unexpected error occurred during database reversion.

Action

Contact Technical Support.

Must drop long identifiers in table *name* in database *name*

Cause

Identifiers greater than 18 bytes in length are not supported in the database server version to which you are reverting.

Action

Make sure that all long identifiers in the system are either dropped or renamed before you attempt reversion.

Must drop new database (*name*) before attempting reversion. Iserrno *error_number*

Cause

The system contains a database that was created in a more recent version of the database server.

Action

Drop the new database and attempt reversion.

Must drop new user defined statistics in database *name*, iserrno *number*

Cause

Some distributions in the **sysdistrib** system table use user-defined statistics. This feature is not supported in the version to which you are reverting.

Action

Ensure that no user-defined statistics are present or used in the system and then attempt reversion.

The pload database contains load/unload jobs referring to long table names, column names, or database names. These jobs will not work as expected until they are redefined.

Cause

Printed during **onpload** reversion testing if the **onpload** database contains references to long table names, column names, or database names. But the reversion will complete.

Action

Redefine the load and unload jobs in the **onpload** database that have references to long identifiers.

Reversion canceled.

Cause

The reversion process was canceled because of errors encountered.

Action

Correct the cause of the errors, and restart reversion.

There is a semi-detached index in this table, which cannot be reverted.

Cause

A semi-detached index on this table cannot be reverted.

Action

To see the list of all semi-detached indexes, refer to the database server message log. Drop all semi-detached indexes, and retry reversion. You might need to recreate those indexes after reversion is complete.

WARNING: Target server version must have a certified Storage Manager installed after conversion/reversion and before bringing up server.

Cause

ON-Bar is being converted or reverted. The user must ensure that a storage manager, certified with the target database server version, is installed.

Action

None.

Conversion and Reversion Messages for Enterprise Replication

During conversion and reversion, specific messages are logged for Enterprise Replication by the concdr, revcdr, and revtestcdr scripts.

The scripts write the messages to standard output by default. The messages are stored in the concdr.out, revcdr.out, and revtestcdr.out files in $INFORMIXDIR/etc on UNIX or %INFORMIXDIR%\etc on Windows.

- CDR reversion test failed; for details look in $INFORMIXDIR/etc/revtestcdr.out.
- Enterprise Replication is not ready for conversion. The Control and TRG send queues should be empty for conversion/reversion to proceed.
- Enterprise Replication should be in a stopped state for conversion/reversion to proceed.
- ...'syscdr' reversion failed; for details look in $INFORMIXDIR/etc/revcdr.out.
- 'syscdr' conversion failed. For details, look in $INFORMIXDIR/etc/concdr.out.
- Syscdr should NOT contain new replicate sets for reversion to succeed.
- Syscdr should not contain replicates defined with the --floatieee option for reversion to succeed.

CDR reversion test failed; for details look in $INFORMIXDIR/etc/revtestcdr.out.

Cause

Enterprise Replication is not revertible.

Action

For more information, look at the messages in **revtestcdr.out**. Fix the reported problem before you attempt reversion.

Prints the output of the **revcdr.sh** or **revcdr.bat** script to standard output.

Enterprise Replication is not ready for conversion. The Control and TRG send queues should be empty for conversion/reversion to proceed.

Cause

There are elements in the control and Transaction Send Queue (also called TRG) send queues. The database server sends replicated data to the TRG queue before sending it to the target system.

Action

Wait for these queues to empty before you attempt either conversion or reversion. For more information, see the .

Prints this message to concdr.out during conversion or to revcdr.out during reversion.

Enterprise Replication should be in a stopped state for conversion/reversion to proceed.

Cause

Enterprise Replication should be in a stopped state for conversion or reversion to proceed.

Action

Stop Enterprise Replication. For more information, see the .

Prints this message to concdr.out during conversion or to revcdr.out during reversion.

...'syscdr' reversion failed; for details look in $INFORMIXDIR/etc/revcdr.out.

Cause

The reversion of the syscdr database failed.

Action

Find the cause of failure in the **revcdr.out** file, then fix the problem before you attempt reversion.

Prints the output of the **revcdr.sh** or **revcdr.bat** script to standard output.

'syscdr' conversion failed. For details, look in $INFORMIXDIR/etc/concdr.out.

Cause

Conversion of the **syscdr** database failed.

Action

If conversion fails, resolve the problem reported in **concdr.out**. Restore the **syscdr** database from backup and reattempt conversion.

Prints the output of the **concdr.sh** or **concdr.bat** script to standard output.

Syscdr should NOT contain new replicate sets for reversion to succeed.

Cause

The new replicate sets in the **syscdr** database are not compatible with older versions.

Action

Use the **cdr delete replicateset** command to delete the replicate sets. Then rerun the **revcdr.sh** or **revcdr.bat** script to reattempt reversion.

Prints this message to **revtestcdr.out**.

Syscdr should not contain replicates defined with the --floatieee option for reversion to succeed.

Cause

Replicates have been defined with the **--floatieee** option. You cannot revert these replicates to the older version.

Action

Use the **cdr delete replicateset** command to delete replicates defined with the **--floatieee** option, then reattempt reversion.

Prints this message to **revtestcdr.out**.

Dynamic Log Messages

Dynamically added log file *logid* to DBspace *dbspace_number*.

Cause

The next active log file contains records of an open transaction. Whenever the database server adds a log dynamically, it logs this message. Example: `Dynamically added log file 38 to DBspace 5`.

Action

Complete the transaction as soon as possible.

Log file *logid* added to DBspace *dbspace_number*.

Cause

Whenever the administrator adds a log file manually, the database server logs this message. Example: `Log file 97 added to Dbspace 2`.

Action

None required.

Log file number *logid* has been dropped from DBspace *dbspace_number*.

Cause

When you drop a newly-added log file, the database server logs this message. Example: `Log file number 204 has been dropped from DBspace 17`.

Action

None required.

Log file *logid* has been pre-dropped.

Cause

When you drop a used log file, it is marked as deleted (status **D**) and cannot be used again. After you perform a level-0 backup, the database server drops this log file and can reuse the space. Example: `Log file 12 has been pre-dropped.`

Action

To delete the log file, perform a level-0 backup of all storage spaces.

Pre-dropped log file number *logid* has been deleted from DBspace *dbspace_number*.

Cause

After a backup, the database server deletes a pre-dropped log file and logs this message. Example: `Pre-dropped log file number 12 has been deleted from DBspace 3.`

Action

None required.

ALERT: Because the oldest logical log (*logid*) contains records from an open transaction (*transaction_address*), the server is attempting to dynamically add a log file. But there is no space available. Please add a DBspace or chunk. Then complete the transaction as soon as possible.

Cause

If the database server is unable to dynamically add a log file because the instance is out of space, it logs this message.

Action

Add a dbspace or chunk to an existing dbspace. Then complete the transaction as soon as possible.

ALERT: The oldest logical log (*logid*) contains records from an open transaction (*transaction_address*). Logical logging will remain blocked until a log file is added. Add the log file with the onparams -a command, using the -i (insert) option, as in: onparams -a -d *dbspace* -s *size* -i. Then complete the transaction as soon as possible.

Cause

If the DYNAMIC_LOGS parameter is set to `1`, the database server prompts the administrator to add log files manually when they are needed.

Action

Use the **onparams -a** command with the **-i** option to add the log file after the current log file. Then complete the transaction as soon as possible.

Log file *logid* has been pre-dropped. It will be deleted from the log list and its space can be reused once you take level-0 archives of all BLOBspaces, Smart BLOBspaces and non-temporary DBspaces.

Cause

When you drop a used log file, it is marked as deleted (status **D**) and cannot be used again, and **onparams** prints this message.

Action

To delete the log file, perform a level-0 backup of all storage spaces.

Sbspace Metadata Messages

Allocated *number* pages to Metadata from chunk *number*.

Cause

The database server freed the specified number of pages from the reserved area and moved them to the metadata area of chunk *number*.

Action

None required.

Allocated *number* pages to Userdata from chunk *number*.

Cause

The database server freed the specified number of pages from the reserved area and moved them to the user-data area of chunk *number*.

Action

None required.

Freeing reserved space from chunk *number* to Metadata.

Cause

The metadata area in chunk *number* is full. The database server is trying to free space from the reserved area to the metadata area.

Action

None required.

Freeing reserved space from chunk *number* to Userdata.

Cause

The user-data area in chunk *number* is full. The database server is trying to free space from the reserved area to the user-data area.

Action

None required.

Truncate Table Messages

- The table cannot be truncated if it has an open cursor or dirty readers.
- The table cannot be truncated. It has at least one non-empty child table with referential constraints.

The table cannot be truncated if it has an open cursor or dirty readers.

Cause

You must have exclusive access to the table.

Action

Wait for dirty readers to complete or close all the open cursors and reissue the TRUNCATE TABLE command.

The table cannot be truncated. It has at least one non-empty child table with referential constraints.

Cause

You cannot truncate a table if it has child tables with referential constraints and at least one row.

Action

Empty the child tables before you truncate this table.

Limits in Informix

This topic lists the system-level and table-level parameter limits, the system defaults, and the access capabilities of Informix®.

- System-Level Parameter Limits
- Table-level parameter limits
- Informix System Defaults
- Access capabilities

System-Level Parameter Limits

Table 1. System-level parameter limits

System-Level Parameters	Maximum Capacity per Computer System
IBM® Informix systems per computer (Dependent on available system resources)	255
Maximum number of accessible remote sites	Machine specific
Maximum virtual shared memory segment (SHMVIRTSIZE)	2GB (32-bit platforms) or 4TB (64-bit platforms)
Maximum number of Informix shared memory segments	1024

System-Level Parameters	Maximum Capacity per Computer System
Maximum address space	UNIX: Machine specific Windows: 2.7 GB if 4-gigabyte tuning is enabled: • All Windows versions later than Windows 2003 • Windows 2003 and earlier versions if the boot.ini file contains the /3GB switch 1.7 GB for Windows 2003 and earlier versions if the boot.ini file does not contain the /3GB switch

Table-level parameter limits

Table 2. Table-level parameter limits

Table-level parameters (2K page size unless otherwise stated)	Maximum capacity per table
Data rows per page	255
Data rows per fragment	4,277,659,295
Data pages per fragment	16,775,134
Data bytes per fragment (excludes smart large objects (BLOB, CLOB) and simple large objects (BYTE, TEXT) created in blobspaces	2K page size = 33,818,670,144 4K page size = 68,174,144,576 8K page size = 136,885,093,440 12K page size = 205,596,042,304 16K page size = 274,306,991,168
Binary large object BLOB/CLOB pages	4 TB
Binary large objects TEXT/BYTE bytes	4 TB
Row length	32,767
Number of columns	32K
Maximum number of pages per index fragment	2,147,483,647
Key parts per index	16
Columns per functional index	102 (for C UDRs) 341 (for SPL or Java™ UDRs)
Maximum bytes per index key:	2K page size = 387 4K page size = 796 8K page size = 1615 12K page size = 2435 16K page size = 3254
Maximum size of an SQL statement	Limited only by available memory

Informix System Defaults

Table 3. System defaults

Database characteristic	Informix system default
Table lock mode	Page
Initial extent size	8 pages
Next extent size	8 pages
Read-only isolation level (with database transactions)	Committed Read
Read-only isolation level (ANSI-compliant database)	Repeatable Read

Access capabilities

Table 4. Access capabilities

Access Capabilities	Maximum Capacity per System
Maximum databases per Informix system	21 million
Maximum tables per Informix system	477 102 080
Maximum active users per Informix (minus the minimum number of system threads)	32K user threads
Maximum active users per database and table (also limited by the number of available locks, a tunable parameter)	32K user threads
Maximum number of open databases in a session	UNIX: 32 databases Windows: 8 databases
Maximum number of open tables per Informix system	Dynamic allocation
Maximum number of open tables per user and join	Dynamic allocation

Access Capabilities	Maximum Capacity per System
Maximum number of open transactions per instance	32 767
Maximum locks per Informix system and database	Dynamic allocation
Maximum number of page cleaners	128
Maximum number of partitions per dbspace	4K page size: 1048445, 2K page size: 1048314 (based on 4-bit bitmaps)
Maximum number of recursive synonym mappings	16
Maximum number of tables locked with LOCK TABLE per user	32
Maximum number of cursors per user	Machine specific
Maximum Enterprise Replication transaction size	4 TB
Maximum dbspace size	131 PB
Maximum sbspace size	131 PB
Maximum chunk size	4 TB
Maximum number of chunks	32 766
Maximum number of 2K pages per chunk	2 billion
Maximum number of open Simple Large Objects (applies only to TEXT and BYTE data types)	20
Maximum number of B-tree levels	20
Maximum amount of decision support memory	Machine specific
Utility support for large files	17 billion GB
Maximum number of storage spaces (dbspaces, blobspaces, sbspaces, or extspaces)	2047

DB-Access User's Guide

This publication describes how to use the DB-Access utility to access, modify, and retrieve information from IBM® Informix® database servers.

Important: Use DB-Access with the current version of the database server. If you use DB-Access with a database server from a different version, you might obtain inconsistent results, such as when you use a version that does not support long identifiers with a version that does.
This publication is written for the following users:

- Database users
- Database administrators
- Database-application programmers

This publication assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming

These topics are taken from *IBM Informix DB-Access User's Guide*.

- Getting started with DB-Access
  DB-Access provides a menu-driven interface for entering, running, and debugging Structured Query Language (SQL) statements and Stored Procedure Language (SPL) routines. You can also run DB-Access interactively from the command line.
- The full-screen menu interface
  The DB-Access full-screen menu interface guides you through running SQL statements.
- Appendixes

Getting started with DB-Access

DB-Access provides a menu-driven interface for entering, running, and debugging Structured Query Language (SQL) statements and Stored Procedure Language (SPL) routines. You can also run DB-Access interactively from the command line.

You use SQL and SPL commands to perform data-definition tasks, such as specifying the number and type of data columns in a table, and data-management tasks, such as storing, viewing, and changing table data.

You can use DB-Access for the following aspects of database processing:

- Running ad hoc queries that you run infrequently
- Connecting to one or more databases, transferring data between the database and external text files, and displaying information about a database
- Displaying system catalog tables and the Information Schema of databases
- Practicing the SQL and SPL statements and examples that are provided in the *IBM Informix Guide to SQL: Tutorial* or the *IBM Informix Database Design and Implementation Guide*
- Testing applications that you intend to store for use in a production environment
- Creating demonstration databases

Important: DB-Access is not intended as an application-development environment. You cannot branch conditionally or loop through SQL statements when you run them within DB-Access.
The DB-Access utility is included with the Informix® server and with the Informix Client Software Development Kit.

- Requirements for the Informix server DB-Access utility
  Before you start DB-Access, prepare the Informix server environment.
- Requirements for the Informix Client Software Development Kit DB-Access utility
  Before you start DB-Access on a client, set up the client environment.
- Demonstration databases
  You can practice using DB-Access with a demonstration database.
- Start DB-Access
  You start DB-Access by running the **dbaccess** command from the command line. You can choose whether to use the DB-Access menu interface or the command-line interface.

Requirements for the Informix server DB-Access utility

Before you start DB-Access, prepare the Informix® server environment.

Do the following tasks before you start the DB-Access utility that is included with the Informix server:

- Set environment variables
- If you require globalization, set up the Global Language Support (GLS) locale
- Start the database server

To secure DB-Access connections with IBM® Informix, you can use the Secure Sockets Layer (SSL) protocol.

- Environment variables for DB-Access
  As part of the installation and setup process, the system or database administrator sets certain environment variables that enable IBM Informix products to work within a particular operating-system environment.

Related information:
Secure sockets layer protocol

Environment variables for DB-Access

As part of the installation and setup process, the system or database administrator sets certain environment variables that enable IBM® Informix® products to work within a particular operating-system environment.

You must have $INFORMIXDIR/bin in your path if you use DB-Access on a UNIX operating system or %INFORMIXDIR%\bin in your path if you use DB-Access on a Windows operating system. Your operating system uses the path to locate the initialization script and the **dbaccess** executable file.

In a UNIX environment, the database server must have the appropriate terminal that is set up from among the terminals that are listed by the INFORMIXTERM environment variable.

DB-Access uses the terminal definitions in the terminfo directory unless the INFORMIXTERM environment variable is set to the termcap file. If DB-Access fails to initialize the menus that are based on the INFORMIXTERM setting, DB-Access tries to use the other setting. For example, if DB-Access fails to initialize the menus using the terminfo directory, DB-Access starts the menus using the termcap file.

You can set the following optional environment variables:

DBACCNOIGN
: Rolls back an incomplete transaction if you run the LOAD command in menu mode.

DBCENTURY
: Sets the appropriate expansion for DATE and DATETIME values that have only a two-digit year, such as 04/15/12.

DBDATE
: Specifies the user formats of DATE values.

DBEDIT
: Sets the default DB-Access text editor without changing the default text editor that is associated with the operating-system shell.
For more information about how DB-Access uses the text editor that you specify as default, see A system editor.

DBFLTMASK
: Sets the default floating-point values of data types FLOAT, SMALLFLOAT, and DECIMAL within a 14-character buffer.
The effect of this variable is limited to the DB-Access display size for numbers.

DELIMIDENT
: Causes the database server to interpret double quoted (“) text as identifiers rather than strings.

IFX_LONGID
: Determines whether a client application can handle long identifiers.
If you use the IFX_LONGID environment variable to support SQL identifiers with up to 128 bytes, some error, warning, or other messages of DB-Access might truncate database object names that include more than 18 bytes in their identifiers. You can avoid this truncation by not declaring names that have more than 18 bytes.

GL_DATETIME
: Defines the end-user formats for data values in DATETIME columns. In databases where GL_DATETIME has a nondefault setting, you must also set the USE_DTENV environment variable to `1` for user formats to be applied correctly in some load and unload operations.
Important:
The `%F` directive implies no default separator between the SECOND and FRACTION fields of DATETIME values. Defining no separator before the `%F` directive concatenates SECOND and FRACTION values, as in the following example, where GL_DATETIME has this setting:

```
%Y-%m-%d %H:%M:%S%F
```

Below is the end-user display for a DATETIME YEAR TO FRACTION(2) value on August 23, 2013, at exactly 53 seconds after 1:15 PM:

```
2013-08-23 13:15:5300
```

Here `5300` represents 53 seconds, concatenated with the FRACTION precision of 2. To display a separator between the integer and fractional parts of the seconds value, your GL_DATETIME setting must include a literal separator character immediately before the `%F` formatting directive.

Related information:
Environment variables

Requirements for the Informix Client Software Development Kit DB-Access utility

Before you start DB-Access on a client, set up the client environment.

The DB-Access utility on a client can directly access Informix® databases with which Client SDK has a client/server connection.

Do the following tasks before you start DB-Access utility that is included with Client SDK:

- Set the client/server connectivity information in the sqlhosts file.
  - If you renamed the file or moved it from the default location, set the INFORMIXSQLHOSTS environment variable to the full path name and file name of the file that contains the connectivity information.
  - You can use the **syncsqlhosts** utility to convert connection information from the Windows registry format to the sqlhosts file format.
- Set the INFORMIXDIR environment variable to the Client SDK installation directory.
- Set the INFORMIXSERVER environment variable for a default server name.

Related reference:
Start DB-Access
Related information:
The sqlhosts file and the SQLHOSTS registry key
INFORMIXDIR environment variable
INFORMIXSERVER environment variable
INFORMIXSQLHOSTS environment variable
The syncsqlhosts utility

Demonstration databases

You can practice using DB-Access with a demonstration database.

If you use IBM® Informix® demonstration databases, you can add, delete, or change the provided data and scripts. You can restore the database to its original condition.

You can configure the following demonstration databases:

- The **stores_demo** database illustrates a relational schema with tables about a fictitious wholesale sporting-goods distributor, as well as other tables that are used in examples. Tables containing electricity usage and geographical location data illustrate time series and spatial information. Many examples in IBM Informix manuals are based on the **stores_demo** database.
- The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines.

The scripts that you use to install the demonstration databases are in the $INFORMIXDIR/bin directory on UNIX and in the %INFORMIXDIR%\bin directory on Windows.

Some operating systems require that you have execute permissions to run SQL command files, read permissions to open these files or their contents in DB-Access, or write permissions to save modified or new files. Use the UNIX **chmod** command to enable execution of the SQL files that the initialization script installed.

The demonstration scripts are designed for the default locale. If you use a non-default locale, such as **en_us.utf8**, some features, such as the SET COLLATION statement, might not function correctly.

- Creating a demonstration database
  You create demonstration databases by running the **dbaccessdemo** command.
- dbaccessdemo command: Create demonstration databases
  Use the **dbaccessdemo** or **dbaccessdemo_ud** command to create the demonstration databases.

Related information:
The stores_demo Database
The superstores_demo database

Creating a demonstration database

You create demonstration databases by running the **dbaccessdemo** command.

When you create a demonstration database, the script confirms that you want to copy sample SQL command files. Command files that the demonstration database includes have a .sql extension and contain sample SQL statements that you can use.

To create a demonstration database:

1. Create a directory.

You must have UNIX read and execute permissions for each directory in the path name that you create.

2. Change directories to the new directory and run the **dbaccessdemo** or **dbaccessdemo_ud** command.
3. The initialization script displays a series of messages on the screen as the database is created. Press Y to copy the command files into the directory that you created. The demonstration database is created. You are the owner and database administrator (DBA) of that database.

If you want to discard changes that you made to your database or to the command files, rerun the **dbaccessdemo** or **dbaccessdemo_ud** command and press Y to replace the existing command files with the original versions.

dbaccessdemo command: Create demonstration databases

Use the **dbaccessdemo** or **dbaccessdemo_ud** command to create the demonstration databases.

Syntax for stores_demo

```
>>-dbaccessdemo--+-------+--+---------+-------------------------->
                 '- -log-'  '-dbname-'

>--+-------------------------+--+---------+--------------------><
   '- -dbspace--dbspace_name-'  '- -nots-'
```

Syntax for superstores_demo

```
>>-dbaccessdemo_ud--+-------+--+---------+----------------------->
                    '- -log-'  '-dbname-'

>--+-------------------------+---------------------------------><
   '- -dbspace--dbspace_name-'
```

-log
: Requests transaction logging for the demonstration database.

dbname
: Substitutes for the default database name. Must follow Identifier naming guidelines.

-dbspace
: Requests a particular dbspace location for the demonstration database.

dbspace_name
: Houses the demonstration database. If you do not specify a dbspace name, by default, the data for the database is put in the root dbspace. To create a dbspace, use the **onspaces** utility.

-nots
: Prevents the creation of the time series tables in the **stores_demo** database.

Examples

The following command creates a database that is named **stores_demo**:

```
dbaccessdemo
```

The following example creates an instance of the **stores_demo** database named **demo_db**:

```
dbaccessdemo demo_db
```

The following command initializes the **stores_demo** database and also initiates log transactions:

```
dbaccessdemo -log
```

The following command creates an instance of the **stores_demo** database named **demo_db** in **dbspace_2**:

```
dbaccessdemo demo_db -dbspace dbspace_2
```

The following command creates a database that is named **superstores_demo**:

```
dbaccessdemo_ud
```

Related information:
The stores_demo Database
The superstores_demo database
Identifier

Start DB-Access

You start DB-Access by running the **dbaccess** command from the command line. You can choose whether to use the DB-Access menu interface or the command-line interface.

You can start and use DB-Access in the following ways:

- Start DB-Access at the main menu.
- Start DB-Access from a specific menu or screen.
- Run a file that contains SQL statements without showing the DB-Access menus.
- Start and run DB-Access interactively at the command line, without the menu interface.

On Windows, you can set up the DB-Access program icon to run any of the **dbaccess** commands.

If the TERM, TERMCAP, or TERMINFO environment variables on UNIX do not enable DB-Access to recognize the type of terminal you use, the main menu does not show. Instead, a message similar to the following text is displayed:

```
Unknown terminal type.
```

If you use a window interface on a UNIX terminal, issue the **dbaccess** command from a nonscrolling console window.

If you use a Windows terminal to run DB-Access on a UNIX database server, the terminal-emulation window must emulate a terminal type that DB-Access can recognize, or the database server shows an unknown terminal-type message in the terminal-emulation window.

Tip: If your operating system cannot find **dbaccess**, include the full path before the program name, as follows:

```
$INFORMIXDIR/bin/dbaccess
```

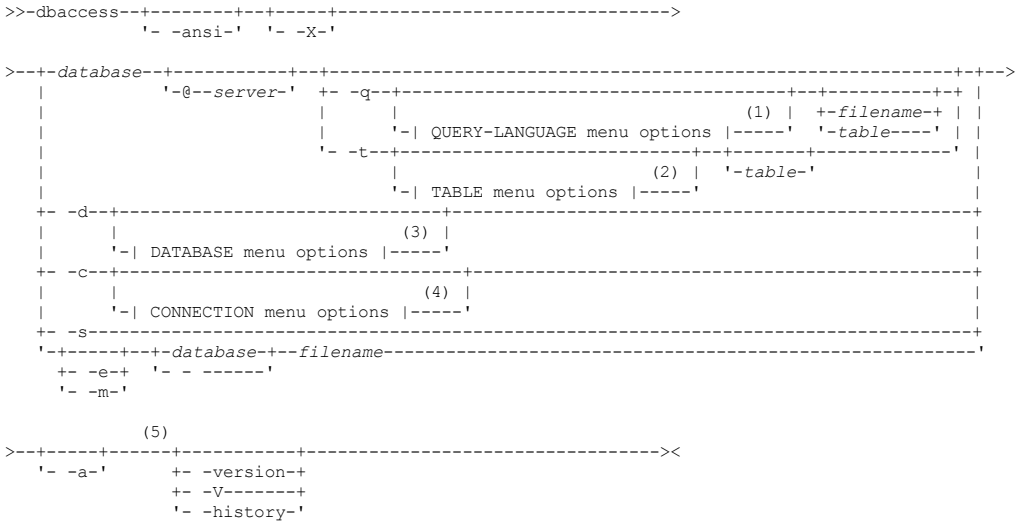
- dbaccess command: Start DB-Access
  Use the **dbaccess** command to start DB-Access. Include options to specify the database, command files, or to go to a specific menu screen.
- Run DB-Access in interactive mode without menus
  If you do not want to use the menus and do not have a prepared SQL file, use your keyboard or standard input device to enter SQL statements from the command line.

Related concepts:
Requirements for the Informix Client Software Development Kit DB-Access utility
Related information:
Environment variables

dbaccess command: Start DB-Access

Use the **dbaccess** command to start DB-Access. Include options to specify the database, command files, or to go to a specific menu screen.

Syntax



Notes:

1. See QUERY-LANGUAGE menu options.
2. See TABLE menu options.
3. See DATABASE menu options.
4. See CONNECTION menu options.
5. Cannot be combined with any other option.

The **dbaccess** command without options starts the main menu with no database selected and no options activated. You select submenus from the main menu.

-ansi
: Causes DB-Access to generate a warning whenever it encounters IBM® Informix® extension to ANSI-compliant syntax. For more information, see Example: Check for ANSI compliance.

-a
: Stops a process directly after the first error is encountered. Stopping a process from continuing after the first error can ensure greater database consistency.

-c
: Starts with the CONNECTION menu as the top-level menu.

-d

Starts with the DATABASE menu as the top-level menu.

-e
Echoes each line from a command file designated by *filename*.

-history
Can be abbreviated to **-his**. Displays the previous commands that you ran from the **dbaccess** command line during the current session. Commands that you ran from the menu interfaces are not shown. Each command is numbered. In the interactive mode without menus, you can rerun previous commands with the following command:

```
run number
```

The *number* is the number of the command to run. For an example, see Run DB-Access in interactive mode without menus.

-m
Displays all error messages generated by multiple levels of the server that pertain to an SQL statement in command files.

-q
Starts at the query-language menu (SQL-menu) as the top-level menu.

-s
Connects you to the main DB-Access menu and displays information about the current session.
This information includes database server name, database server type, the host computer, server capabilities, and other settings.

-t
Starts at the TABLE menu as the top-level menu.

-V
Displays the version number and serial number for DB-Access without launching the application. You cannot use any other options with **-V**.

-version
Displays the version number and build information for DB-Access, including the GLS library version, without launching the application. You cannot use any other options with **-version**.

-X
Activates the hexadecimal format for LOAD and UNLOAD statements.

database
Name of the database that you want DB-Access to connect to at the startup of your current session. A hyphen (-) indicates that the database is specified in a DATABASE statement in a command file.

filename
Names a command file to load with the SQL menu.

server
Name of the database server.

table
Specifies a table in the database.

If you exit from a submenu or option that you specified from the command line, you will exit directly to the operating-system command line.

- CONNECTION menu options
  The CONNECTION menu options for the **dbaccess** command represent short cut keys for the CONNECTION menu.
- DATABASE menu options
  The DATABASE menu options for the **dbaccess** command represent short cut keys for the DATABASE menu.
- QUERY-LANGUAGE menu options
  The QUERY-LANGUAGE menu options for the **dbaccess** command represent short cut keys for the QUERY-LANGUAGE menu.
- TABLE menu options
  The TABLE menu options for the **dbaccess** command represent short cut keys for the TABLE menu.
- Example: Start DB-Access for a database
  This example shows how to start DB-Access and specify a database to which to connect.
- Example: Run a command file
  This example shows how to start DB-Access and run a command file that contains SQL statements.
- Example: View the Information Schema
  This example shows how to start DB-Access and view the Information Schema for the specified database.
- Example: Check for ANSI compliance
  This example shows how to start DB-Access and check whether a database is ANSI-compliant.
- Example: Show nonprintable characters in hexadecimal
  This example starts DB-Access and activates the hexadecimal load and unload format (XLUF) so that the LOAD and UNLOAD SQL statements can format nonprintable ASCII signs in hexadecimal format.

CONNECTION menu options

The CONNECTION menu options for the **dbaccess** command represent short cut keys for the CONNECTION menu.

-cc
Chooses the Connect option on the CONNECTION menu.

-cd
Chooses the Disconnect option on the CONNECTION menu.

DATABASE menu options

The DATABASE menu options for the **dbaccess** command represent short cut keys for the DATABASE menu.

-dc
Chooses the Create option on the DATABASE menu.

-dcl
Takes you to the LOG option on the CREATE DATABASE menu

-dd
Chooses the Drop option on the DATABASE menu.

-di
Chooses the Info option on the DATABASE menu. With this option, you can add another letter as follows to go to the next menu level and view:

-dib
The dbspaces information for the current database

-din
The NLS information for the current database

-dip
Stored procedures in the current database

If you do not include a database name before any **-di** option, you must choose a current database from the SELECT DATABASE screen.

-dl
Chooses the CLose option on the DATABASE menu.

-ds
Chooses the Select option on the DATABASE menu.

QUERY-LANGUAGE menu options

The QUERY-LANGUAGE menu options for the **dbaccess** command represent short cut keys for the QUERY-LANGUAGE menu.

-qc
Chooses the Choose option on the SQL menu.

-qd
Chooses the Drop option on the SQL menu.

-qi
Chooses the Info option on the SQL menu. With this option, you can add another letter as shown in the following list (and specify a table) to go to the next menu level and view:

-qic
Columns in the table

-qif
Information about fragmentation strategy for the table

-qig
Information about triggers in the table

-qii
Indexes on the table

-qio
Table constraints

-qip
Access privileges on the table

-qir
Table-level references privilege on the table

-qis
Table status information

If you do not include a table name with the **-qi** option, you must choose one from the INFO FOR TABLE screen.

-qm
Chooses the Modify option on the SQL menu.

-qn
Chooses the New option on the SQL menu.

-qs
Chooses the Save option on the SQL menu.

-qu
Chooses the Use-editor option on the SQL menu.
If you do not include a database name before a **-q** option, you must choose a current database from the SELECT DATABASE screen.

When you select the Modify option on the QUERY-LANGUAGE menu, you must first select a command file to modify from the CHOOSE menu. The MODIFY screen is then displayed and shows the text.
Restriction: You cannot go directly to the Run or Output option on the SQL menu. Trying to do so results in an error message.

TABLE menu options

The TABLE menu options for the **dbaccess** command represent short cut keys for the TABLE menu.

-ta
Chooses the Alter option on the TABLE menu.

-tc
Chooses the Create option on the TABLE menu.

-td
Chooses the Drop option on the TABLE menu.

-ti
: Chooses the Info option on the TABLE menu. With this option, you can add another letter as shown in the following list (and specify a table) to go to the next menu level and view:

  -tic
  : Columns in the table
  -tif
  : Information about fragmentation strategy for the table
  -tig
  : Information about triggers in the table
  -tii
  : Indexes on the table
  -tio
  : Table constraints
  -tip
  : Access privileges on the table
  -tir
  : Table-level references privilege on the table
  -tis
  : Table status information

  If you do not include a table name with the **-ti** option, you must choose one from the INFO FOR TABLE screen.

  If you do not include a database name before a **-t** option, you must choose a current database from the SELECT DATABASE screen.

Example: Start DB-Access for a database

This example shows how to start DB-Access and specify a database to which to connect.

Assume that the database server that you have online contains a database named **mystores**. To make the **mystores** database the current database, start DB-Access with the following command:

```
dbaccess mystores
```

You can specify a database on a database server that is not online. For example, either of the following commands selects the **newstores** database on the **xyz** database server:

```
dbaccess newstores@xyz
dbaccess //xyz/newstores
```

When DB-Access starts, the database and database server name that you specify are displayed on the dashed line, as the following figure shows.
Figure 1. The DB-Access main menu with database and database server name

```
DB-Access:   Query-language  Connection  Database  Table  Session  Exit


---------------- newstores@xyz ---------------------Press CTRL-W for Help ---
```

Example: Run a command file

This example shows how to start DB-Access and run a command file that contains SQL statements.

The following sample command runs the SQL statements in a file named sel_stock.sql on the **mystores** database:

```
dbaccess mystores sel_stock
```

The following sample command runs the SQL statements in the sel_all.sql file on the database that file specifies:

```
dbaccess - sel_all.sql
```

Some operating systems require that you have execute permissions to run SQL command files, read permissions to open these files or their contents in DB-Access, or write permissions to save modified or new files.

Use the UNIX **chmod** command to enable execution of the SQL files that the initialization script installed.

Example: View the Information Schema

This example shows how to start DB-Access and view the Information Schema for the specified database.

The xpg4_is.sql file in the $INFORMIXDIR/etc directory creates the Information Schema and installs the views for a specified database. The following command creates the Information Schema for database **mystores**:

```
dbaccess mystores $INFORMIXDIR/etc/xpg4_is.sql
```

The Information Schema adds to the database four information-only views that conform to X/Open XPG4 with IBM® Informix® extensions. After you run xpg4_is.sql, use DB-Access to retrieve information about the tables and columns that you have access to in the specified database.
Tip: Do not install XPG4-compliant views on an ANSI database, because the format of XPG4-compliant views differs considerably from the format of the ANSI-compliant Information Schema views that are defined by the SQL standards committee.

Related information:
Information Schema

Example: Check for ANSI compliance

This example shows how to start DB-Access and check whether a database is ANSI-compliant.

To check your SQL statements for compliance with ANSI standards, include the **-ansi** option or set the DBANSIWARN environment variable. Use the **-ansi** option with other **dbaccess** options such as **-dc** (to create a database), **-tc** or **-ta** (to create or alter a table), or **-qc *filename*** (to choose a command file). The following command checks for ANSI compliance while DB-Access creates the database **research**:

```
dbaccess -ansi -dc research
```

You do not need to specify the **-ansi** option on the command line if the DBANSIWARN environment variable is set.

DB-Access displays the SQLSTATE value with the warning under the following circumstances:

- You include the **-ansi** option or set the DBANSIWARN environment variable.
- You access or create an ANSI database.
- You run DB-Access in command-line mode or specify a .sql input file.
- Running an SQL statement generates a warning rather than an error.

Related information:
DECIMAL(p) Floating Point
GET DIAGNOSTICS statement
DBANSIWARN environment variable
ANSI-Compliant Databases

Example: Show nonprintable characters in hexadecimal

This example starts DB-Access and activates the hexadecimal load and unload format (XLUF) so that the LOAD and UNLOAD SQL statements can format nonprintable ASCII signs in hexadecimal format.

The following command activates the XLUF format for the **mystores** database:

```
dbaccess -X mystores
```

A .unl file that the UNLOAD statement produces contains the hexadecimal format changes.

Related information:
The LOAD and UNLOAD statements

Run DB-Access in interactive mode without menus

If you do not want to use the menus and do not have a prepared SQL file, use your keyboard or standard input device to enter SQL statements from the command line.

When you start DB-Access without a menu argument and with a hyphen as the final argument, DB-Access processes commands from the standard input device (on UNIX) or the keyboard (on Windows). DB-Access reads what you type until you indicate that the input is completed. Then DB-Access processes your input and writes the results to the standard output device (on UNIX), or the command window (on Windows).

DB-Access reads and runs SQL statements from the terminal keyboard interactively. While DB-Access runs interactively, the greater than (>) prompt marks the line where you type your next SQL statement.

When you type a semicolon (;) to end a single SQL statement, DB-Access processes that statement. When you press CTRL-D to end the interactive session, DB-Access stops running. The following example shows user input and results in an interactive session:

```
dbaccess - -
>database stores_demo;

Database selected.

>select count(*) from systables;

(count(*))

     21

1 row(s) retrieved.

>^D

dbaccess - -
>database stores_demo;

Database selected.

>select count(*) from systables;

(count(*))
```

```
     21

1 row(s) retrieved.

>^D
```

Batch command input (UNIX)

You can use an inline shell script to supply one or more SQL statements. For example, you can use the UNIX C, Bourne, or Korn shell with inline standard input files:

```
dbaccess mystores- <<EOT!
select avg(customer_num) from customer
where fname matches '[A-G]*';
EOT!
```

You can use a pipe to supply SQL statements, as in this UNIX example:

```
echo 'select count(*) from systables' | dbaccess mystores
```

DB-Access interprets any line that begins with an exclamation mark (!) as a shell command. You can mix shell escape lines with SQL statements and put them in SQL statements, as follows:

```
dbaccess mystores -
>select
!echo hello
>hello
count(*) from systables;
>
(count(*))

     21

1 row(s) retrieved.
>
```

View and rerun DB-Access commands

You can view and rerun DB-Access commands that you ran from the command line during the current session.

To view the previous commands, run the **dbaccess -history** command. In the following example, the command history shows three previous commands:

```
dbaccess -history - -
1> create database sales_demo;

Database created.

2> create table customer (
2>         customer_code integer,
2>         customer_name char(31),
2>         company_name char(20));

Table created.

3> insert into customer values (102, "Carole Sadler", "Sports Spot");

1 row(s) inserted.

4> history;

1   create database sales_demo
2   create table customer (
           customer_code integer,
           customer_name char(31),
           company_name char(20))
3   insert into customer values (102, "Carole Sadler", "Sports Spot")
```

You can rerun the third command by running the **run 3** command:

```
4> run 3;

1 row(s) inserted.
```

- Connect to a database environment in interactive mode
  You can use the CONNECT . . . USER syntax in SQL statements that you issue in interactive mode. However, DB-Access does not support the USER clause of the CONNECT statement when you connect to a default database server.

Connect to a database environment in interactive mode

You can use the CONNECT . . . USER syntax in SQL statements that you issue in interactive mode. However, DB-Access does not support the USER clause of the CONNECT statement when you connect to a default database server.

When you include the USER '*user identifier*' clause in a CONNECT statement in interactive mode, DB-Access prompts you to enter a password.

The following two command examples show how to connect to a database server in interactive mode. The first example uses the CONNECT statement without specifying a user identifier.

```
dbaccess -nohistory- -

> connect to '@starfish';

Connected.
```

If you include the USER clause in a CONNECT statement, as the second example shows, DB-Access uses echo suppression to prompt you for a password:

```
> connect to '@starfish' user 'marae';

ENTER PASSWORD:

Connected.
```

Restriction: For security reasons, do not enter the password on the screen where it can be seen. Also, do not include the USING *password* clause in a CONNECT statement when you use DB-Access interactively. If you are in interactive mode and attempt to enter a password before the prompt, an error message is displayed.
You can run the USER clause of a CONNECT statement in a DB-Access file that includes the USER clause. The following example uses a command file that contains a CONNECT statement with a USING clause to connect to a database server:

```
dbaccess - connfile.sql
```

Important: An SQL command file that contains the following statement is protected from access by anyone other than the **user_id** that the USER clause identifies:

```
CONNECT TO '@dbserver' USER 'user_id' USING password
```

For UNIX, the following example uses a shell file to connect to a database server. DB-Access prompts you for a password.

```
dbaccess - - <<\!
connect to '@starfish' user 'marae';
!

ENTER PASSWORD:
```

Here the delimiting quotation marks preserve letter case in the database server name and in the authorization identifier of the user.

The full-screen menu interface

The DB-Access full-screen menu interface guides you through running SQL statements.

The DB-Access user interface combines the following features:

- A hierarchy of menus
- Screens that prompt you for brief responses and choices from selection lists
- Contextual HELP screens
- The interactive Schema Editor that helps you structure tables
- An SQL programmer environment, which includes the following features:
  - The built-in SQL editor where you enter and modify SQL and SPL statements
  - An option to use another editor of your choice
  - The database server syntax checker and runtime debugger
  - Storage, retrieval, and execution of SQL and SPL routines
- A choice of output for database queries and reports

- The Query-language option
  Use the Query-language option to enter, modify, save, retrieve, and run SQL statements. DB-Access retains the statements, if any, in the editor. These statements are called the *current* statements.
- The Database option
  Use the Database option to work with databases and transactions.
- The Table option
  Use the Table option to work with tables.
- The Connection and Session options
  Use the Connection option if you want to connect to a specific database server and database or explicitly disconnect from the current database environment. Use the Session option to display information about the current DB-Access session.

The Query-language option

Use the Query-language option to enter, modify, save, retrieve, and run SQL statements. DB-Access retains the statements, if any, in the editor. These statements are called the *current* statements.

Use the Query-language option to:

- Learn SQL and SPL.
  For example, use the Query-language option to practice the examples in the *IBM® Informix® Guide to SQL: Tutorial*.

- Create and alter table structures as an alternative to the DB-Access Schema Editor.
- Select, display, add, update, and delete data.

The SQL menu has the following options.

Option	Purpose
New	Clears current statements and positions cursor in SQL editor.

Option	Purpose
Run	Runs current SQL statements. A message is displayed or the data that is retrieved by a query is displayed with the number of rows retrieved.
Modify	Allows you to modify current SQL statements in SQL editor.
Use-editor	Starts a system editor so that you can modify current statements or create new statements. Use-editor is interchangeable with New and Modify.
Output	Redirects Run-option output to a file, printer, or system pipe.
Choose	Lists SQL command files so that you can choose a file to run or modify.
Save	Saves current SQL statements in a file for later use.
Info	Shows table information, such as columns, indexes, privileges, constraints, triggers, status, and fragmentation strategy.
Drop	Deletes a specified SQL command file.
Exit	Returns to main menu.

- SQL editor
  When you choose the New of Modify option, you see the SQL editor. You can type as many lines of text as you need. You are not limited by the size of the screen, although you might be limited by the memory constraints of your system. If you do not use the Save option to save your typed statements, they are deleted when you select an option that clears the SQL editor (such as New or Choose).
- A system editor
  When you want to enter or modify a long SQL statement or series of statements, you might prefer the flexibility and familiarity of a system editor to the SQL editor. Select the Use-editor option from the SQL menu to use the system editor.
- Statements that the Run option supports
  After you exit the editor screen, the SQL menu reopens with the Run option highlighted and the statement text is displayed in the bottom of the screen. You can run most SQL statements with the Run option.
- Redirect query results
  The output from a SELECT statement is normally displayed on the screen. You can use the Output option on the SQL menu to route query results to the printer, store them in a system file, or pipe them to a program. The Output option has the same result as the OUTPUT statement of SQL.
- Choose an existing SQL statement
  When you save SQL statements in a command file, you can retrieve the command file and run or edit the SQL statements at any time.
- Save the current SQL statement
  You can save SQL statements in a file for later use, such as to run the statements from the command line or retrieve the saved statements with the Choose option on the SQL menu.
- Support for SPL Routines
  You can create and run routines that are written in SPL from the SQL menu.
- What happens when errors occur
  If you make any syntax or typing mistakes in an SQL statement, DB-Access does not process the statement. Instead, it continues to display the text of the statement with a message that describes the error.

SQL editor

When you choose the New of Modify option, you see the SQL editor. You can type as many lines of text as you need. You are not limited by the size of the screen, although you might be limited by the memory constraints of your system. If you do not use the Save option to save your typed statements, they are deleted when you select an option that clears the SQL editor (such as New or Choose).

The SQL editor does not display more than 80 characters on a line and does not wrap lines.

- If you choose an existing command file in which the characters in a line extend beyond the 80th column, DB-Access displays a percent sign (%) in the 80th column to indicate an overflow. You cannot see all the characters beyond the percent sign, but the statement runs correctly.
- If you type characters in a new command file so that a line extends beyond the 80th column, DB-Access overwrites all the characters in the 80th column. You cannot see the overflow, and the statement does *not* run correctly.

To make the full text show on the screen, press Enter at a logical place in the first 80 characters of each line.

If you must type a quoted character string that exceeds 80 characters, such as an insert into a long CHAR column, use a system editor instead of the SQL editor.

If you want to include comments in the text:

- Use double minus signs for ANSI-compliant databases.
- Preface each comment line with a double minus sign (--) comment indicator. The comment indicator spans the entire line.
- Use braces ({ }) for databases that are not ANSI-compliant. Enclose the entire comment indicator between the braces.

A system editor

When you want to enter or modify a long SQL statement or series of statements, you might prefer the flexibility and familiarity of a system editor to the SQL editor. Select the Use-editor option from the SQL menu to use the system editor.

If you have not set the DBEDIT environment variable, you must select a text editor to use for the session. If you select Use-editor, DB-Access prompts you to accept or override the default system editor once each session.

The default editor that DB-Access displays depends on the preference that you establish for your operating system:

- Common UNIX system editors are **vi** and **ex**.
- If you use a text editor as the system default, you must save the .sql files as text.

Press RETURN to select the default editor you named after the USE-EDITOR prompt. To use a different editor, type the name of that editor and press RETURN.

Statements that the Run option supports

After you exit the editor screen, the SQL menu reopens with the Run option highlighted and the statement text is displayed in the bottom of the screen. You can run most SQL statements with the Run option.

To run statements that are not listed, use the SQL menu options New (or Use-editor) and Save to enter and save them, and then run the saved file from the command line.

The following is a list of SQL statements that you can run with the Run option.

- ALLOCATE COLLECTION
- ALLOCATE DESCRIPTOR
- ALLOCATE ROW
- ALTER ACCESS_METHOD
- ALTER FRAGMENT
- ALTER FUNCTION
- ALTER INDEX
- ALTER PROCEDURE
- ALTER ROUTINE
- ALTER SECURITY LABEL COMPONENT
- ALTER SEQUENCE
- ALTER TABLE
- BEGIN WORK
- CLOSE
- CLOSE DATABASE
- COMMIT WORK
- CONNECT
- CREATE ACCESS_METHOD
- CREATE AGGREGATE
- CREATE CAST
- CREATE DATABASE
- CREATE DISTINCT TYPE
- CREATE EXTERNAL TABLE
- CREATE FUNCTION
- CREATE FUNCTION FROM
- CREATE INDEX
- CREATE OPAQUE TYPE
- CREATE OPCLASS
- CREATE PROCEDURE
- CREATE ROLE
- CREATE ROUTINE FROM
- CREATE ROW TYPE
- CREATE SCHEMA
- CREATE SECURITY LABEL COMPONENT
- CREATE SECURITY LABEL
- CREATE SECURITY POLICY
- CREATE SEQUENCE
- CREATE SYNONYM
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- CREATE XADATASOURCE
- CREATE XADATASOURCE TYPE
- DATABASE
- DEALLOCATE COLLECTION
- DEALLOCATE DESCRIPTOR
- DEALLOCATE ROW
- DECLARE
- DELETE
- DESCRIBE
- DESCRIBE INPUT
- DISCONNECT
- DROP ACCESS METHOD
- DROP AGGREGATE
- DROP CAST
- DROP DATABASE
- DROP FUNCTION
- DROP INDEX
- DROP OPAQUE TYPE
- DROP OPCLASS
- DROP PROCEDURE
- DROP ROLE
- DROP ROW TYPE
- DROP SECURITY LABEL COMPONENT/POLICY/LABEL
- DROP SEQUENCE
- DROP SYNONYM
- DROP TABLE

- DROP TRIGGER
- DROP TYPE
- DROP VIEW
- DROP XADATASOURCE
- DROP XADATASOURCE TYPE
- EXECUTE
- EXECUTE FUNCTION
- EXECUTE IMMEDIATE
- EXECUTE PROCEDURE
- FETCH
- FLUSH
- FREE
- GET DESCRIPTOR
- GET DIAGNOSTICS
- GRANT
- GRANT DBSECADM
- GRANT DEFAULT ROLE
- GRANT EXEMPTION
- GRANT FRAGMENT
- GRANT SECURITY LABEL
- INFO
- INSERT
- LOAD
- LOCK TABLE
- MERGE
- OPEN
- OUTPUT
- PREPARE
- PUT
- RENAME COLUMN
- RENAME DATABASE
- RENAME INDEX
- RENAME SEQUENCE
- RENAME TABLE
- REVOKE
- REVOKE DBSECADM
- REVOKE DEFAULT ROLE
- REVOKE EXEMPTION
- REVOKE FRAGMENT
- REVOKE SECURITY LABEL
- ROLLBACK WORK
- SAVE EXTERNAL DIRECTIVES
- SELECT
- SET AUTOFREE
- SET COLLATION
- SET CONNECTION
- SET CONSTRAINTS
- SET DATASKIP
- SET DEBUG FILE TO
- SET DEFERRED PREPARE
- SET DESCRIPTOR
- SET ENCRYPTION PASSWORD
- SET ENVIRONMENT
- SET EXPLAIN
- SET ISOLATION
- SET LOCK MODE
- SET LOG
- SET OPTIMIZATION
- SET PDQPRIORITY
- SET ROLE
- SET SESSION AUTHORIZATION
- SET STATEMENT CACHE
- SET TRANSACTION
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE
- TRUNCATE
- UNLOAD
- UNLOCK TABLE
- UPDATE
- UPDATE STATISTICS
- WHENEVER

Redirect query results

The output from a SELECT statement is normally displayed on the screen. You can use the Output option on the SQL menu to route query results to the printer, store them in a system file, or pipe them to a program. The Output option has the same result as the OUTPUT statement of SQL.

The SELECT statement must be on the screen as the current statement. Select the Output option from the SQL menu, which displays the OUTPUT menu.

You have the following output options:

- Send your query results directly to a printer. DB-Access sends the results to your default printer and displays a message on the bottom of the screen that indicates how many rows were retrieved. The query results do not show on the screen. You can set the DBPRINT environment variable to specify a default printer.
- Write query results to a new file or append the results to an existing file. If you do not specify a path when DB-Access prompts you for a file name, the file is stored in the directory that you were in when you started DB-Access.
- Send query results to a pipe. Specify a target program, such as **more**, through which to pipe output. DB-Access sends the results to that pipe.
  On UNIX systems, you must have permission to run the target program.

  On Windows systems, the **cat** utility can serve as a target program through which to pipe output.

Choose an existing SQL statement

When you save SQL statements in a command file, you can retrieve the command file and run or edit the SQL statements at any time.

Select the Choose option on the SQL menu to display the CHOOSE screen with a list of the command files that you can access. These files have the extension .sql, although the extension is not shown. For example, the following figure lists the command files that are included in the demonstration database.
Figure 1. The CHOOSE screen listing current .sql files

```
CHOOSE >>
Choose a command file with the Arrow Keys, or enter a name, then press Return.

---------------- mystores@dbserver1 ----------- Press CTRL-W for Help -------

 alt_cat              c_state             d_trig              sel_ojoin1

c_calls              c_stock             d_view              sel_ojoin2

c_cat                c_stores            del_stock           sel_ojoin3

c_custom             c_table             ins_table           sel_ojoin4

c_index              c_trig              opt_disk            sel_order

c_items              c_type              sel_agg             sel_sub

c_manuf              c_view1             sel_all             sel_union

c_orders             c_view2             sel_group           upd_table

c_proc               d_proc              sel_join
```

If no current database exists, the list includes all the command files in the current directory and in any directories that the DBPATH environment variable specifies.
Important: This list includes only those file names that have the .sql extension. If you create an SQL file outside of DB-Access and save it without the .sql extension, the files does not show in the list of files to choose.
DB-Access can only recognize files that are stored in the directory from which you started DB-Access. If the Choose command results in an empty list, and you know that you have command files, exit DB-Access, change directories to the directory that contains your .sql files, and restart DB-Access.

Save the current SQL statement

You can save SQL statements in a file for later use, such as to run the statements from the command line or retrieve the saved statements with the Choose option on the SQL menu.

To save the current SQL statement or statements in a file, select the Save option on the SQL menu. Enter a name for the command file:

- Use 1 - 10 characters. Start with a letter, then use any combination of letters, numbers, and underscores (_). Press Enter to save the file.
- UNIX: File names are case-sensitive. The file orders is not the same as Orders or ORDERS.

DB-Access appends the extension .sql to the file name. For example, if you name your file cust1, DB-Access stores the file with the name **cust1.sql**. The CHOOSE screen still lists cust1, but the operating system identifies the same file as cust1.sql if you list the directory files from the command line.

Support for SPL Routines

You can create and run routines that are written in SPL from the SQL menu.

You can store the SPL routine in a separate command file and then call it from an application or run it as a stand-alone program. After you create the SPL routine, you can run it within DB-Access with the appropriate SQL statement. The following example details the steps.

To create and run an SQL routine

1. To create the text of the routine, type directly in the NEW screen or the Use-editor screen. Enter the SPL and SQL statements for your routine in the statement block of a CREATE PROCEDURE statement.
   Use the CREATE FUNCTION statement if the routine returns values.

2. Use the Run option to create the routine and register it in the **sysprocedures** system catalog table.

3. Use the NEW screen to enter an EXECUTE PROCEDURE statement that names the routine that you want to run.
   If you use IBM® Informix® and created your routine with the CREATE FUNCTION statement, enter an EXECUTE FUNCTION statement to run the function.
4. Use the Run option to run the routine and display the results.

For example, the c_proc.sql command file, which is supplied with the demonstration database, contains an SPL. Before you can run the routine, change the word *procedure* in the c_proc.sql file to *function* because the routine returns a value. Then use the Choose option and select **c_proc**.

First you must register the routine in the database. Select the Run option, as the following figure shows.
Figure 1. Displaying the text of an SPL routine on the SQL menu

```
SQL:   New  Run   Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit

Run the current SQL statements.

---------------------- mydata@mynewdb ------- Press CTRL-W for Help --------

create function read_address (lastname char(15))

        returning char(15), char(15), char(20), char(15),char(2), char(5);
        define p_fname, p_city char(15);
        define p_add char(20);
        define p_state char(2);
        define p_zip char(5);
        select fname,  address1, city, state, zipcode
                into p_fname,  p_add, p_city, p_state, p_zip
                from customer
                where lname = lastname;

        return p_fname, lastname, p_add, p_city, p_state, p_zip;

end procedure;
```

DB-Access displays a message to indicate that the database server created the routine. To run the routine, select New from the SQL menu and then enter the appropriate EXECUTE statement. In the following example, the user requests the address of a customer whose surname is Pauli:

```
EXECUTE PROCEDURE read_address ("Pauli")
```

After you enter the EXECUTE PROCEDURE or EXECUTE FUNCTION statement on the NEW screen, press Esc to return to the SQL menu. Select Run from the SQL menu to run the routine. The following figure shows the result of running the routine.
Figure 2. Result of running an SPL routine on the SQL menu

```
SQL:   New   Run   Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit

Run the current SQL statements.

---------------------- mydata@mynewdb ------- Press CTRL-W for Help --------

Ludwig
Pauli
213 Erstwild Court
Sunnyvale
CA
94086
```

Tip: SPL routines are stored in the system catalog tables in executable format. Use the Routines option on the DATABASE INFO menu to display a list of the routines in the current database or to display the text of a specified routine.

What happens when errors occur

If you make any syntax or typing mistakes in an SQL statement, DB-Access does not process the statement. Instead, it continues to display the text of the statement with a message that describes the error.

If an execution or runtime error occurs, DB-Access continues to process the statement and returns an error message. For example, if you try to create a table that was already created, the following message is displayed at the bottom of the screen:

```
310: Table (mavis.mystock) already exists in database.
```

If you try to run a statement that contains more than one SQL statement, you might not see an error message immediately. If, for example, the first statement is a SELECT statement that runs correctly and the next statement contains a typing error, the data that the first statement retrieved shows on the screen before the error message is displayed for the second statement.

When DB-Access detects an error, processing stops and the Modify option on the SQL menu is highlighted. Select one of the following methods to correct the statement:

- Press Enter to choose Modify, which returns you to the SQL editor.
- Select the Use-editor option to use the default editor of your choice.

The Database option

Use the Database option to work with databases and transactions.

Use the Database option to:

- Create a database or select a database.

The database that you work with is called the *current* database.

- Retrieve and display information about a database, such as available dbspaces and the text of routines.
- Delete an existing database or close the current database.
- Commit or rollback transactions.

You can access only databases that are on the current database server. To select a database server as current, you can specify a database server when you start DB-Access, you can use the Connection menu, or you can run a CONNECT statement from the SQL menu. If you do not explicitly select a database server, DB-Access uses the default database server that the $INFORMIXSERVER environment variable specifies as the current database.

If you select or create a database when another database is already open, DB-Access closes that database before it makes your selection the current or new database.

The DATABASE menu displays the following options.

Option	Purpose
Select	Makes a database the current database
Create	Builds a new database and makes that database the current database
Info	Displays information about the current database
Drop	Removes a database from the system. You cannot delete the current database.
cLose	Closes the current database
Exit	Exits the DATABASE menu and returns you to the main menu

- List of available databases
  When you choose the Select option, the SELECT DATABASE screen opens. The first database in the list of available databases is highlighted, accompanied by the names of database servers.
- Retrieve nondefault locale information
  Globalization affects the order in which lists are displayed in DB-Access. Globalization enables the display and appropriate ordering of non-English-language data. Informix® supports globalization with Global Language Support (GLS) locales. Earlier database server versions used Native Language Support (NLS) for this purpose.
- Close a database
  To close the current database, use the cLose option from the DATABASE menu.

List of available databases

When you choose the Select option, the SELECT DATABASE screen opens. The first database in the list of available databases is highlighted, accompanied by the names of database servers.

The list is organized alphabetically by database server and then by database for each database server. You can display a maximum of 512 database names on the SELECT DATABASE screen.

Important: In the SELECT DATABASE screen, the names of databases are limited to 18 characters. If a database name is longer than 18 characters, you see the first 17 characters of the name followed by a '+' sign. Enter a '+' sign to display the complete long name in **vi**. To exit from **vi**, press `ESC ZZ`.
The list of available databases that is displayed depends on two factors:

- The settings of certain environment variables.
  - If you use one database server, DB-Access displays the names of all databases on the current database server and in your DBPATH setting.
  - If you use multiple database servers, the ONCONFIG environment variable determines the current database server.
- The current connection. For example:
  - If no explicit connection exists, DB-Access displays the databases in the DBPATH setting.
  - If a current explicit connection exists, all databases in the DBPATH setting that pertain to the current database server are displayed.

Retrieve nondefault locale information

Globalization affects the order in which lists are displayed in DB-Access. Globalization enables the display and appropriate ordering of non-English-language data. Informix® supports globalization with Global Language Support (GLS) locales. Earlier database server versions used Native Language Support (NLS) for this purpose.

If the current database supports globalization, you can select the Nls option on the DATABASE INFO menu to display information about collating sequence and C CType (character classification type), as the following figure shows.
Figure 1. The DATABASE INFO menu with globalization information displayed

```
DATABASE INFO:  dBspace   Nls   Routine  Databases  Exit
Display NLS information for a database.

-------------- - database_demo ------------ Press CTRL-W for Help --------


fr_fr.8859-1 Collating Sequence
CType
```

DB-Access does not provide an option on the DATABASE INFO menu for displaying the GLS collating sequence and character classification type. To obtain information about the GLS locale that is enabled for your database server, enter the following query with the SQL editor:

```
SELECT tabname, site FROM systables
WHERE tabid = 90 OR tabid = 91
```

The row with tabid 90 stores the COLLATION category of the database locale. The row with tabid 91 stores the CTYPE category of the database locale. The following figure shows the result of the preceding query for the default US English locale.
Figure 2. Retrieving GLS information

```
SQL:   New  Run   Modify  Use-editor  Output  Choose  Save  Info  Drop  Exit
Run the current SQL statements.

------- mydata@mynewdb ------ Press CTRL-W for Help ---

tabname             GL_COLLATE
site                en_US.819

tabname             GL_CTYPE
site                en_US.819

2 row(s) retrieved
```

For further information about the COLLATION and CTYPE categories in a GLS locale file, see the *IBM® Informix GLS User's Guide*.

Close a database

To close the current database, use the cLose option from the DATABASE menu.

If you begin a transaction but do not commit it or roll it back, and then you try to close a database with transactions, the TRANSACTION menu opens. The TRANSACTION menu ensures that you either commit or roll back an active transaction before you close the current database.
Important: Select an option carefully. You might commit transactions that you do not want if you select Commit, and you do lose any new transactions if you select Rollback.
The TRANSACTION menu also opens whenever you attempt to open a new database or try to leave the DB-Access menu system without first terminating a transaction.

Important: If you begin a transaction in an ANSI-compliant database but do not issue a COMMIT statement or ROLLBACK statement, then try to close the database using a non-menu mode, DB-Access commits the transaction for you. If you do not want to commit the transaction, issue both a ROLLBACK statement and a CLOSE DATABASE statement from the command line.

The Table option

Use the Table option to work with tables.

Use the Table option to perform any of the following table management tasks without SQL programming:

- Create a new table
- Define fragmentation strategy for a new or existing table
- Alter, delete, or display information about an existing table

Use the TABLE menu options as the following table shows.

Option	Purpose
Create	Enables you to define the structure of a new table. The CREATE TABLE menu provides data type options for built-in data types. To define a column with one of the extended data types, such as smart large objects, user-defined (opaque) data types, or a collection data type, use the SQL menu to enter and run a CREATE TABLE statement. DB-Access can construct only a nonclustered, ascending B-tree column index. If you want hash or hybrid fragmentation, use the SQL menu to enter and run the CREATE TABLE or ALTER TABLE statement.
Alter	Enables you to alter the structure of an existing table, including columns, fragmentation, and constraints. You must have the Alter privilege to successfully alter a table. Note: If you use the UI to build the SQL statements to Alter a table without the Alter privilege, the table will be locked and the operation will fail. Further, the lock will be held until you exit the alter menu. To use the LOAD statement to insert data into a table, you must have both Insert and Select privileges for the table.
Info	Displays information about the structure of a table
Drop	Deletes a table from the database
Move	Moves a table from the current database to another database.
Exit	Returns to the DB-Access main menu

Both the CREATE TABLE and ALTER TABLE menus have the same options, which are described in the following table.

Option	Purpose
Add	Displays the Schema Editor, from which you can add a new column to the table
Modify	Displays the columns that you defined with the Add option so that you can modify the column structure before building the table
Drop	Drops an existing column from the table
Screen	Displays the next screen of column definitions in the Schema Editor
Table_options	Enables you to display and select storage spaces for a new table. Displays choices from which to set a fragmentation strategy for a new table. Enables you to set extent sizes and lock mode for a new table. Adds or deletes rowids for an existing fragmented table.
Constraints	Enables you to define primary-key, foreign-key, check, and unique constraints, and to set default column values

Option	Purpose
Exit	Builds, rebuilds, or discards the schema and structure that you specified with the other options, and then returns to the TABLE menu

Important: You must use the SPACEBAR to move between menu options, because the arrow keys control cursor movement in the Schema Editor.

- Display table information
  Use the Info option on the TABLE menu to display information about the structure of a table.

Display table information

Use the Info option on the TABLE menu to display information about the structure of a table.

Note the following items:

- If you are not the table owner, the table name is prefixed by the owner name, as in **june.clients**.
- If the list of tables does not fit on one screen, the last entry is an ellipsis (...). Use the arrow keys to highlight the ellipsis, and the next page of table names are displayed.
- If globalization is enabled, the list of table names is sorted according to the database collation rules defined when the database was created. Thus, different users using different collating sequences for DB-Access see the table names in the database listed in the same order.

To request information about tables on a different database server, use the format *database*@*server*:*table* or *database*@*server*:*owner*.*table* at the prompt. The following example requests information about the **customer** table that **dba** created in the **accounts** database on the database server **topend**:

```
INFO FOR TABLE >> accounts@topend:dba.customer
```

The INFO menu has the following options.

Option	Purpose
Columns	Lists data type by column name and indicates which columns can contain a null value
Indexes	Describes each index that is defined for a specified table
Privileges	Lists the users who have Select, Update, Insert, Delete, Index, or Alter privileges for the specified table
References	Lists the users who have the table-level References privilege for the specified table and the names of the columns they can reference
Status	Lists the table name, owner, row size, number of rows and columns, and creation date of the current table
cOnstraints	Displays the referential, primary, unique, and check constraints, and the default values for the columns in the specified table
triGgers	Displays header and body information for a specified trigger
Table	Redisplays the INFO FOR TABLE menu so that you can select a different table for examination
Fragments	Lists fragmented dbspaces assigned to the table and, for expression-based fragmentation, displays the expression that is assigned to each dbspace
Exit	Returns to the TABLE menu

Tip: From the CREATE TABLE menu, use Table-options to view extent and lock mode information, or issue a SELECT statement to list the table description in the **systables** system catalog table.

The Connection and Session options

Use the Connection option if you want to connect to a specific database server and database or explicitly disconnect from the current database environment. Use the Session option to display information about the current DB-Access session.

For the globalization considerations that apply to establishing a connection between a client application, such as DB-Access, and a database, see the *IBM Informix GLS User's Guide*. The database server examines the client locale information passed by the client, verifies the database locale, and determines the server-processing locale for transferring data between the client and the database.

You can use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures privacy and integrity of data that is transmitted over the network, for DB-Access connections with IBM® Informix®. For information about the SSL protocol, see Secure sockets layer protocol.

On Windows, if you specify a user identifier but no domain name for a connection to a machine that expects both a domain name and a user name (*domain\user*), DB-Access checks only the local machine and the primary domain for the user account. If you explicitly specify a domain name, that domain is used to search for the user account. The attempted connection fails with error -951 if no matching *domain\user* account is found on the local machine.

The CONNECTION menu displays the following options.

Option	Purpose
Connect	Connects to a database environment. To access a specific database, you must have permission.
Disconnect	Disconnects from the current database environment
Exit	Returns to the DB-Access main menu

When you use the Connect option, the SELECT DATABASE screen alphabetically lists all available databases on the specified database server. The database list on the SELECT DATABASE screen depends on the current connection. For example:

- If no current connection exists or the current connection is an implicit default connection, all the databases that are listed in the DBPATH environment variable setting are displayed.
- If a current explicit connection exists, all the databases in the DBPATH that pertain to the current server are displayed.

- Implicit closures
  DB-Access closes any open connections or databases when you connect to a new environment.

Implicit closures

DB-Access closes any open connections or databases when you connect to a new environment.

DB-Access closes any open connections or databases in the following situations:

- When you connect to a new database environment without explicitly disconnecting from the current one, DB-Access performs an implicit disconnect and the database closes.
- When you connect to a *database*@*server* and then close the database, the database server remains connected.
- When you connect to a database server, open a database, and then close the database, the database server remains connected.
- If you open a database and then try to connect to a database server, DB-Access performs an implicit disconnect and closes the database.
  Only one connection is allowed. You must disconnect from the database server associated with the open database or close the database before you can connect to another database server.

If DB-Access must close a database that still has outstanding transactions, it prompts you to commit or roll back those transactions.

Appendixes

- How to read online help for SQL statements
  Specific conventions are used to represent the syntax of SQL statements in DB-Access online help screens.
- Demonstration SQL
  Various command files that are available with DB-Access.

How to read online help for SQL statements

Specific conventions are used to represent the syntax of SQL statements in DB-Access online help screens.

You can request online help for SQL statements in either of the following ways:

- Highlight the New, Modify, or Use-editor options on the SQL menu and press CTRL-W.
- Press CTRL-W while you are on the NEW or MODIFY screens of the SQL menu.

The form of the syntax diagrams that shows when you request online Help for SQL statements in DB-Access is different from the syntax diagrams in the *IBM® Informix® Guide to SQL: Syntax*.

The conventions and rules governing SQL statement syntax in DB-Access online help screens are described in the following list.

ABC
: Any term in an SQL statement that is displayed in uppercase letters is a keyword. Type keywords exactly, disregarding case, as shown in the following example:

  ```
  CREATE SYNONYM synonym-name
  ```

  This syntax indicates you must type the keywords CREATE SYNONYM or create synonym without adding or deleting spaces or letters.

abc
: Substitute a value for any term that is displayed in lowercase letters. In the previous example, substitute a value for synonym-name.

( )
: Type any parentheses as shown. They are part of the syntax of an SQL statement and are not special symbols.

[ ]
: Do not type brackets as part of a statement. They surround any part of a statement that is optional. For example:

  ```
  CREATE [TEMP] TABLE
  ```

  This syntax indicates that you can type either CREATE TABLE or CREATE TEMP TABLE.

|
: The vertical bar indicates a choice among several options. For example:

  ```
  [VANILLA | CHOCOLATE [MINT] | STRAWBERRY]
  ```

  This syntax indicates that you can enter either VANILLA, CHOCOLATE, or STRAWBERRY and that, if you enter CHOCOLATE, you can also enter MINT.

{ }
: When you must choose only one of several options, the options are enclosed in braces and are separated by vertical bars. For example:

  ```
  {GUAVA | MANGO | PASSIONFRUIT}
  ```

  This syntax indicates that you must enter either GUAVA, MANGO, or PASSIONFRUIT, but you cannot enter more than one choice.

...
: An ellipsis indicates that you can enter an indefinite number of additional items, such as the one immediately preceding the ellipsis. For example:

```
old-column-name
...
```

This syntax indicates that you can enter a series of existing column names after the first one.

The *IBM Informix Guide to SQL: Syntax* contains more detailed syntax diagrams and instructions for interpreting the diagram format that is used in the publication.

Demonstration SQL

Various command files that are available with DB-Access.

The command files all have the extension .sql when displayed from the command line but are displayed without the extension on the SQL CHOOSE menu.

Keywords in these command files are shown in uppercase letters to make the SQL statements easier to read. Keywords in the actual command files are lowercase.
Important: Although the command files are listed alphabetically in this appendix, you cannot run the command files that create tables in that order without causing errors. The order in which the tables are created is important because of the referential constraints that link those tables.
When you select the Choose option on the SQL menu, the CHOOSE screen opens. The screen shows a list of the command files that you can access, similar to the display that the following figure shows. These files are included with the **stores_demo** database. Other .sql files are described later in this appendix.

Figure 1. Command files listed on the CHOOSE screen

```
CHOOSE >>
Choose a command file with the Arrow Keys, or enter a name, then press Return.

----------------- stores_demo @dbserver1 -------------- Press CTRL-W for Help ------

alt_cat             c_state             d_trig              sel_ojoin1

c_calls             c_stock             d_view              sel_ojoin2

c_cat               c_stores_demo       del_stock           sel_ojoin3

c_custom            c_table             ins_table           sel_ojoin4

c_index             c_trig              opt_disk            sel_order

c_items             c_type              sel_agg             sel_sub

c_manuf             c_view1             sel_all             sel_union

c_orders            c_view2             sel_group           upd_table

c_proc              d_proc              sel_join
```

If you do not see the command files included with your demonstration database, check the following:

- Did you copy the demonstration SQL command files to your current directory when you ran the demonstration database initialization script? If not, you can rerun the initialization script to copy them.
- Did you start DB-Access from the directory in which you installed the demonstration SQL command files? If not, exit DB-Access, change to the appropriate directory, and start DB-Access again.

For instructions about running the initialization script, copying command files, and starting DB-Access, seedbaccess command: Start DB-Access.

Use these command files with DB-Access for practice with SQL and the demonstration database. You can rerun the demonstration database initialization script whenever you want to refresh the database tables and SQL files.

- SQL files for the relational database model
  You can run sample SQL command files on the **stores_demo** demonstration database.
- SQL files for the Dimensional Database Model
  You can implement a dimensional database for data-warehousing applications by running scripts that create the **sales_demo** database.
- User-defined routines for the object-relational database model
  You can run sample user-defined routines on the **superstores_demo** database.

SQL files for the relational database model

You can run sample SQL command files on the **stores_demo** demonstration database.

- The alt_cat.sql command file
- The c_calls.sql command file
- The c_cat.sql command file
- The c_custom.sql command file
- The c_index.sql command file
- The c_items.sql command file
- The c_manuf.sql command file
- The c_orders.sql file
- The c_proc.sql command file
- The c_state command file
- The c_stock.sql command file
- The c_stores.sql command file
- The c_table.sql command file
- The c_trig.sql command file

Related information:
The stores_demo Database

The alt_cat.sql command file

The following command file alters the **catalog** table. It drops the existing constraint **aa** on the **catalog** table and adds a new constraint, **ab**, which specifies cascading deletes. You can use this command file and then the del_stock.sql command file for practice with cascading deletes on a database with logging.

```
ALTER TABLE catalog DROP CONSTRAINT aa;

ALTER TABLE catalog ADD CONSTRAINT
   (FOREIGN KEY (stock_num, manu_code) REFERENCES stock
    ON DELETE CASCADE CONSTRAINT ab);
```

The c_calls.sql command file

The following command file creates the **cust_calls** table:

```
CREATE TABLE cust_calls
   (
   customer_num          INTEGER,
   call_dtime            DATETIME YEAR TO MINUTE,
   user_id               CHAR(18) DEFAULT USER,
   call_code             CHAR(1),
   call_descr            CHAR(240),
   res_dtime             DATETIME YEAR TO MINUTE,
   res_descr             CHAR(240),
   PRIMARY KEY (customer_num, call_dtime),
   FOREIGN KEY (customer_num) REFERENCES customer (customer_num),
   FOREIGN KEY (call_code) REFERENCES call_type (call_code)
   );
```

The c_cat.sql command file

The following command file creates the **catalog** table. It contains a constraint, **aa**, which allows you to practice with cascading deletes by running the SQL statements in the alt_cat.sql and del_stock.sql command files on a database with logging.

```
CREATE TABLE catalog
   (
   catalog_num      SERIAL(10001),
   stock_num        SMALLINT NOT NULL,
   manu_code        CHAR(3) NOT NULL,
   cat_descr        TEXT,
   cat_picture      BYTE,
   cat_advert       VARCHAR(255, 65),
   PRIMARY KEY (catalog_num),
   FOREIGN KEY (stock_num, manu_code) REFERENCES stock
      CONSTRAINT aa
   );
```

The c_custom.sql command file

The following command file creates the **customer** table:

```
CREATE TABLE customer
   (
   customer_num       SERIAL(101),
   fname              CHAR(15),
   lname              CHAR(15),
   company            CHAR(20),
   address1           CHAR(20),
   address2           CHAR(20),
   city               CHAR(15),
   state              CHAR(2),
   zipcode            CHAR(5),
   phone              CHAR(18),
   PRIMARY KEY (customer_num)
   );
```

The c_index.sql command file

The following command file creates an index on the **zipcode** column of the **customer** table:

```
CREATE INDEX zip_ix ON customer (zipcode);
```

The c_items.sql command file

The following command file creates the **items** table:

```
CREATE TABLE items
   (
   item_num           SMALLINT,
   order_num          INTEGER,
   stock_num          SMALLINT NOT NULL,
   manu_code          CHAR(3) NOT NULL,
   quantity           SMALLINT CHECK (quantity >= 1),
   total_price        MONEY(8),
   PRIMARY KEY (item_num, order_num),
   FOREIGN KEY (order_num) REFERENCES orders (order_num),
   FOREIGN KEY (stock_num, manu_code) REFERENCES stock
      (stock_num, manu_code)
   );
```

The c_manuf.sql command file

The following command file creates the **manufact** table:

```
CREATE TABLE manufact
   (
   manu_code               CHAR(3),
   manu_name               CHAR(15),
   lead_time               INTERVAL DAY(3) TO DAY,
   PRIMARY KEY (manu_code)
   );
```

The c_orders.sql file

The following command file creates the **orders** table:

```
CREATE TABLE orders
   (
   order_num            SERIAL(1001),
   order_date           DATE,
   customer_num         INTEGER NOT NULL,
   ship_instruct        CHAR(40),
   backlog              CHAR(1),
   po_num               CHAR(10),
   ship_date            DATE,
   ship_weight          DECIMAL(8,2),
   ship_charge          MONEY(6),
   paid_date            DATE,
   PRIMARY KEY (order_num),
   FOREIGN KEY (customer_num) REFERENCES customer (customer_num)
   );
```

The c_proc.sql command file

The following command file creates an SPL routine. It reads the full name and address of a customer and takes a last name as its only argument.

This routine shows the legacy use of CREATE PROCEDURE.

To conform with the SQL standard preferred with IBM® Informix®, define a *function* if you want to return values from a routine.

```
CREATE PROCEDURE read_address (lastname CHAR(15))
   RETURNING CHAR(15), CHAR(15), CHAR(20), CHAR(15), CHAR(2), CHAR(5);
   DEFINE p_fname, p_city CHAR(15);
   DEFINE p_add CHAR(20);
   DEFINE p_state CHAR(2);
   DEFINE p_zip CHAR(5);
   SELECT fname, address1, city, state, zipcode
      INTO p_fname,  p_add, p_city, p_state, p_zip
      FROM customer
      WHERE lname = lastname;

   RETURN p_fname, lastname, p_add, p_city, p_state, p_zip;

END PROCEDURE;
```

The c_state command file

The following command file creates the **state** table:

```
CREATE TABLE state
   (
   code                CHAR(2),
   sname               CHAR(15),
   PRIMARY KEY (code)
   );
```

The c_stock.sql command file

The following command file creates the **stock** table:

```
CREATE TABLE stock
   (
   stock_num           SMALLINT,
   manu_code           CHAR(3),
   description         CHAR(15),
   unit_price          MONEY(6),
   unit                CHAR(4),
   unit_descr          CHAR(15),
   PRIMARY KEY (stock_num, manu_code),
   FOREIGN KEY (manu_code) REFERENCES manufact
   );
```

The c_stores.sql command file

The following command file creates the **stores_demo** database:

```
CREATE DATABASE stores_demo;
```

The c_table.sql command file

The following command file creates a database named **restock** and then creates a custom table named **sports** in that database:

```
CREATE DATABASE restock;

CREATE TABLE sports
   (
   catalog_no          SERIAL UNIQUE,
   stock_no            SMALLINT,
   mfg_code            CHAR(5),
   mfg_name            CHAR(20),
   phone               CHAR(18),
   descript            VARCHAR(255)
   );
```

The c_trig.sql command file

The following command file creates a table named **log_record** and then creates a trigger named **upqty_i**, which updates it:

```
CREATE TABLE log_record
   (item_num    SMALLINT,
   ord_num      INTEGER,
   username     CHARACTER(8),
   update_time  DATETIME YEAR TO MINUTE,
   old_qty      SMALLINT,
```

```
   new_qty      SMALLINT);

CREATE TRIGGER upqty_i
UPDATE OF quantity ON items
REFERENCING OLD AS pre_upd
        NEW AS post_upd
FOR EACH ROW(INSERT INTO log_record
   VALUES (pre_upd.item_num, pre_upd.order_num, USER, CURRENT,
         pre_upd.quantity, post_upd.quantity));
```

The c_type.sql command file

The following command file creates the **call_type** table:

```
CREATE TABLE call_type
   (
   call_code               CHAR(1),
   code_descr              CHAR(30),
   PRIMARY KEY (call_code)
   );
```

The c_view1.sql command file

The following command file creates a view called **custview** on a single table and grants privileges on the view to **public**. It includes the WITH CHECK OPTION keywords to verify that any changes made to underlying tables through the view do not violate the definition of the view.

```
CREATE VIEW custview (firstname, lastname, company, city) AS
   SELECT fname, lname, company, city
      FROM customer
      WHERE city = 'Redwood City'
      WITH CHECK OPTION;

GRANT DELETE, INSERT, SELECT, UPDATE
   ON custview
   TO public;
```

The c_view2.sql command file

The following command file creates a view on the **orders** and **items** tables:

```
CREATE VIEW someorders (custnum,ocustnum,newprice) AS
   SELECT orders.order_num, items.order_num,
         items.total_price*1.5
      FROM orders, items
      WHERE orders.order_num = items.order_num
      AND items.total_price > 100.00;
```

The d_proc.sql command file

The following command file drops the SPL routine that the c_proc.sql command file created:

```
DROP PROCEDURE read_address;
```

The d_trig.sql command file

The following command file drops the trigger that the c_trig.sql command file created:

```
DROP TRIGGER upqty_i;
```

The d_view.sql command file

The following command file drops the view named **custview** that the c_view1.sql command file created:

```
DROP VIEW custview;
```

The del_stock.sql command file

The following command file deletes rows from the **stock** table where the stock number is 102. This delete will cascade to the **catalog** table (although the related manufacturer codes will remain in the **manufact** table). The del_stock.sql command file can be used following the alt_cat.sql command file for practice with cascading deletes on a database with logging.

```
DELETE FROM stock WHERE stock_num = 102;
```

After running the SQL statements in the alt_cat.sql and del_stock.sql command files, issue the following query on the **catalog** table to verify that the rows were deleted:

```
SELECT * FROM catalog WHERE stock_num = 102;
```

The **stores_demo** database has been changed. You might want to rerun the **dbaccessdemo** script to rebuild the original database.

The ins_table.sql command file

The following command file inserts one row into the **sports** table that the c_table.sql command file created:

```
INSERT INTO sports
   VALUES (0,18,'PARKR', 'Parker Products', '503-555-1212',
   'Heavy-weight cotton canvas gi, designed for aikido or
   judo but suitable for karate. Quilted top with side ties,
   drawstring waist on pants. White with white belt.
   Pre-washed for minimum shrinkage. Sizes 3-6.');
```

The sel_agg.sql command file

The SELECT statement in the following command file queries on table data using aggregate functions. It combines the aggregate functions MAX and MIN in a single statement.

```
SELECT MAX (ship_charge), MIN (ship_charge)
   FROM orders;
```

The sel_all.sql command file

The following example command file contains all seven SELECT statement clauses that you can use in the IBM® Informix® implementation of interactive SQL. This SELECT statement joins the **orders** and **items** tables. It also uses display labels, table aliases, and integers as column indicators; groups and orders the data; and puts the results into a temporary table.

```
SELECT o.order_num, SUM (i.total_price) price,
     paid_date - order_date span
   FROM orders o, items i
   WHERE o.order_date > '01/01/90'
      AND o.customer_num > 110
      AND o.order_num = i.order_num
   GROUP BY 1, 3
   HAVING COUNT (*) < 5
   ORDER BY 3
   INTO TEMP temptab1;
```

The sel_group.sql command file

The following example command file includes the GROUP BY and HAVING clauses. The HAVING clause usually complements a GROUP BY clause by applying one or more qualifying conditions to groups after they are formed, which is similar to the way the WHERE clause qualifies individual rows. (One advantage to using a HAVING clause is that you can include aggregates in the search condition; you cannot include aggregates in the search condition of a WHERE clause.)

Each HAVING clause compares one column or aggregate expression of the group with another aggregate expression of the group or with a constant. You can use the HAVING clause to place conditions on both column values and aggregate values in the group list.

```
SELECT order_num, COUNT(*) number, AVG (total_price) average
   FROM items
   GROUP BY order_num
   HAVING COUNT(*) > 2;
```

The sel_join.sql command file

The following example command file uses a simple join on the **customer** and **cust_calls** tables. This query returns only those rows that show the customer has made a call to customer service.

```
SELECT c.customer_num, c.lname, c.company,
     c.phone, u.call_dtime, u.call_descr
   FROM customer c, cust_calls u
   WHERE c.customer_num = u.customer_num;
```

The sel_ojoin1.sql command file

The following example command file uses a simple outer join on two tables. The use of the keyword OUTER in front of the **cust_calls** table makes it the subservient table. An outer join causes the query to return information about all customers, even if they do not make calls to customer service. All rows from the dominant **customer** table are retrieved, and null values are assigned to corresponding rows from the subservient **cust_calls** table.

```
SELECT c.customer_num, c.lname, c.company,
     c.phone, u.call_dtime, u.call_descr
   FROM customer c, OUTER cust_calls u
   WHERE c.customer_num = u.customer_num;
```

The sel_ojoin2.sql command file

The following example command file creates an outer join, which is the result of a simple join to a third table. This second type of outer join is called a *nested simple join*.

This query first performs a simple join on the **orders** and **items** tables, retrieving information about all orders for items with a **manu_code** of KAR or SHM. It then performs an outer join, which combines this information with data from the dominant **customer** table. An optional ORDER BY clause reorganizes the data.

```
SELECT c.customer_num, c.lname, o.order_num,
     i.stock_num, i.manu_code, i.quantity
   FROM customer c, OUTER (orders o, items i)
   WHERE c.customer_num = o.customer_num
      AND o.order_num = i.order_num
      AND manu_code IN ('KAR', 'SHM')
   ORDER BY lname;
```

The sel_ojoin3.sql command file

The following example SELECT statement is the third type of outer join, known as a *nested outer join*. It queries on table data by creating an outer join, which is the result of an outer join to a third table.

This query first performs an outer join on the **orders** and **items** tables, retrieving information about all orders for items with a **manu_code** of KAR or SHM. It then performs an outer join, which combines this information with data from the dominant **customer** table. This query preserves order numbers that the previous example eliminated, returning rows for orders that do not contain items with either manufacturer code. An optional ORDER BY clause reorganizes the data.

```
SELECT c.customer_num, lname, o.order_num,
     stock_num, manu_code, quantity
   FROM customer c, OUTER (orders o, OUTER items i)
   WHERE c.customer_num = o.customer_num
      AND o.order_num = i.order_num
      AND manu_code IN ('KAR', 'SHM')
   ORDER BY lname;
```

The sel_ojoin4.sql command file

The following example queries on table data using the fourth type of outer join. This query shows an outer join, which is the result of an outer join of each of two tables to a third table. In this type of outer join, join relationships are possible *only* between the dominant table and subservient tables.

This query individually joins the subservient tables **orders** and **cust_calls** to the dominant **customer** table but does not join the two subservient tables. (An INTO TEMP clause selects the results into a temporary table.)

```
SELECT c.customer_num, lname, o.order_num,
     order_date, call_dtime
   FROM customer c, OUTER orders o, OUTER cust_calls x
   WHERE c.customer_num = o.customer_num
      AND c.customer_num = x.customer_num
   INTO temp service;
```

The sel_order.sql command file

The following example uses the ORDER BY and WHERE clauses to query. In this SELECT statement, the comparison 'bicycle%' (LIKE condition, or 'bicycle*' for a MATCHES condition) specifies the letters bicycle followed by any sequence of zero or more characters. It narrows the search further by adding another comparison condition that excludes a **manu_code** of PRC.

```
SELECT * FROM stock
   WHERE description LIKE 'bicycle%'
      AND manu_code NOT LIKE 'PRC'
   ORDER BY description, manu_code;
```

The sel_sub.sql command file

The following example uses a subquery to query. This self-join uses a correlated subquery to retrieve and list the 10 highest-priced items ordered.

```
SELECT order_num, total_price
   FROM items a
   WHERE 10 >
      (SELECT COUNT (*)
         FROM items b
         WHERE b.total_price < a.total_price)
   ORDER BY total_price;
```

The sel_union.sql command file

The following example uses the UNION clause to query on data in two tables. The compound query performs a union on the **stock_num** and **manu_code** columns in the **stock** and **items** tables. The statement selects items that have a unit price of less than $25.00 or that have been ordered in quantities greater than three, and it lists their **stock_num** and **manu_code**.

```
SELECT DISTINCT stock_num, manu_code
   FROM stock
   WHERE unit_price < 25.00

UNION

SELECT stock_num, manu_code
   FROM items
   WHERE quantity > 3;
```

The upd_table.sql command file

The following example updates the **sports** table that the c_table.sql command file created:

```
UPDATE sports
   SET phone = '808-555-1212'
   WHERE mfg_code = 'PARKR';
```

SQL files for the Dimensional Database Model

You can implement a dimensional database for data-warehousing applications by running scripts that create the **sales_demo** database.

The **sales_demo** database is based on the **stores_demo** schema and data.

To create the **sales_demo** database:

1. Create a **stores_demo** database with the following command:

   ```
   dbaccessdemo -log
   ```

2. Make sure that the createdw.sql and loaddw.sql files are in the same directory as the files with extension .unl that the loaddw.sql uses.
3. Run the createdw.sql file.
4. Run the loaddw.sql file.

- The createdw.sql file
- The loaddw.sql file

The createdw.sql file

This file creates the new **sales_demo** database with logging and then creates tables within that database. It contains the following statements:

```
create database sales_demo with log;

create table product (
   product_code integer,
   product_name char(31),
   vendor_code char(3),
   vendor_name char(15),
   product_line_code smallint,
   product_line_name char(15));

create table customer (
   customer_code integer,
   customer_name char(31),
   company_name char(20));


create table sales (
   customer_code integer,
   district_code smallint,
   time_code integer,
   product_code integer,
```

```
   units_sold smallint,
   revenue money (8,2),
   cost money (8,2),
   net_profit money(8,2));

create table time
   (
   time_code int,
   order_date date,
   month_code smallint,
   month_name char(10),
   quarter_code smallint,
   quarter_name char(10),
   year integer
   );

create table geography (
    district_code serial,
   district_name char(15),
   state_code char(2),
   state_name char(18),
   region smallint);
```

The loaddw.sql file

This file contains the commands necessary to load data from two sources:

- The files with the extension .unl in your demonstration directory
- Data selected from the **stores_demo** database

These SQL statements in loaddw.sql accomplish these actions:

```
connect to "stores_demo ";
load from "add_orders.unl"
   insert into stores_demo :orders;
load from 'add_items.unl'
   insert into stores_demo :items;

connect to "sales_demo";
load from 'costs.unl'
   insert into cost;
load from 'time.unl'
   insert into time;

insert into geography(district_name, state_code, state_name)
   select distinct c.city, s.code, s.sname
from stores_demo :customer c, stores_demo :state s
   where c.state = s.code;
update geography      -- converts state_code values to region values
   set region = 1
   where state_code = "CA";
update geography
   set region = 2
   where state_code <> "CA";

insert into customer (customer_code, customer_name, company_name)
   select c.customer_num, trim(c.fname) || " " || c.lname, c.company
   from stores_demo :customer c;

insert into product (product_code, product_name, vendor_code,
   vendor_name, product_line_code, product_line_name)
 select a.catalog_num,
      trim(m.manu_name) || " "|| s.description,
      m.manu_code, m.manu_name, s.stock_num, s.description
   from stores_demo :catalog a, stores_demo :manufact m,
      stores_demo :stock s
   where a.stock_num = s.stock_num and
      a.manu_code = s.manu_code and
       s.manu_code = m.manu_code;
insert into sales (customer_code, district_code,
   time_code, product_code,
   units_sold, revenue, cost, net_profit)
   select c.customer_num, g.district_code, t.time_code, p.product_code,
   SUM(i.quantity), SUM(i.total_price),
   SUM(i.quantity * x.cost),
   SUM(i.total_price) - SUM(i.quantity * x.cost)
   from stores_demo :customer c, geography g, time t,
       product p,
      stores_demo :items i, stores_demo :orders o, cost x
    where c.customer_num = o.customer_num and
       o.order_num = i.order_num and
       p.product_line_code = i.stock_num and
      p.vendor_code = i.manu_code and
      t.order_date = o.order_date and
      p.product_code = x.product_code and
      c.city = g.district_name
      GROUP BY 1,2,3,4;

connect to "stores_demo ";
load from 'add_orders.unl'
   insert into stores_demo :orders;
```

```
load from 'add_items.unl'
   insert into stores_demo :items;

connect to "sales_demo";
load from 'costs.unl'
   insert into cost;
load from 'time.unl'
   insert into time;

insert into geography(district_name, state_code, state_name)
   select distinct c.city, s.code, s.sname
from stores_demo :customer c, stores_demo :state s
   where c.state = s.code;
update geography      -- converts state_code values to region values
   set region = 1
   where state_code = "CA";
update geography
   set region = 2
   where state_code <> "CA";

insert into customer (customer_code, customer_name, company_name)
   select c.customer_num, trim(c.fname) || " " || c.lname, c.company
   from stores_demo :customer c;

insert into product (product_code, product_name, vendor_code,
   vendor_name, product_line_code, product_line_name)
 select a.catalog_num,
      trim(m.manu_name) || " " || s.description,
      m.manu_code, m.manu_name, s.stock_num, s.description
   from stores_demo :catalog a, stores_demo :manufact m,
      stores_demo :stock s
   where a.stock_num = s.stock_num and
      a.manu_code = s.manu_code and
       s.manu_code = m.manu_code;

insert into sales (customer_code, district_code,
   time_code, product_code,
   units_sold, revenue, cost, net_profit)
   select c.customer_num, g.district_code, t.time_code, p.product_code,
   SUM(i.quantity), SUM(i.total_price),
   SUM(i.quantity * x.cost),
   SUM(i.total_price) - SUM(i.quantity * x.cost)
   from stores_demo :customer c, geography g, time t, product p,
      stores_demo :items i, stores_demo :orders o, cost x
    where c.customer_num = o.customer_num and
       o.order_num = i.order_num and
       p.product_line_code = i.stock_num and
      p.vendor_code = i.manu_code and
      t.order_date = o.order_date and
      p.product_code = x.product_code and
      c.city = g.district_name
      GROUP BY 1,2,3,4;
```

User-defined routines for the object-relational database model

You can run sample user-defined routines on the **superstores_demo** database.

The **superstores_demo** database does not replace the **stores_demo** database. Both databases are available. The **superstores_demo** database schema is not compatible with earlier versions with **stores_demo**. In many cases, you cannot use test queries developed for **stores_demo** against the tables of **superstores_demo** because the tables differ.

No SQL command files are associated specifically with **superstores_demo**. However, there are user-defined routines that you can run in the SQL editor or a system editor.

The **superstores_demo** database includes examples of the following features:

- Collection types: SET, LIST
- Named row types: location_t, loc_us_t, loc_non_us_t
- Unnamed row types
- Type and table inheritance
- Built-in data types: BOOLEAN, SERIAL8, INT8
- Distinct data type: percent
- Smart large objects: BLOB and CLOB

The **superstores_demo** database has row types and tables to support the following table-inheritance hierarchies:

- customer/retail_customer
- customer/whlsale_customer
- location/location_us
- location/location_non_us

For more information about user-defined routines, see *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

Related information:
The superstores_demo database

High-Performance Loader User's Guide

These topics describe how to use the IBM® Informix® High-Performance Loader (HPL) to load and unload large quantities of data efficiently to or from databases.

These topics describe:

- The architecture of HPL
- The **onpload** utility that allows you to control HPL from a script.
- The onpload database that maintains information about load and unload jobs that you have prepared.
- The **ipload** graphical user interface (GUI).
- The **onpladm** utility.

The **ipload** utility is a UNIX application that helps users prepare load and unload jobs for both UNIX and Windows. The **onpladm** utility is a command-line version of the **ipload** utility that operates on both UNIX and Windows.

The first section introduces the HPL, provides a general overview of the tasks that the HPL performs, describes the architecture of the HPL, and includes two tutorial examples that take you through the process of loading and unloading data.

The second section introduces the **ipload** utility, a graphical user interface that you can use to set the parameters for the HPL.

Subsequent sections give details about developing the onpload database by using the individual components of the **ipload**, **onpload**, and **onpladm** utilities.

These topics are written for the following users:

- Database administrators
- Database server administrators

These topics assume that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides.
- Some experience working with relational databases or exposure to database concepts.
- Some experience with computer programming.
- Some experience with database server administration, operating-system administration, or network administration.

For information about software compatibility, see the IBM Informix release notes.

These topics are taken from *IBM Informix High-Performance Loader User's Guide*.

- High-Performance Loader overview
- Examples of loading and unloading jobs using the ipload utility
  This section shows how the components of the High-Performance Loader (HPL) fit together. The section includes a step-by-step tutorial with two examples (for a load and an unload job) that use the **ipload** graphical user interface (GUI).
- The ipload utility windows
  This section contains information about using the **ipload** utility.
- Define HPL projects
  This section explains how to create a project and how projects are related. The individual components that you store in projects are described in later sections.
- Configure the High-Performance Loader
  This section describes the process of configure the High-Performance Loader (HPL).
- Define device arrays
  This section describes how to define and use device arrays with the High-Performance Loader (HPL).
- Define formats
  This section describes the formats that the High-Performance Loader (HPL) provides and shows how to prepare and edit the format component.
- Define queries
  This section describe how to define queries including how to create, edit, and export and import queries.
- Define maps
  This section describes how to define maps with the High-Performance Loader (HPL). It also describes the options that are available for defining maps.
- Define filters
  This section describes how to create, edit, and delete filters.
- Unload data from a database
  This section describes the Unload Job window.
- Load data to a database table
  This section describes the load process.
- The Generate options of the ipload utility
  This section describes the Generate options for the **ipload** utility.
- The HPL browsing options
  This section describes the browsing options that are available for the High-Performance Loader (HPL).
- Manage the High-Performance Loader
- The onpload utility
  This section describes how to use the **onpload** utility.
- The onpladm utility
  This section describes how to use the **onpladm** utility.
- Appendixes
  This section contains additional reference information.

High-Performance Loader overview

- Overview of HPL features
  The High-Performance Loader (HPL) is a database server tool that you can use to load and unload large quantities of data efficiently to or from a database.
- The HPL data-load process
  The data-load process reads a source data file, converts the data to a different format, and inserts the converted data into a database table.
- The HPL data-unload process

- HPL loading modes
  The High-Performance Loader (HPL) offers two load modes, deluxe mode and express mode. The express mode is faster, and the deluxe mode is more flexible.
- HPL components
  The HPL consists of the **onpload** utility, the **ipload** utility, the **onpladm** command-line utility, and the onpload database.
- Environment variables needed for the HPL
  The High-Performance Loader (HPL) is part of the database server, so you must start the database server before you use the HPL. Before you start the database server and use HPL, you must set environment variables.
- Architecture of the onpload utility
  The **ipload** utility is the interface that allows you to prepare the parameters that the **onpload** utility uses. The **onpload** utility, which is a client application that attaches to the database server, actually loads and unloads the data.

Overview of HPL features

The High-Performance Loader (HPL) is a database server tool that you can use to load and unload large quantities of data efficiently to or from a database.

The HPL lets you exchange data with tapes, data files, and programs, and converts data from these sources into a format compatible with IBM® Informix® databases. The HPL also allows you to manipulate and filter the data as you perform load and unload operations.

The HPL includes the following components:

- The **ipload** utility, which is a graphical user interface (GUI). This UNIX application helps you prepare load and unload jobs for both UNIX and Windows and includes the following functionality:
  - The **ipload** utility provides a Generate option that lets you automatically generate the HPL components that are required for a load or unload job.
  - The database-load feature lets you update your databases with data from any of the supported file types, while allowing you to control the format and selection of records from the input files. Data can be loaded from or unloaded to files, tapes, or application pipes (for UNIX), or to any combination of these three device types.
  - The load and unload operations run in the background of your multitasking operating system. Once the operation begins, you can continue to use **ipload** to perform other functions.
  - The **ipload** utility provides context-sensitive online help. The online help also includes a glossary.
- The **onpladm** utility, which is a command-line version of the **ipload** utility.
- The **onpload** utility, which allows you to control HPL from a script.

- The onpload database, which maintains information about load and unload jobs that you have prepared.

You use the **ipload** utility to manage the onpload database on any database server on your network.

Any database server on your network can use the onpload database, which contains parameters and controls that the HPL uses. This accessibility allows centralized management of your load and unload controls. These parameters and controls include the HPL components such as formats, maps, and projects.

The HPL:

- Supports COBOL, ASCII, multibyte, delimited, or binary data.
- Can load and unload data of a different GLS locale than that of the database server.
- Provides synonym support for tables that are valid for the local database server. You can use synonyms for both the load and unload operations.
- Provides support for unloading data with a query that accesses a view in its SELECT statement.
- Supports loading of raw tables in express mode.

The HPL data-load process

The data-load process reads a source data file, converts the data to a different format, and inserts the converted data into a database table.

The source data can come from one or more of the following sources:

- Files
- Tapes
- Pipes (application-generated data) (UNIX)

The High-Performance Loader (HPL) supports load and unload data greater than 2 GB to files and tapes.

During conversion, the source data is often manipulated so that the converted data displays different characteristics. Examples of this manipulation include:

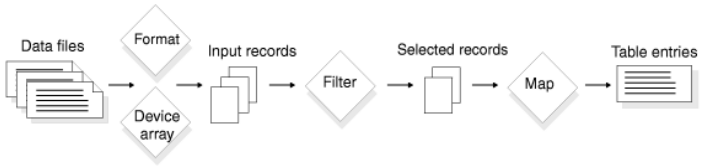
- Changing lowercase letters to uppercase letters
- Loading default values, loading certain table columns, or replacing nulls
- Masking the data to include only part of a value
- Converting from one data type to another, such as conversion of a numeric string to a float
- Converting from the code set of one locale to the code set of another locale

When you prepare to run a data load with the HPL, you describe the actions that the HPL must take by defining a set of metadata *components*. The components describe different aspects of the load process. The HPL uses information from:

- The device array to find the set of the source-data files
- The format to define the data types and layout of the source data
- The filter to select the records from the source data that should be written to the database table
- The map to modify and reorganize the data

The **ipload** utility helps you prepare the components. Load data to a database table, addresses the process of loading a file to a database.

The following figure shows the data load process. The figure summarizes the data-load process by showing how data moves from data files to table entries.
Figure 1. The data-load process



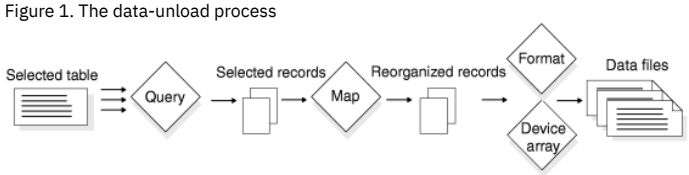
The HPL data-unload process

The data-unload process is essentially the same as the load process, but in reverse. The data-unload process extracts the source data from one or more database tables; converts the data to a new format; and writes the converted data to a file, tape, or on UNIX to a pipe (application). As in a load, you can manipulate the data from a database table so that the converted data displays different characteristics.

The following figure shows how the components of the High-Performance Loader (HPL) affect the data as it moves from a database to data files during the unload process. The HPL uses:

- The query to select records from the database
- The map to reorganize or modify the selected records
- The format to prepare the records for writing out to the data files
- The device array to find the location of the data files

Figure 1. The data-unload process



The HPL uses the same components for an unload job as for a load job, with one exception. For an unload job, **ipload** creates a Structured Query Language (SQL) query that extracts selected data from the table. As with a load, unload components are grouped together into an unload job. Unload jobs can be saved, retrieved, and rerun as many times as necessary. Unload jobs can be grouped together with load jobs in the same project.

Related reference:
Unload data from a database

HPL loading modes

The High-Performance Loader (HPL) offers two load modes, deluxe mode and express mode. The express mode is faster, and the deluxe mode is more flexible.

The HPL express mode

The HPL express-mode loads are faster than deluxe-mode loads, but less flexible. In express mode, you cannot update the table or read the new data entries until the load is complete. The express mode disables indexes, constraints, and triggers during the load. After the load, HPL rebuilds and re-enables indexes, evaluates and re-enables constraints, if possible, and re-enables triggers. (HPL does not evaluate triggers with respect to the loaded data.)

You must perform a level-0 backup after an express-mode load.

The HPL deluxe mode

The HPL deluxe-mode loads are not as fast as express-mode loads, but are more flexible. In deluxe mode, you can access and update the table that is being loaded. The deluxe mode updates indexes, performs constraint checking, and evaluates triggers as data is inserted into the table.

Related reference:
Manage the High-Performance Loader

HPL components

The HPL consists of the **onpload** utility, the **ipload** utility, the **onpladm** command-line utility, and the onpload database.

- The onpload utility
- The ipload utility
- The onpladm utility
  The **onpladm** utility is a command-line utility for both UNIX and Windows with the same functionality as **ipload**.
- The onpload database
- Relationships among the parts of the HPL

The onpload utility

The **onpload** utility has the following features:

- Converts, filters, and moves data between a database and a storage device
- Uses information from the onpload database to run the load and unload jobs and to convert the data
- Records information during a load about data records that do not meet the load criteria

The **onpload** utility can load or unload data from files that are larger than 2 GB and can generate .log, .rej and .flt files that are larger than 2 GB.

The **onpload** database must be accessible from the machine on which the **onpload** utility is run. It does not need to be on the same machine.

The database that the **onpload** loads or unloads must be in a database server instance on the machine that is running the **onpload** (or **ipload** or **onpladm**) utility.

You can start **onpload** by using **ipload** or from the command line.

When you start **onpload** from **ipload**, the **onpload** and **ipload** utilities communicate by using a socket connection. The **onpload** utility runs independently from **ipload** (see the information to the right of the vertical line in Figure 1).

Related concepts:
Architecture of the onpload utility
Related reference:
The onpload utility

The ipload utility

The **ipload** utility is a UNIX-based graphical user interface that has the following features:

- Creates and manages the onpload database
- Creates and stores information for onpload
- Lets you create, edit, and group the components of the load and unload jobs
- Stores information about the load components in the database

You can use **ipload** to manage onpload databases on both UNIX and Windows. The **ipload** utility is not available on Mac OS X.

Related concepts:
Start the ipload utility
Related reference:
The ipload utility
Configure the ipload utility
Run load and unload jobs on a Windows computer

The onpladm utility

The **onpladm** utility is a command-line utility for both UNIX and Windows with the same functionality as **ipload**.

Related reference:
The onpladm utility

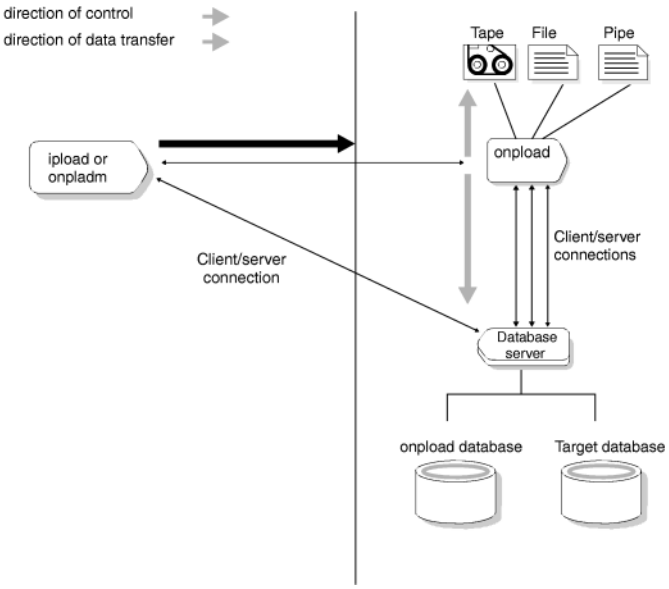
The onpload database

The onpload database has the following features:

- Contains information that the **onpload** utility requires to perform data loads and unloads
- Must be accessible from the machine where the **onpload** utility is run. It does not need to be on the same machine.
  However; the database that the utility loads or unloads must be in a database server instance on the machine where the **onpload** utility (and hence the **ipload** or **onpladm** utility) is being run.

Relationships among the parts of the HPL

The following figure shows the relationships among the parts of the High-Performance Loader (HPL). The **ipload** utility or **onpladm** command-line utility connects to the database server to populate the onpload database. The **ipload** utility or the **onpladm** utility controls the **onpload** utility, which uses multithreaded architecture to make multiple connections to the database server and multiple I/O connections to tapes, files, or pipes.
Figure 1. Relationships among the parts of the HPL



The information to the right of the vertical line in Figure 1 shows **onpload** loading or unloading data with no interaction from **ipload**.

Environment variables needed for the HPL

The High-Performance Loader (HPL) is part of the database server, so you must start the database server before you use the HPL. Before you start the database server and use HPL, you must set environment variables.

The following environment variable must be set:

- INFORMIXDIR
- ONCONFIG
- INFORMIXSERVER
- LD_LIBRARY_PATH

Some computers use the LD_LIBRARY_PATH environment variable for shared libraries. The name of this environment variable is platform-dependent. See your operating system documentation for the name of the environment variable that specifies the search path for shared libraries. Then see the machine notes for information about LD_LIBRARY_PATH.

You can use the IFX_ONPLOAD_AUTO_UPGRADE environment variable with the **ipload** or **onpladm** utilities to automatically upgrade the onpload database the first time you run an HPL utility by using the **ipload** or **onpladm** command after you migrate to a new database server version. You cannot use the IFX_ONPLOAD_AUTO_UPGRADE environment variable with the **onpload** utility.

In addition to the environment variables listed above, the following environment variables pertain to the HPL:

- DBONPLOAD
- PLCONFIG
- PLOAD_SHMBASE
- PLOAD_LO_PATH
- PLOAD_SHMAT

Tip: To maximize available memory and scan resources, HPL automatically sets the PDQPRIORITY environment variable to `100`, if it is not already set. If the PDQPRIORITY environment variable is set, HPL uses that value. If the PDQPRIORITY environment variable is set to `0`, then HPL cannot unload multiple devices.

- Preparing multiple onpload databases
  Each database server can have only one active onpload database. In most cases, you can use the default onpload database, which is named onpload. If you choose to prepare multiple databases to hold different types of control information, you must set the DBONPLOAD environment variable before you can run a load or unload.
- The PLCONFIG environment variable
- The PLOAD_SHMBASE environment variable
- The PLOAD_LO_PATH environment variable
  The PLOAD_LO_PATH environment variable allows you to specify the location of smart large object (CLOB or BLOB) output files. If this environment variable is not set, the database server puts the output files in /tmp.
- The PLOAD_SHMAT environment variable
  If the pload converter thread cannot attach to the passed address, you receive a message that asks you to set the PLOAD_SHMAT environment variable.

Related information:
INFORMIXDIR environment variable
INFORMIXSERVER environment variable
ONCONFIG environment variable
PDQPRIORITY environment variable

Preparing multiple onpload databases

Each database server can have only one active onpload database. In most cases, you can use the default onpload database, which is named onpload. If you choose to prepare multiple databases to hold different types of control information, you must set the DBONPLOAD environment variable before you can run a load or unload.

To prepare multiple onpload databases with **ipload**:

1. Set the DBONPLOAD environment variable to the name of the alternative onpload database.
2. Restart **ipload**.
3. Use **ipload** to prepare the alternative database.

During load and unload jobs, the **ipload**, **onpload**, and **onpladm** utilities use the value of the DBONPLOAD environment variable that is set in the client environment, regardless of whether the DBONPLOAD environment variable is set on the server. The **ipload**, **onpload**, and **onpladm** utilities use the value of the DBONPLOAD environment variable that is set on the server only if the DBONPLOAD environment variable is not set in the client environment.

Related concepts:
Create the onpload database
Related information:
DBONPLOAD environment variable

The PLCONFIG environment variable

The default configuration file for the **onpload** utility is plconfig.std. The configuration file is always in the $INFORMIXDIR/etc directory. To use an alternative configuration file, you must set the PLCONFIG environment variable to the name of the alternative **onpload** configuration file. If you use plconfig.std, you do not need to set PLCONFIG.

Related reference:
High-Performance Loader configuration file
Related information:
PLCONFIG environment variable

The PLOAD_SHMBASE environment variable

The PLOAD_SHMBASE environment variable allows you to specify shared-memory address attachments specifically for **onpload** processes. You can set the PLOAD_SHMBASE environment variable to avoid shared-memory collisions between **onpload** and the database server or allow the HPL to specify the attachments.
Tip: To use the PLOAD_SHMBASE environment variable, you must start **onpload** from the command line, not from within **ipload**.

- Avoid shared-memory collision
- Set the PLOAD_SHMBASE environment variable
  To override the initial shared-memory allocation in a shared-memory collision between **onpload** and the database server, set the PLOAD_SHMBASE environment variable to a value much higher or much lower than the value for the shared memory that the database server uses.

Related information:
PLOAD_SHMBASE environment variable

Avoid shared-memory collision

Both the database server and onpload allocate shared memory. When onpload starts up, the database server allocates shared memory for buffers for **onpload** processes. The **onpload** utility also allocates shared memory for its internal use. Because of the dynamic nature of shared-memory allocations, shared-memory collisions can occur between onpload and the database server. If this collision occurs, the **onpload** job can also fail and error messages are sent to the **onpload** log file or the database server log file.

To verify if a collision has occurred, use the **onstat -g seg** option. Check for overlap between the shared-memory segments that the database server is using and the SHMBASE reported in the **onpload** log file.

Related information:
onstat -g seg command: Print shared memory segment statistics

Set the PLOAD_SHMBASE environment variable

To override the initial shared-memory allocation in a shared-memory collision between **onpload** and the database server, set the PLOAD_SHMBASE environment variable to a value much higher or much lower than the value for the shared memory that the database server uses.

The PLOAD_LO_PATH environment variable

The PLOAD_LO_PATH environment variable allows you to specify the location of smart large object (CLOB or BLOB) output files. If this environment variable is not set, the database server puts the output files in /tmp.

Related information:
PLOAD_LO_PATH environment variable

The PLOAD_SHMAT environment variable

If the pload converter thread cannot attach to the passed address, you receive a message that asks you to set the PLOAD_SHMAT environment variable.

When you set the PLOAD_SHMAT environment variable, the pload converter calculates the address by using a global attached segment list that is maintained across pload virtual processors. The pload converter attaches at the next available address after the highest address on the list, ensuring that the converter always attaches to an unused shared memory segment.

Architecture of the onpload utility

The **ipload** utility is the interface that allows you to prepare the parameters that the **onpload** utility uses. The **onpload** utility, which is a client application that attaches to the database server, actually loads and unloads the data.

The **onpload** utility can take advantage of parallel processing to perform both I/O and data conversion as efficiently as possible because it uses the same multithreading architecture that the database server uses.

The following sections describe how **onpload** uses multithreading for deluxe loads, express loads, and unloads.

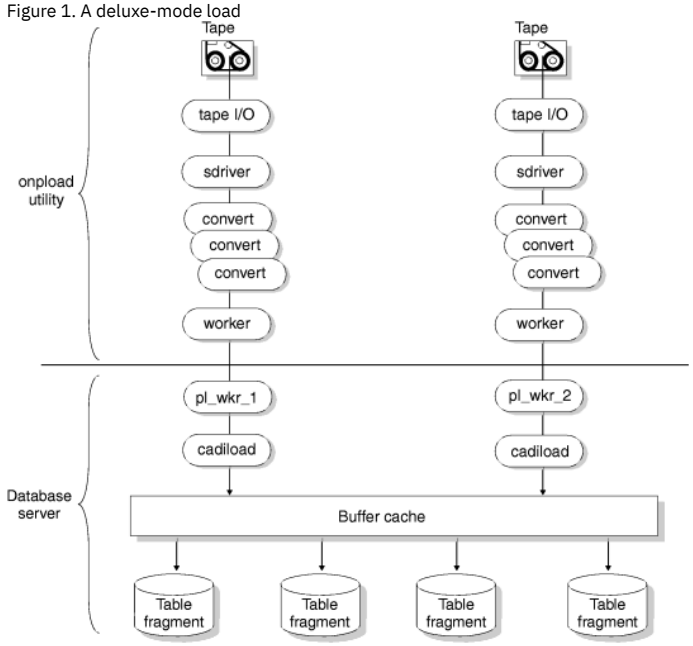
- The onpload utility deluxe-mode process
  The **onpload** utility uses specific threads during the deluxe-mode process.
- The onpload utility express-mode load process
  The **onpload** utility behaves the same way during express-mode deluxe-mode loads. However, the database server behaves differently during an express load.
- The onpload utility unload process

Related concepts:
The onpload utility
Related information:
Save memory and resources

The onpload utility deluxe-mode process

The **onpload** utility uses specific threads during the deluxe-mode process.

The following figure shows the threads that **onpload** uses in a deluxe-mode load process. In deluxe mode, data is subject to the same constraints as if it were loaded by using SQL INSERT statements.
Figure 1. A deluxe-mode load



- Threads that the onpload utility uses
- Threads that the database server uses

Related concepts:
The HPL deluxe mode

Threads that the onpload utility uses

The **onpload** utility starts the following threads:

Table 1. Threads that the onpload utility starts

Thread	Description
tape I/O threads	The **onpload** utility starts one **tape I/O** thread for each tape device. It reads data from the tape device asynchronously. The **onpload** utility starts a similar thread for piped output. (UNIX only) If the disk file inputs, **onpload** uses the multithreading AIO subsystem instead of a dedicated **I/O** thread.
sdriver threads	The **worker** threads control **I/O** from input files. They handle device abstraction for the different device types handled. The **sdriver** threads also are responsible for passing out records from the input and passing records to the converters.
convert threads	The **onpload** utility starts one or more **convert** threads for each device. These threads perform conversions on the input data such as uppercase to lowercase conversion or code-set conversion.
worker threads	The **onpload** utility starts one **worker** thread for each input device. These threads communicate with the database server. The main responsibility of the **worker** thread is to pass data to the database server.

To see the status of the **onpload** threads, you must use the **-j** option of the **onstat** utility.

Related reference:
The onstat -j option

Threads that the database server uses

The database server uses the following threads to insert the data into the database:

Table 1. Threads that insert data into the database

Thread	Description
pl_wkr threads	Each **worker** thread of the **onpload** utility is paired with a **pl_wkr** thread in the database server. These threads receive the data from **onpload**. In a utility that shows the database server status, the **pl_wkr** threads are named **pl_wkr_1**, **pl_wkr_2**, **pl_wkr_3**, and so on.
cadiload threads	The **cadiload** threads are the **insert** threads. The **insert** threads perform a normal insert into the database, like during an INSERT statement. The SQL optimizer governs the method that is used for inserting the data.

The onpload utility express-mode load process

The **onpload** utility behaves the same way during express-mode deluxe-mode loads. However, the database server behaves differently during an express load.

During an express-mode load, the **pl_wkr** threads pass the data to **stream** threads (also called **fragmenter** threads) that decide where the data is to be stored. The **fragmenter** threads pass the data to an exchange that distributes the data to **setrw** threads. The **setrw** threads write table rows to disk a page at a time, bypassing the buffer cache.

The number of input devices can be different from the number of table fragments. The exchange operator handles multiplexing of data. The data is processed in parallel with respect to the data read from the device array and also with respect to the data written out to table fragments on separate disks. Pipeline parallelism is also present in the data flow from input devices to table fragments on disk. Parallelism is the main mechanism for achieving high performance.

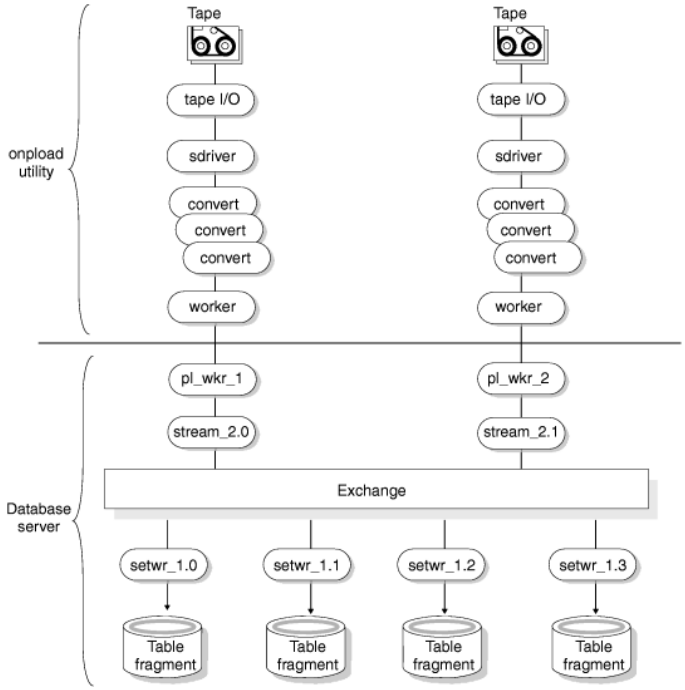
During express-mode load, the database server writes the data to new extents on disk, but those extents are not yet part of the table. At the end of an express-mode load, the database server adds the new extents to the table.

Important: After the express-mode load, you must perform a level-0 backup before you can write to the target database. If you try to write to the table before you perform a level-0 backup, the database server issues ISAM error -197, as follows:

```
Partition recently appended to; can't open for write or logging.
```

If your database is ANSI-compliant, all access (both read and write) is denied until you perform a level-0 backup. Because data is not logged in express mode, the level-0 backup is necessary to allow for recovery in case of media failure.

The following figure shows a single express-mode load process. In express mode, the data is inserted directly into an extent without any evaluation of objects such as constraints, indexes, or triggers.
Figure 1. An express-mode load

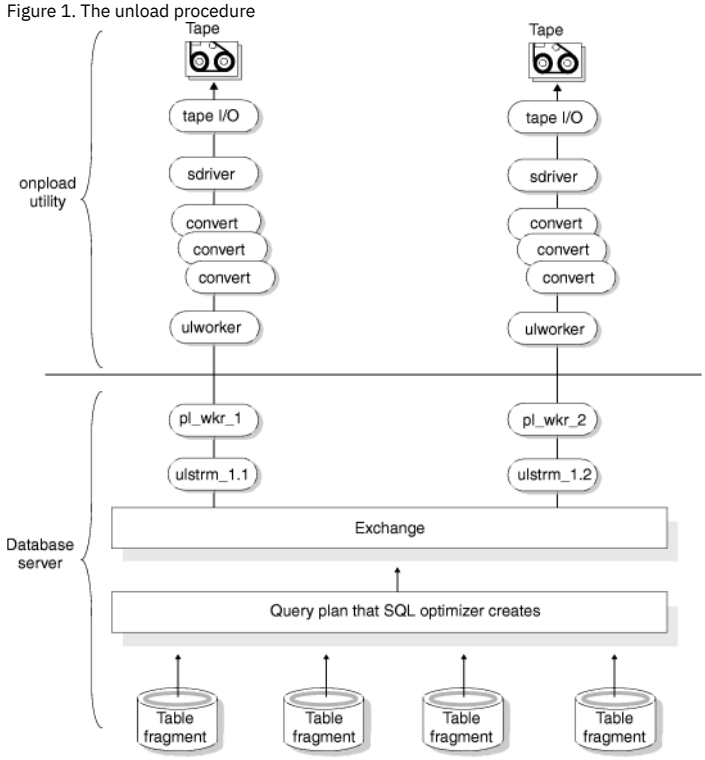


Related concepts:
The HPL express mode

The onpload utility unload process

The following figure shows the **onpload** unload process. In the unload process, the behavior of **onpload** parallels the behavior described in Threads that the onpload utility uses and Threads that the database server uses, except that the threads are unloading the data instead of loading it.

Figure 1. The unload procedure



The **ulstrm** (unload-stream) thread packages data for output to the **onpload** client from the query plan. The SQL optimizer creates the query plan. The query plan behaves like a query plan running from any other client, such as DB-Acess. The exchange operator distributes the resulting data to the **ulworker** threads in a round-robin fashion, and **onpload** unloads the data onto tapes or files.

Parallelism with respect to the output device, the source table fragments, and the flow of the data is evident in the preceding figure.

Examples of loading and unloading jobs using the ipload utility

This section shows how the components of the High-Performance Loader (HPL) fit together. The section includes a step-by-step tutorial with two examples (for a load and an unload job) that use the **ipload** graphical user interface (GUI).

The illustrations in the first example use a database with only one table. The table contains three columns. The data to be loaded into the database is in a file that has only four records. In a real production environment, you would probably use the INSERT statement, the **dbimport** utility, or the LOAD statement for such a simple operation. However, by using a simple example, the illustrations can show what happens at each step.

The **onpladm** command-line interface is equivalent to the **ipload** utility.

- Prepare to use the ipload utility
  The **ipload** utility is a part of UNIX database servers. Before you can use **ipload**, you must have installed the database server. If the database that you want to load or unload is on the computer where you plan to run **ipload**, you must also start the database server.
- Create a file of data
- Create a database
- The ipload utility
  The High-Performance Loader (HPL) uses information from the onpload database to control loading and unloading of data. Theoretically, you could create the onpload database and use DB-Acess or some other database tool to populate it. However, it is recommended that you always use **ipload** to manage the onpload database.
- Load Job windows
  A load job is a collection of the specific pieces of information that you require to move data from a data file into a database. The Load Job window shows a flowchart that includes all of the components of a load job.
- Device-Array windows
  A device array is a collection of files, tape devices, and pipes that **onpload** uses for input and output. Pipes are only supported on UNIX. You can create a device array and specify the location of the input data from the Device Array Selection and Device-Array Definition windows.
- Format windows
  The format specifies the organization of the input data.
- Filter, Discard Records, and Logfile text boxes
  The Load Job window now has entries for a device and format. The next incomplete items in the Load Job window are the Filter text box, the Discard Records text box, and the Logfile text box.
- Map Views window
  From the Map Views window, you can create a map that specifies which data field from the input file (/work/mydata) is entered into which columns in the database table.
- Completing the Load Record Maps window
  The Load Record Maps window specifies the device array that holds the input data, the format that describes the input data, and the database and table where the input data will be stored.
- Map-Definition window
- Load Options window
- Running the ipload job
  You are now ready to perform the load.
- Active Job window
- The ipload utility Generate options
  The **ipload** utility has Generate options that you can use to automatically create a format, map, query, and device. After the components are generated, you can modify the components to meet your needs.

Related reference:
The onpladm utility

Prepare to use the ipload utility

The **ipload** utility is a part of UNIX database servers. Before you can use **ipload**, you must have installed the database server. If the database that you want to load or unload is on the computer where you plan to run **ipload**, you must also start the database server.

You can also use **ipload** to manage load and unload operations for a database server on a different computer. For example, you can use **ipload** on a UNIX computer to manage the onpload database on a Windows computer.

Related reference:
Run load and unload jobs on a Windows computer

Create a file of data

The illustrations in this section assume that the data to be loaded is in a file named /work/mydata. Create a file that contains the following data:

```
1|a|50
2|b|25
3|c|10
4|d|5
```

Create a database

The High-Performance Loader (HPL) loads data into an existing table in an existing database. The examples in this section load the information from the file /work/mydata into a three-column table named **tab1** in a database named **testdb**. You can use DB-Acess to prepare the database and table, as follows:

```
CREATE DATABASE testdb;
CREATE TABLE tab1
(
```

```
col1 INTEGER,
col2 CHAR(1),
col3 INTEGER
);
GRANT ALL ON tab1 TO PUBLIC;
GRANT CONNECT TO PUBLIC;
```

After you finish preparing the database for the examples, exit from DB-Acess.

The ipload utility

The High-Performance Loader (HPL) uses information from the onpload database to control loading and unloading of data. Theoretically, you could create the onpload database and use DB-Acess or some other database tool to populate it. However, it is recommended that you always use **ipload** to manage the onpload database.

- Start the ipload utility
  To start **ipload**, type the following at the system prompt `ipload`.
- Choose a project
  You use the High-Performance Loader (HPL) by preparing load jobs that import data or unload jobs that export data. You can assign the load and unload jobs to various projects to organize the jobs into functional groups.
- Check the ipload utility default values
  The default values that **ipload** selects when it is first started specify machine type, character code set for character-type data, and other operating characteristics. In most cases, the only default that you might need to change is the machine type.

Related concepts:
The ipload utility

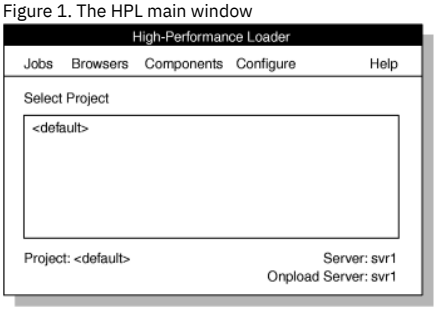
Start the ipload utility

To start **ipload**, type the following at the system prompt `ipload`.

A decorative splash screen appears and stays on the display while **ipload** finishes loading. If you do not want to see the splash screen, use the **-n** flag, `ipload -n`.

The first time you start **ipload**, it automatically creates the onpload database. The **ipload** utility also puts certain default values into the database.

When **ipload** starts, the High-Performance Loader (HPL) main window appears, as the following figure shows.
Figure 1. The HPL main window



Tip: To exit from **ipload**, choose **Exit** from the **Jobs** menu.
The **onpladm** command-line utility provides the same functionality as the **ipload** graphical user interface.

Related reference:
The onpladm utility
The onpload database

Choose a project

You use the High-Performance Loader (HPL) by preparing load jobs that import data or unload jobs that export data. You can assign the load and unload jobs to various projects to organize the jobs into functional groups.

The **ipload** utility automatically creates a project named **<default>**. If you choose not to organize your work into projects, you can put all of the load and unload jobs in the default project.

For this example, you can use the default project. Click <default> on the HPL main window to choose the default project.

Related concepts:
Define HPL projects

Check the ipload utility default values

The default values that **ipload** selects when it is first started specify machine type, character code set for character-type data, and other operating characteristics. In most cases, the only default that you might need to change is the machine type.

- Look at the Defaults window
  Choose Configure > Defaults to see the current default values. After you check the defaults, you can click Cancel to exit from the Defaults window.
- Look at the Machines window
  Choose Configure > Machines to see the current default values. Make sure that the machine type displayed in the Machines window matches the current machine type. Click the Machine > Type down arrow to select a machine type. Click Cancel to exit from the Machines window.

Related reference:
Configure the High-Performance Loader

Look at the Defaults window

Choose Configure > Defaults to see the current default values. After you check the defaults, you can click Cancel to exit from the Defaults window.

Related concepts:
Modify the onpload default values

Look at the Machines window

Choose Configure > Machines to see the current default values. Make sure that the machine type displayed in the Machines window matches the current machine type. Click the Machine > Type down arrow to select a machine type. Click Cancel to exit from the Machines window.

Related concepts:
Modify the machine description

Load Job windows

A load job is a collection of the specific pieces of information that you require to move data from a data file into a database. The Load Job window shows a flowchart that includes all of the components of a load job.

The Load Job window is illustrated in Figure 2. After you become familiar with **ipload**, you can create or modify the individual components of a load or unload job with direct menu choices from the HPL main window.

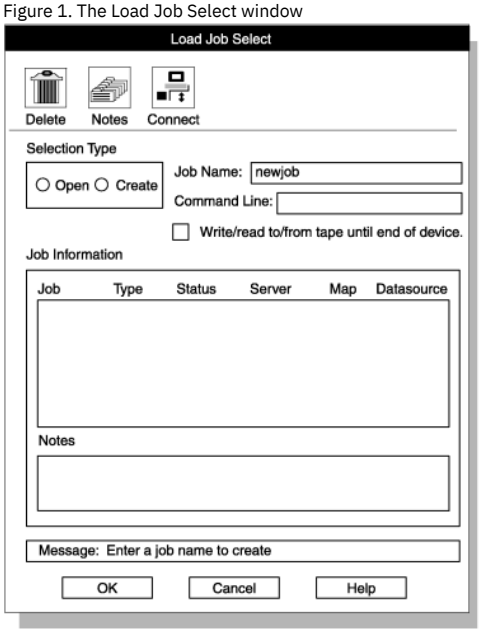
- Load Job Select window
  The data-load example takes records from /work/mydata and loads them into **tab1** of the **testdb** database. To perform the load, you need to create a load job.
- Load Job window

Load Job Select window

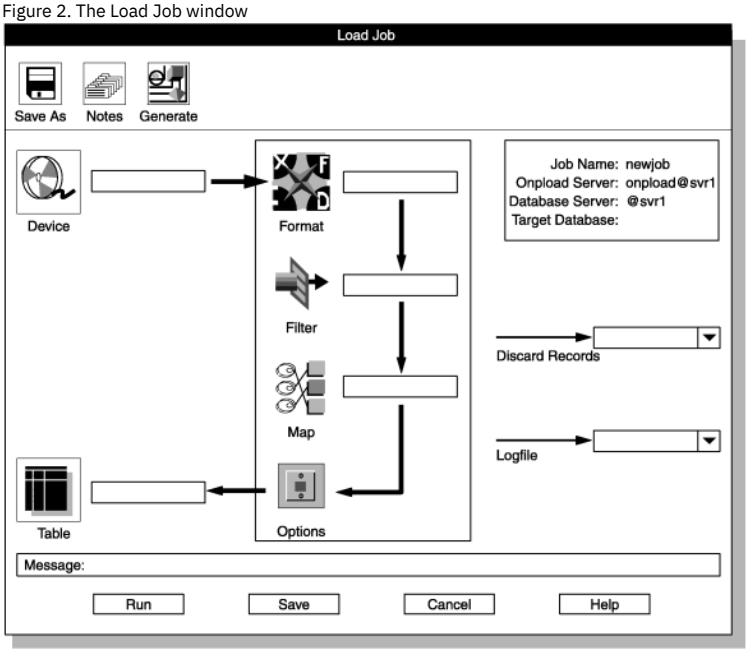
The data-load example takes records from /work/mydata and loads them into **tab1** of the **testdb** database. To perform the load, you need to create a load job.

To create a load job:

1. Choose Jobs > Load from the HPL window.
   The Load Job Select window appears, as the following figure shows.
   Figure 1. The Load Job Select window



2. Click Create in the Selection Type group.
3. Choose a name for the load job and type it in the Job Name text box. This example uses **newjob**.
4. Click OK.
   The Load Job window appears, as the following figure shows.
   Figure 2. The Load Job window



Load Job window

The thick arrows on the Load Job window indicate the steps that you take as you build a load job. The icons indicate the task at each step. The thin arrows indicate file names for error recording. To create a load job, you need to complete the following tasks:

Task	**Click**
Specify the source of the data	Device
Describe the data	Format
Tell **ipload** which data you want to discard (optional)	Filter
Specify association between input fields and load-table columns	Map
Specify options for the load job	Options
Specify the database table to load	Table
Record rejected data records (optional)	Discard Records
Record information about the job (optional)	Logfile

Device-Array windows

A device array is a collection of files, tape devices, and pipes that **onpload** uses for input and output. Pipes are only supported on UNIX. You can create a device array and specify the location of the input data from the Device Array Selection and Device-Array Definition windows.

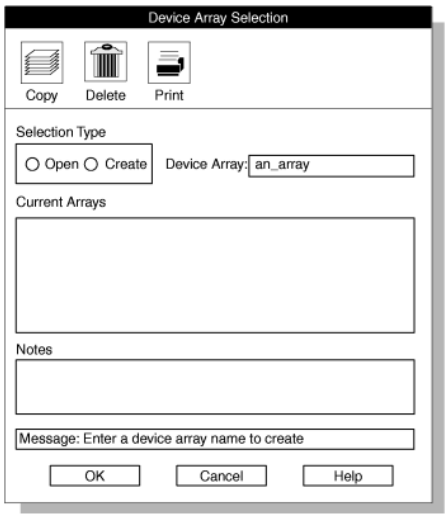
In this example, the input data is the /work/mydata file that you created in Create a file of data.

- Device Array Selection window
- Device-Array Definition window

Device Array Selection window

To create a device array:

1. Click Device in the Load Job window.
   The Device Array Selection window appears, as the following figure shows.
   Figure 1. The Device Array Selection window

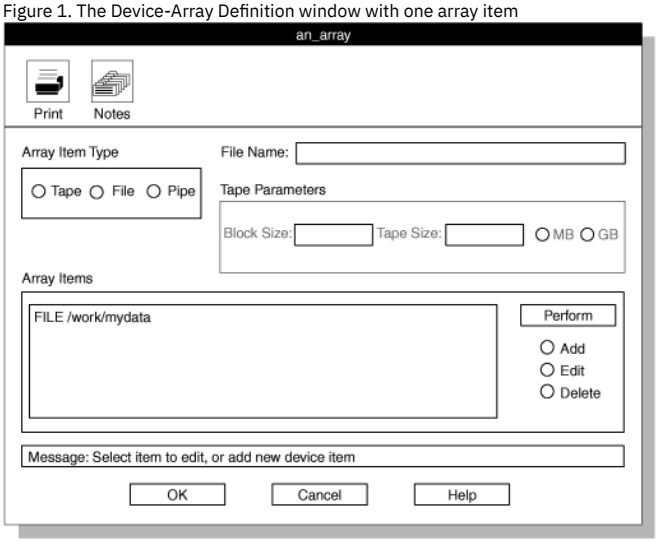


2. Click Create in the Selection Type group.
3. Select a name for the device array and type it in the Device Array text box. This example uses **an_array**.
4. Click OK.
   The Device-Array Definition window appears, as Figure 1 shows.

Device-Array Definition window

The title bar of the Device-Array Definition window shows the array name that you typed in the Device Array text box.

Figure 1. The Device-Array Definition window with one array item



To add a device to the array:

1. Click Add in the Perform group (lower right).
2. Click File in the Array Item Type group (upper left).
3. Type the full path name of a device in the File Name text box. In this example, the device is /work/mydata.
4. Click Perform to add the /work/mydata file to the device array.
   The **ipload** utility lists each item of the device array in the Array Items list box. Figure 1 shows the window after you add /work/mydata to the array.
5. Click OK.
   The display returns to the Load Job window. The Device list box now displays the device array name that you chose.

Format windows

The format specifies the organization of the input data.

In this example, the input data is in the file /work/mydata, which you created in Create a file of data. Each record in the file has three fields.

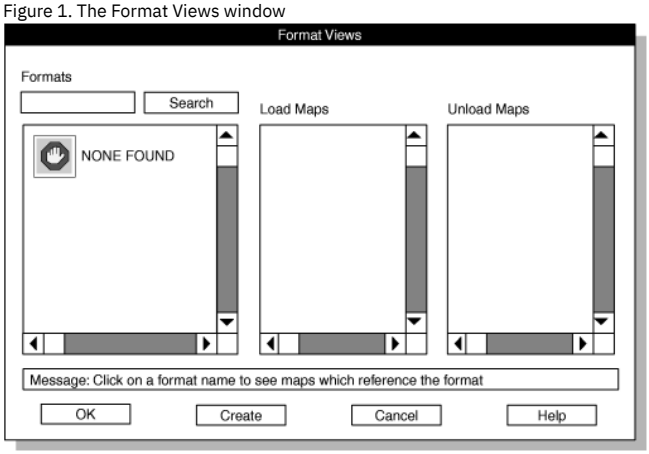
- Format Views window
  The Format Views window displays existing formats, so that you can choose the format to use in the load job.
- Record Formats window
- Format-Definition window
  In the Format-Definition window, you must make an entry for each field of the data records in the input file. The input file for this example (/work/mydata) has three fields of data in each record, so you must enter format information for three pieces of data.

Format Views window

The Format Views window displays existing formats, so that you can choose the format to use in the load job.

To open the Format Views window:

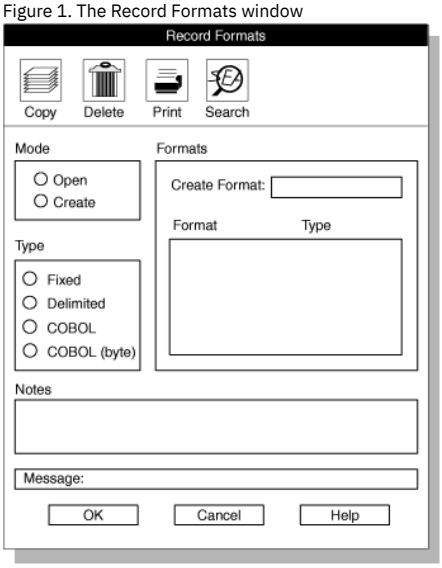
1. Click the Format in the Load Job window.
   The Format Views window appears, as the following figure shows. When you first start **ipload**, no formats are defined, as the NONE FOUND icon illustrates.
   Figure 1. The Format Views window



2. Click Create to open the Record Formats window.

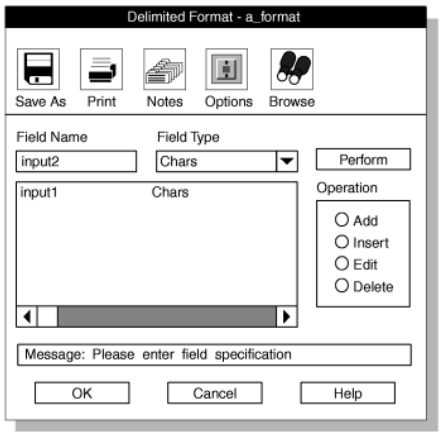
Record Formats window

The following figure shows the Record Formats window. You can create a format or open an existing format from the Record Formats window.
Figure 1. The Record Formats window



To create a format:

1. Click Create in the Mode group.
2. Click Delimited in the Type group.
   The input data file, /work/mydata, is in delimited format. Other formats are described in Define formats.
3. Choose a name for the format and type it in the Create Format text box. This example uses the name **a_format**.
4. Click OK.
   The Format-Definition window appears. The following figure shows a partially completed Format-Definition window. The title bar of the Format-Definition window shows the name that you chose for the new format.
   Figure 2. The Format-Definition window



Format-Definition window

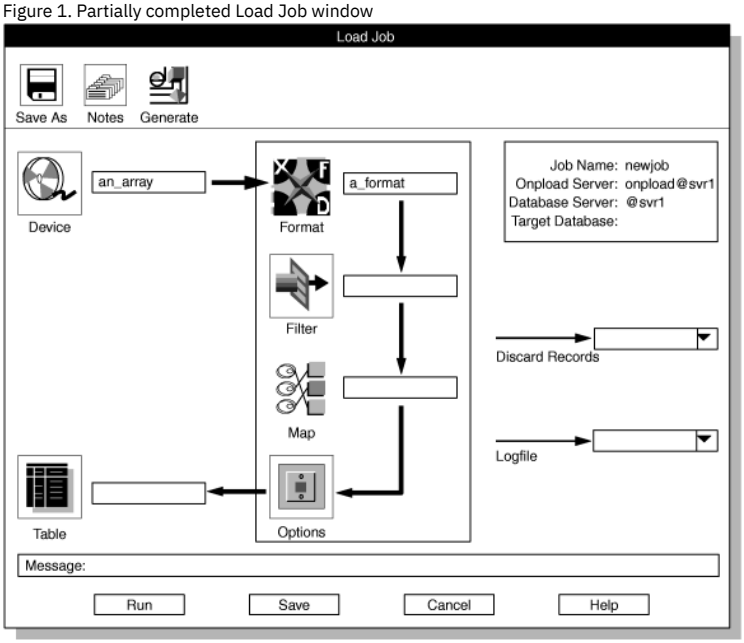
In the Format-Definition window, you must make an entry for each field of the data records in the input file. The input file for this example (/work/mydata) has three fields of data in each record, so you must enter format information for three pieces of data.

To enter a format definition:

1. Click Add in the Operation group.
2. In the Field Name text box, type a descriptive name for the first field of the data record.
   You can choose any descriptive name. This example uses **input1**, **input2**, and **input3** for the three fields of /work/mydata.
3. In the Field Type text box, type the data type or click the down arrow for a list of selections.
   Because the data in /work/mydata is simple ASCII data, the type is **Chars**. Other data types are discussed in Define formats.
4. Click Perform.
   Figure 2 shows the partially completed Format-Definition window. The entry for the first item is complete. The Field Name and Field Type for the second item are present and ready for you to click Perform.
5. Repeat steps 2 through 4 for each of the three input fields.
6. Click OK after you complete all of the input fields.
   The display returns to the Format-Views window, which displays the new format in the Formats list box.
7. Click Cancel to return to the Load Job window.

The Load Job window now displays the name of the device and the name of the format, as the following figure shows.

Figure 1. Partially completed Load Job window



Filter, Discard Records, and Logfile text boxes

The Load Job window now has entries for a device and format. The next incomplete items in the Load Job window are the Filter text box, the Discard Records text box, and the Logfile text box.

Filter text box

Use a filter to choose the records from the data file that should be inserted into the table. In this example, all of the records from the /work/mydata data file are inserted into the database table. Therefore, you do not need to create a filter. For this example, you can leave the Filter text box blank.

Define filters, describes how to create and use a filter.

Discard Records text box

The Discard Records text box specifies a file that keeps information about records that were rejected because of incorrect format or invalid data. For this example, you can leave the Discard Records text box blank.

Reviewing records that the conversion rejected describes how to view rejected records.

Logfile text box

The Logfile text box specifies a file where a record of the load or unload job is kept. For this example, you can leave the Logfile text box blank.

View the status of a load job or unload job describes how to view the log file.

Map Views window

From the Map Views window, you can create a map that specifies which data field from the input file (/work/mydata) is entered into which columns in the database table.

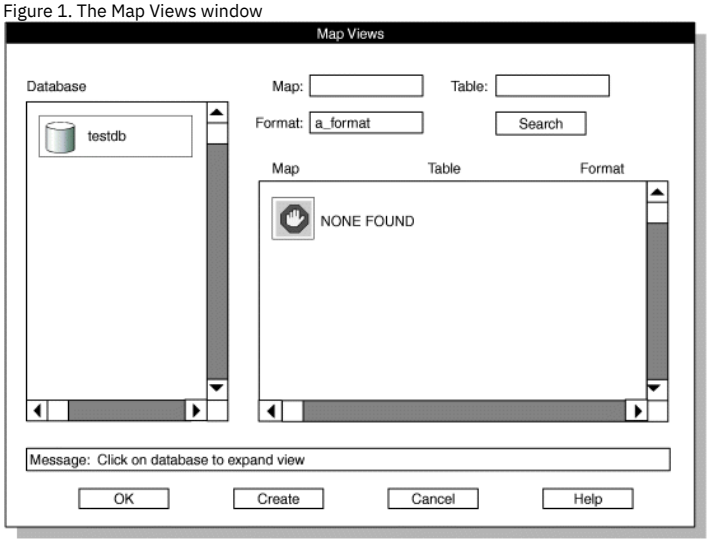
- Creating a map by using Map Views window
  Use the Map Views window to create a map or edit an existing map. You need to create a map that shows how the input data that the record format **a_format** describes is to be loaded into the table **tab1**.

Creating a map by using Map Views window

Use the Map Views window to create a map or edit an existing map. You need to create a map that shows how the input data that the record format **a_format** describes is to be loaded into the table **tab1**.

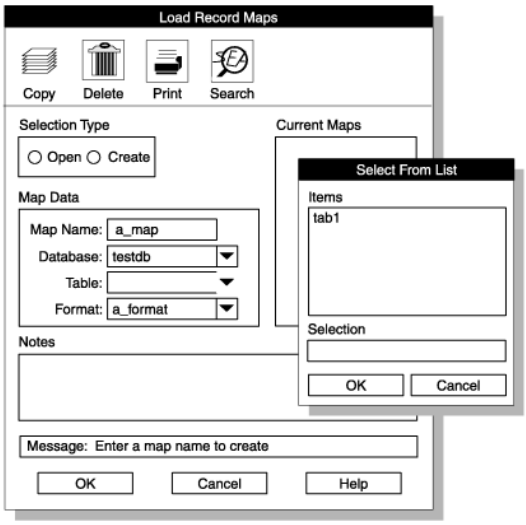
To create a map:

1. Click the Map button in the Load Job window.
   The Map Views window appears, as the following figure shows.
   Figure 1. The Map Views window



   The NONE FOUND icon in the Map column is appropriate. The Map Views window does not contain any information because you have not yet specified any relationships.

2. Click Create.
   The Load Record Maps window appears, as the following figure shows.
   Figure 2. A partially completed Load Record Maps window with an open selection list



Completing the Load Record Maps window

The Load Record Maps window specifies the device array that holds the input data, the format that describes the input data, and the database and table where the input data will be stored.

To complete the Load Record Maps window:

1. Click Create in the Selection Type group.
2. Select a name for the map and type it in the Map Name text box. This example uses **a_map**.
3. Type the name of your database (`testdb`) in the Database text box. Or, click the down arrow to select a database from the selection list.
4. Click the down arrow beside the Table text box to see a list of tables in the selected database.
   Figure 2 shows the Load Record Maps window and the selection list.
5. Select a table from the list and click OK.
   Because you already filled in the Format text box on the Load Jobs window, the Format text box is already complete.
6. Click OK to open the Map-Definition window.

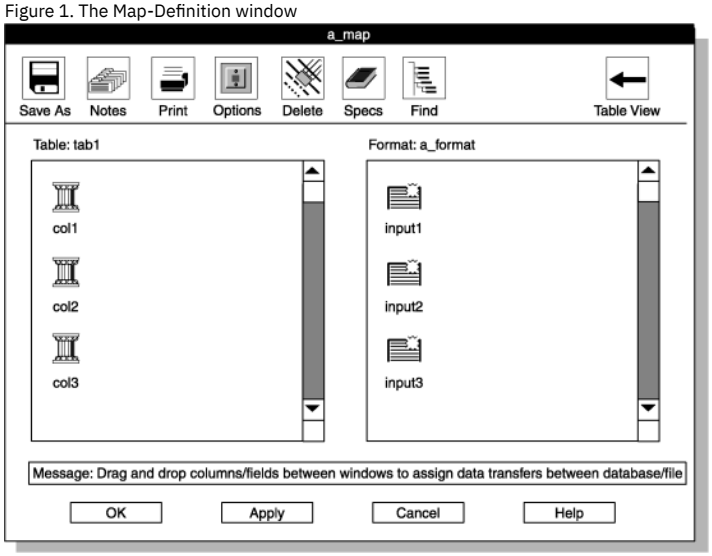
Map-Definition window

You can associate an input item with a table column in the Map-Definition window. This example stores the data from /work/mydata as follows, by using the field names assigned in step 1.

Data from input field	Goes into table column
input1	col3
input2	col2
input3	col1

The following figure shows the Map-Definition window. The title bar of this window shows the map name that you chose.

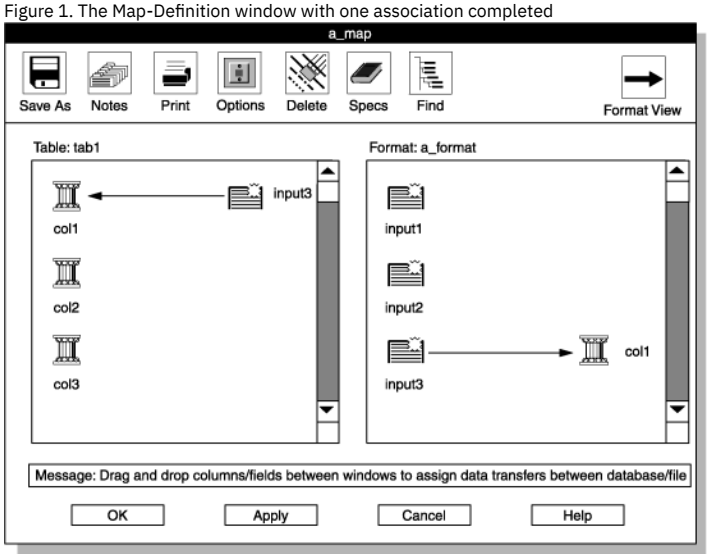
Figure 1. The Map-Definition window



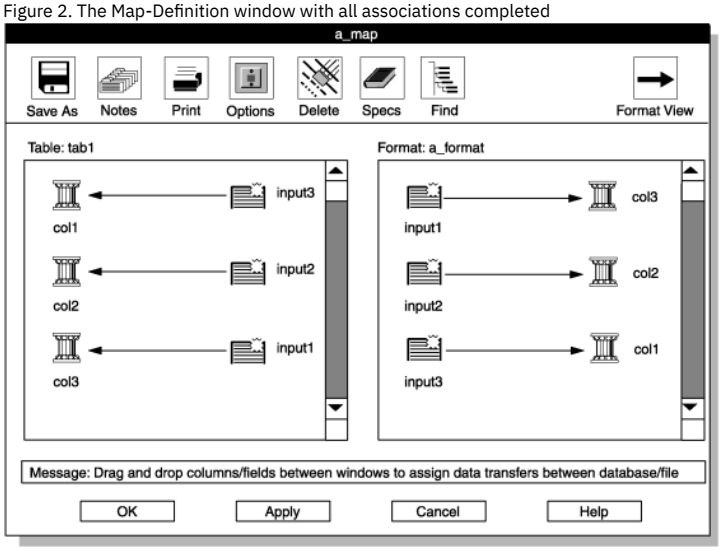
Associating each input item with a column of the database table

To associate each input item with a column of the database table:

1. Click the col1 icon and hold the mouse button down. A box appears around the icon and its name.
2. Drag the boxed icon to the input3 icon in the right pane.
3. Release the mouse button.
   The associated items appear in the second column of each pane. The following figure shows the Map-Definition window with this step completed.
   Figure 1. The Map-Definition window with one association completed

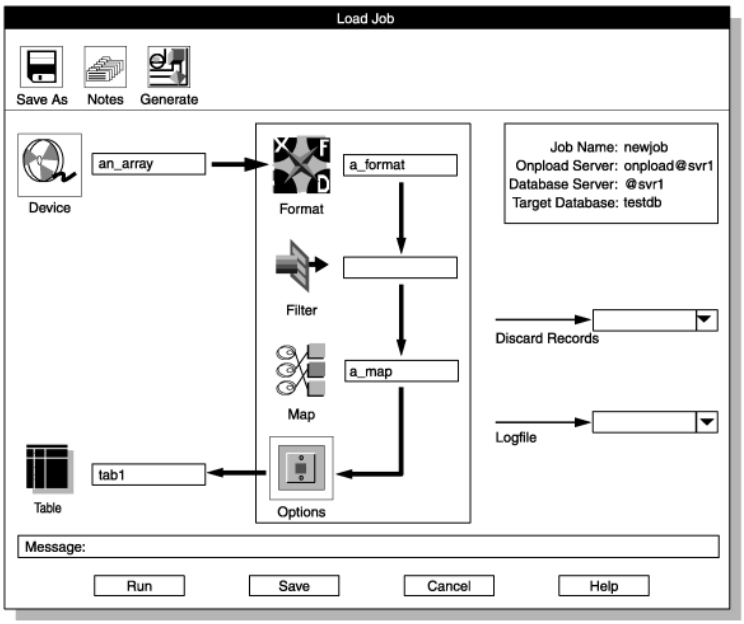


4. Connect **col2** to **input2**.
5. Connect **col3** to **input1**.
   The following figure shows the window with all three connections completed.
   Figure 2. The Map-Definition window with all associations completed



6. Click OK to return to the Map Views window.
7. Click Cancel to return to the Load Job window.

The Load Job window now has entries in all of the required areas, as the following figure shows. The **ipload** utility was able to enter the table name and the target database name (upper right area) because you specified the database and table as you built the map.
Figure 3. The Load Job window with all required component boxes completed

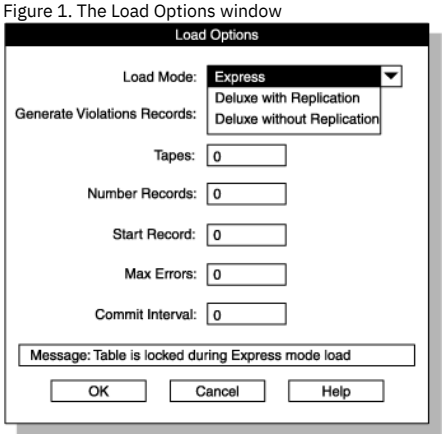


You have finished all of the required parts of the Load Job window, but you might want to modify the options, as discussed in the next section.

Load Options window

The High-Performance Loader (HPL) has three modes of operation: express, deluxe with replication, and deluxe without replication. The express mode is optimized for speed and the deluxe mode provides the full functionality of SQL inserts as data is loaded. For a detailed comparison of these modes, see Manage the High-Performance Loader. This example uses the express mode.

To set the load-job options:

1. Click Options in the Load Job window.
   The Load Options window appears, as the following figure shows.
   Figure 1. The Load Options window
   

2. Select Express from the Load Mode list box.
3. Select Yes from the Generate Violations Records list box.
4. Make sure all of the other entries are `0`.
5. Click OK to return to the Load Job window.

Running the ipload job

You are now ready to perform the load.

To finish this example:

1. Click Save to save the load job.
   After you save the load job, the message line displays the following message:

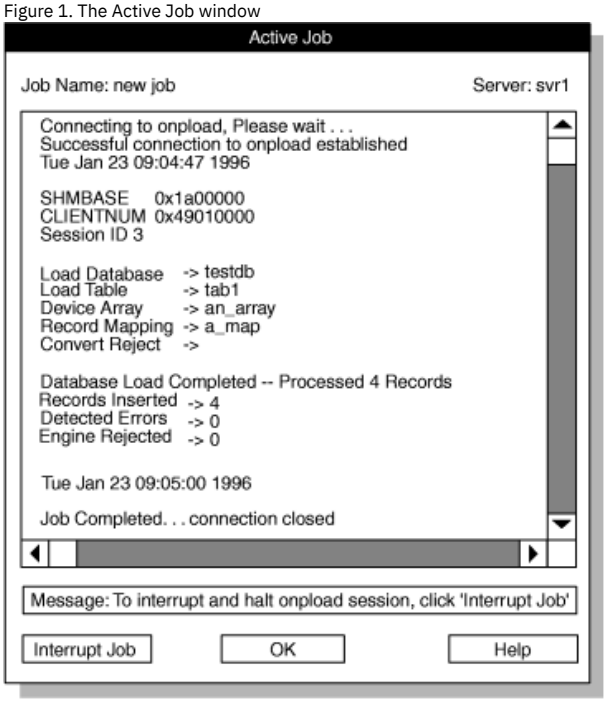
   ```
   Saved job successfully
   ```

   You can now run the job, or you can return at a later time and run the job.
2. Click Run to run the load job.
   The Active Job window appears, as Figure 1 shows.

Tip: The **ipload** utility generates an **onpload** command and then runs the command to run your job. To see the command that **ipload** generates, look at the Command Line text box on the Load Job Select window. For more information, see The command-line information.

Active Job window

The Active Job window reports the progress of your job. The following figure shows the Active Job window after the load is complete.
Figure 1. The Active Job window



When the Active Job window reports that the load job is complete, click OK to return to the Load Job Select window.

Verify the transfer of data

You can use DB-Acess to verify that the data from /work/mydata was transferred into your database.

Perform a level-0 backup

The **onpload** utility does not log the data that it writes to a table during an express-mode load. For safety, **onpload** flags the dbspaces that are associated with the table as read-only. To allow for data recovery in case of disk corruption, you must make a level-0 backup. A level-0 backup on the dbspaces affected by the express-mode load saves the data and unsets the read-only flags.

If you do not care about data recovery, you can make a level-0 backup by using /dev/null as the backup device. This action unsets the read-only flag without backing up data to any real device.

The ipload utility Generate options

The **ipload** utility has Generate options that you can use to automatically create a format, map, query, and device. After the components are generated, you can modify the components to meet your needs.

This example uses the Generate button in the Unload Job window to create the components that are required for an unload job. After you create the components, you can use the Run option to run the unload job.

- Use the information you created with the ipload example
- Preparing the Unload Job window
  An unload job is a collection of the specific pieces of information that are required to move data from a database into a data file. The Unload Job window shows a flowchart that includes all of the components of an unload job. You can use the Generate option to create the components of the unload job and to complete the items on the Unload Job window.
- Performing the unload job
  You are now ready to perform the unload job.

Related reference:
The Generate options of the ipload utility

Use the information you created with the ipload example

If you completed the first example, your database server and **ipload** are ready for you to use. If you did not complete the example, you need to complete the following tasks as the first example describes:

- Start your database server (Prepare to use the ipload utility)
- Start **ipload** (Start the ipload utility)
- Check the defaults (Check the ipload utility default values)

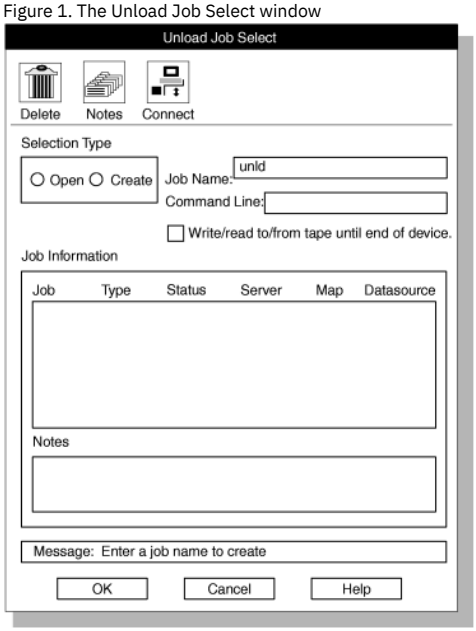
Preparing the Unload Job window

An unload job is a collection of the specific pieces of information that are required to move data from a database into a data file. The Unload Job window shows a flowchart that includes all of the components of an unload job. You can use the Generate option to create the components of the unload job and to complete the items on the Unload Job window.

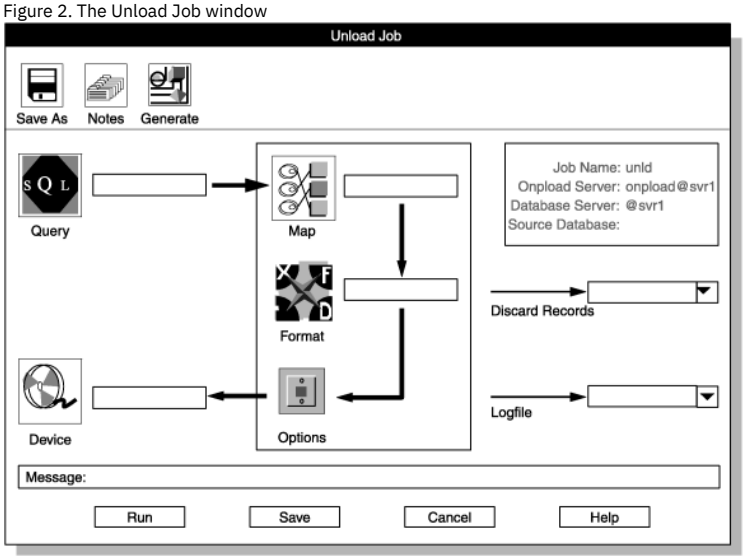
The generate example uses the Generate option to unload the contents of the **items** table of the **stores_demo** database into a file named /work/items_out. For instructions on how to create the **stores_demo** database and other demonstration databases, see the *IBM® Informix® DB-Access User's Guide*.

To generate the unload job:

1. Choose Jobs > Unload from the HPL window.
   The Unload Job Select window appears, as the following figure shows.
   Figure 1. The Unload Job Select window



2. Click Create in the Selection Type group.
3. Choose a name for the unload job and type it in the Job Name text box. This example uses the name **unld**.
4. Click OK.
   The Unload Job window appears, as the following figure shows. The information box in the upper right part of the display shows the name of the unload job, the name of the database server where the onpload database is stored, and the name of the database server where **ipload** is running.
   Figure 2. The Unload Job window



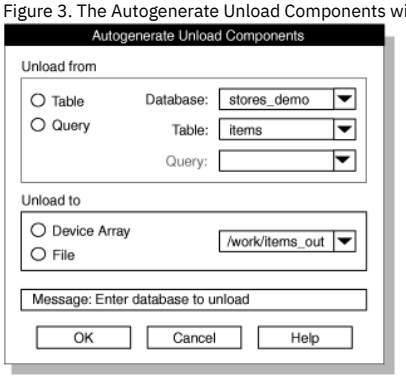
5. Click the Generate button.
   The Autogenerate Unload Components window appears. Figure 3 shows the completed window.

6. Click Table.

You can unload an entire database table or only selected records from the table. Table indicates that you want to unload the entire table. Query indicates that you want to unload selected records.

7. Type `stores_demo` in the Database text box.
   For this step and steps 8 and 10, you can click the down arrow to the right of the text box and select the entry from a selection list. Figure 2 shows an example of a selection list.

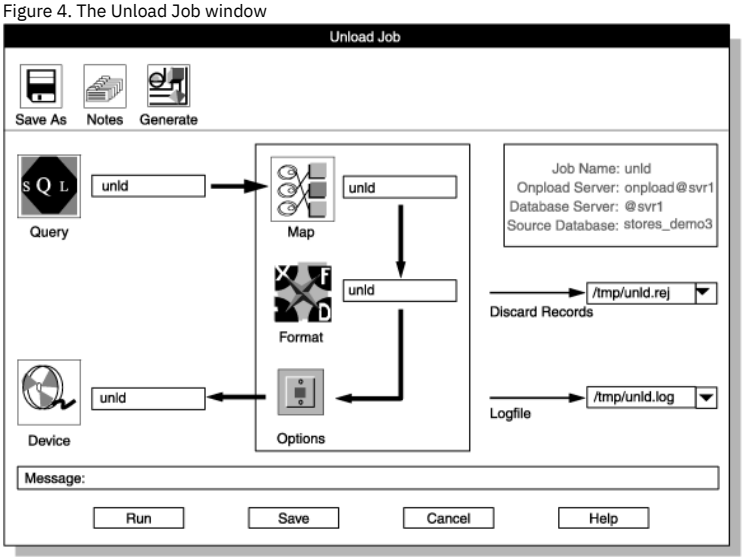
8. Type `items` in the Table text box.
   Figure 3. The Autogenerate Unload Components window



9. Click File.
   File indicates that you want to type the name of a file. If you choose Device Array, you must type the name of an existing device array.

10. Type the full path name of the file that will store the unloaded data. This file can be in any directory to which you have write access.
11. Click OK.

The Generate option creates the Query, Format, and Map components for the unload job and completes the Unload Job window. These components are all named **unld**.
The Generate option also creates a device array named **unld** and puts the file that you specified (/work/items_out) into that array.
Tip: After you finish this exercise, you can choose Components > Devices from the HPL window and examine the **unld** device array.
The following figure shows the Unload Job window as completed by the Generate option.
Figure 4. The Unload Job window



In addition to completing the main flow of the Unload Job window, the Generate option also enters the Source Database information in the upper right corner and creates path names for the Discard Records file and the Logfile. The HPL browsing options, describes the rejected records file and the log file.

Performing the unload job

You are now ready to perform the unload job.

Tip: You might want to preview the records that the query will select. See Preview data-file records.
To finish this example:

1. Click Save to save the unload job.
   After you save the unload job, the Message line displays the following message:

   ```
   Saved job successfully
   ```

   You can now run the job, or you can return at a later time and run the job.

2. Click Run to run the unload job.
   The Active Job window appears. This window reports the progress of your unload job. The Active Job window for an unload job is similar to the Active Job window for a load job, which Figure 1 shows.

3. When the Active Job window reports that the unload job is finished, click OK to return to the Unload Job window.
4. Click Cancel to return to the HPL main window.

5. You can create another job or choose Jobs > Exit to exit **ipload**.

The ipload utility windows

This section contains information about using the **ipload** utility.

- The ipload utility GUI or the onpladm command-line interface
  The **ipload** utility is a graphical user interface (GUI) that contains windows, buttons, online help, and keyboard commands. The **onpladm** command-line interface is equivalent to the **ipload** utility.
- The ipload utility GUI
- The HPL ipload utility buttons
  After you move beyond the main window, every window has at least one button to help you move through the interface.
- The HPL online help
  The Help menu on the High-Performance Loader (HPL) main window has Glossary and Contents choices.
- The UNIX keyboard commands to move the cursor
  Instead of using the mouse to move from area to area in the High-Performance Loader (HPL) user interface, you can use UNIX keyboard commands to move the cursor. As you move around, the currently selected item is highlighted with a box.

The ipload utility GUI or the onpladm command-line interface

The **ipload** utility is a graphical user interface (GUI) that contains windows, buttons, online help, and keyboard commands. The **onpladm** command-line interface is equivalent to the **ipload** utility.

- Start the ipload utility
  To start **ipload**, type `ipload` at the system prompt.

Related reference:
The onpladm utility

Start the ipload utility

To start **ipload**, type `ipload` at the system prompt.

A splash screen appears and stays on the display while **ipload** finishes loading. If you do not want to see the splash screen, use the **-n** flag with your command, `ipload -n`.

Related concepts:
The ipload utility

The ipload utility GUI

The **ipload** utility has the following types of displays:

- HPL main window
- Component-Selection windows
- Component-Definition windows
- Load Job and Unload Job windows
- Views windows
- Selection-List windows
- Message windows

Important: While the **onpload** and **onpladm** utilities include support for object names that contain up to 128 characters, the **ipload** utility does not. If you use long database, table or column names and create jobs by using **onpladm**, you cannot run these jobs by using **ipload**. For **ipload**, database, table, and column names cannot exceed 18 characters.

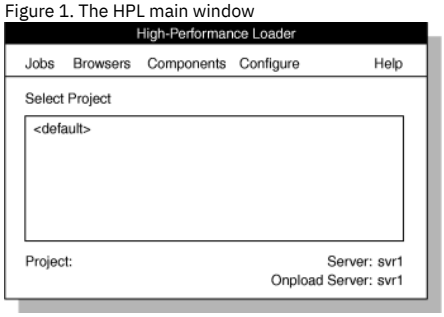
- The HPL main window
  When you start **ipload**, the High-Performance Loader (HPL) main window appears. The HPL main window is the focus of the user interface. You return to the main window after each task and choose a new option.
- Component-Selection windows
  The windows for creating or modifying components often (but not always) come in pairs. In the first window, the Component-Selection window, you can create a component or select an existing component to modify. You can also view notes and copy, delete, or print information about a component. In the second window, the Component-Definition window, you can make changes.
- Component-Definition windows
  After you select a component to create or modify and click OK in the Component-Selection window, you typically see the Component-Definition window. From the Component-Definition window, you can enter, edit, or delete values or items that describe the component.
- Load Job and Unload Job windows
  The Load Job and Unload Job windows provide a visual presentation of the basic components that you choose for each job.
- Views windows
  A Views window shows a graphic representation of the relationships among various **ipload** components. From a Views window, you can search for specific components, select an existing component for editing, or create a component.

- Selection-List windows
  A Selection-List window lists the possible values for a text box. A down arrow that follows a text box indicates that you can use a selection list to see and select possible values for the text box. When you click the down arrow, the corresponding Selection-List window appears.
- Message windows
  A message window typically contains either a warning or an information update. A warning lets you verify or cancel the action that you have chosen. An update informs you about the successful completion of an operation or explains why an operation failed.

The HPL main window

When you start **ipload**, the High-Performance Loader (HPL) main window appears. The HPL main window is the focus of the user interface. You return to the main window after each task and choose a new option.

Figure 1. The HPL main window



- Initial options on the HPL main window
- Options of the HPL main window
  After you select a project, you can choose options from any of the menus on the HPL main window.

Initial options on the HPL main window

When you first enter **ipload**, you can:

- Select the default project in the Select Project list.
- Choose Configure > Project to create a project.
- Choose Configure > Server to select a database server and an onpload database server.
- Choose **Help** to look at the online help.
- Choose Jobs > Exit to exit from **ipload**.

Related concepts:
Select a database server
Related tasks:
Defining a new project

Options of the HPL main window

After you select a project, you can choose options from any of the menus on the HPL main window.

The following table shows the options of the main menu:

Table 1. HPL main menu and submenu options

Main menu option	Submenu option	Purpose	See
Jobs	Load	Create a load job and use the Load Job window to load data into a database.	Load data to a database table
Jobs	Unload	Create an unload job and use the Unload Job window to unload data from a database to a file.	Creating an unload job
Jobs	Exit	Exit from the user interface.	The ipload utility
Browsers	Record	Review records in a specified format, search the list of available formats, or edit a format.	Reviewing records that the conversion rejected
Browsers	Violations	View records that passed the filter and conversion but were rejected by the database.	Viewing the violations table
Browsers	Logfile	View load status and see where any errors occurred.	View the status of a load job or unload job
Components	Formats	Create or modify data-file formats.	Define formats
Components	Maps	Create or modify maps that show the relationship between data-file fields and database columns.	Define maps
Components	Query	Build, modify, or retrieve SQL-based queries.	HPL queries

Main menu option	Submenu option	Purpose	See
Components	Filter	Create or modify filters that determine source data-file records for conversion and load.	Define filters
Components	Devices	Specify a set of files, tapes, or pipes (UNIX only) that will be read simultaneously for loading or unloading the database.	Define device arrays
Components	Generate Job	Automatically generate the components for load and unload jobs.	The Generate options of the ipload utility
Configure	Server	Select the database servers that hold the onpload database and the target database.	Configure the High-Performance Loader
Configure	Project	Create a project under which formats, filters, queries, maps, and load and unload jobs are stored.	Define HPL projects
Configure	Defaults	Specify the default character sets for the data file and databases.	Modify the onpload default values
Configure	Machines	Specify the machine parameters that are used to convert binary data.	Modify the machine description
Help	Glossary	View definitions of terms that pertain to the HPL.	The HPL online help
Help	Contents	View the main contents page that directs you to discussions of various HPL topics.	The HPL online help

Component-Selection windows

The windows for creating or modifying components often (but not always) come in pairs. In the first window, the Component-Selection window, you can create a component or select an existing component to modify. You can also view notes and copy, delete, or print information about a component. In the second window, the Component-Definition window, you can make changes.

The details of a selection window vary depending on the operation that you are performing.

The following figure shows the Device Array Selection window to illustrate the standard features of Component-Selection windows.
Figure 1. The Device Array Selection window

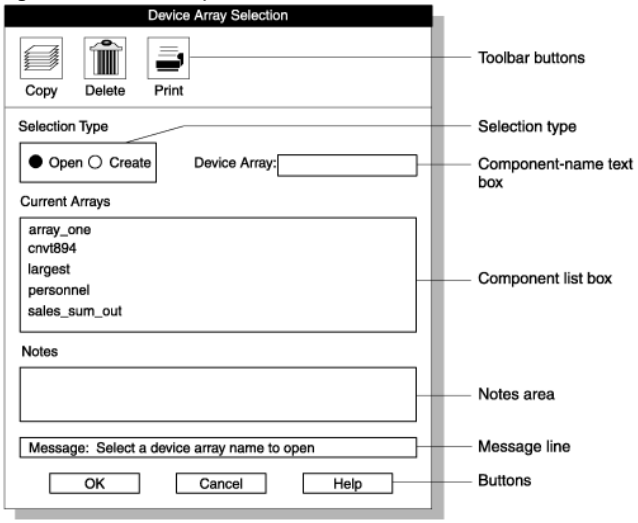


Table 1. The Device Array Selection window options

Display option	Description
Toolbar buttons	The buttons across the top of the display represent actions that you can take after you select a component from the component list. For example, in Figure 1, the toolbar buttons indicate that you can copy, delete, or print an array. The HPL ipload utility buttons explains how to use these buttons.
Selection type	The selection type allows you to specify the action that you want to take. In most of the displays, you can either open an existing component or create a component.
Component-Name text box	If you click Create, you must type a name for the new component in the Device-Array text box. (In Figure 1, you must give a name for the new device array.)
Device Array text box	Before you can type a name in the Device Array text box, you must click inside that text box to activate it. When the text box is active, it has a narrow black border. If you type a character that is not valid, the interface beeps at you, displays a message on the message line, and refuses to display the invalid character.
Component list box	The component list box lists the components that currently exist in this project. If you click Open in the selection group, you must select a component from this list.
Notes area	The notes area displays stored comments about the selected component. This area is not an active area. To store a comment about a component, you must select a component and use the Notes button. For more information about notes, see The Notes button.
Message Line	The message line primarily gives instructions for the next logical action. The message line also gives an error message when an action fails or a completion message when a process is finished.
Buttons	The buttons across the bottom of the display let you indicate your next action. For a more complete discussion, see The HPL ipload utility buttons.

Component-Definition windows

After you select a component to create or modify and click OK in the Component-Selection window, you typically see the Component-Definition window. From the Component-Definition window, you can enter, edit, or delete values or items that describe the component.

The following figure shows an example of a Component-Definition window: the Device-Array Definition window.
Figure 1. The Device-Array Definition window

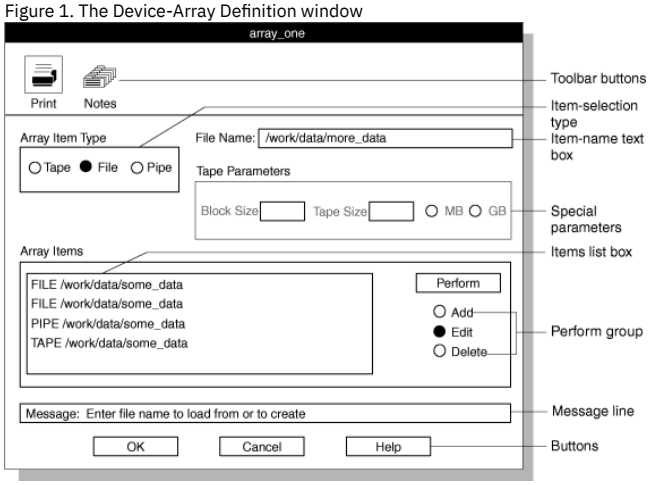


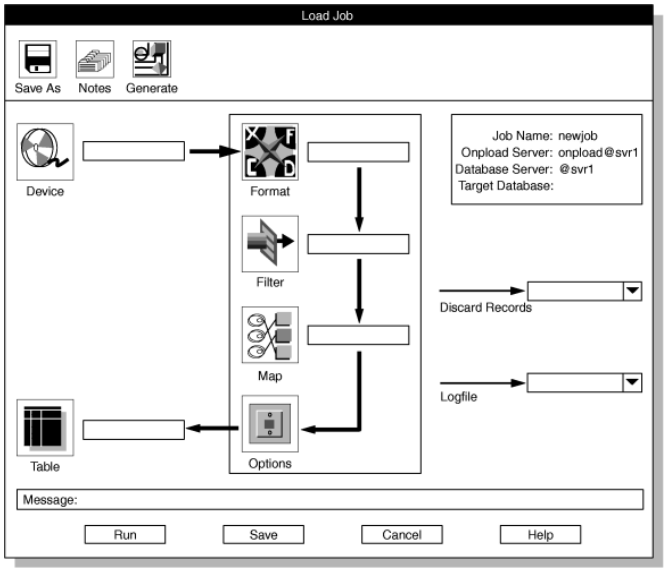
Table 1. The Device-Array Definition window options

Display option	Description
Toolbar buttons	The buttons across the top of the display represent actions that you can take after you select a component from the component list box. For example, in Figure 1 (the Device-Array Definition window), the toolbar buttons indicate that you can print or make a note about an item. The HPL ipload utility buttons explains how to use these buttons.
Item-selection group	The item-selection group lets you specify the type of item that you want to edit or the type of action that you want to take. After you specify a choice in the item-selection group, other options become active. In the Device-Array Definition window, the item-selection group is labeled Array Item Type. After you select Tape, File, or Pipe (only on UNIX), other options become active.
Item-name text box	The item-name text box lets you specify the name or description of one of the items that makes up the component. For example, in the Device-Array Definition window, you type the full path name of a device in the item-name text box. In the Device-Array Definition window, the item-name text box is labeled Tape Name, File Name, or Pipe Name (only on UNIX), depending on the type of component that you select from the Array Item Type group.
Special-parameters group	When a Component-Definition window first appears, some of the choices are inactive (shown in gray letters). In general, the inactive choices are not meaningful until you specify some other characteristic of the component that you are editing. The special-parameters group in the Device-Array Definition window is the Tape Parameters group. The items in the special-parameters group are meaningful only for tapes. The choices in the **Tape Parameters** group become active only if you select **Tape** from the **Array Item Type** group. The choices in Figure 1 are gray because File is selected in the Array Item Type group.
Item list box	The item list box shows items that you already created to define the component. In the Device-Array Definition window, this list is labeled Array Items and shows the tapes, files, and pipes (UNIX only) that are already part of the current device array.
Perform group	The Perform group lets you specify the action that you want to take. After you select an item and an action, you must click Perform to complete the action. For example, to add a new device in the Device-Array Definition window, you must specify the name or description of the device and then click Perform to add it to the Array List. Important: Remember to click Perform to complete the action that you designated in the Perform group.
Message line	The message line primarily gives instructions for the next logical action. The message line also returns an error message when an action fails or a completion message when a process is finished.
Buttons	The buttons across the bottom of the display let you indicate your next action. For a more complete discussion, see The HPL ipload utility buttons.

Load Job and Unload Job windows

The Load Job and Unload Job windows provide a visual presentation of the basic components that you choose for each job.

The following figure shows the Load Job window.
Figure 1. The Load Job window



Related reference:
Unload data from a database
Load data to a database table

Views windows

A Views window shows a graphic representation of the relationships among various **ipload** components. From a Views window, you can search for specific components, select an existing component for editing, or create a component.

A Views window does not allow you to change any values. To change the values of a component, you must display a Component-Definition window.

- Access views windows
- Available options in a Views window

Access views windows

The following table lists the four types of views windows and gives instructions for how to access each view.

Window name	Purpose	How to access	See
Format Views	Shows the load and unload maps that are associated with a particular format	• Click Search in the Record Formats window • Click Format in the Load Job window* • Click Format in the Unload Job window*	Figure 1
Map Views	Shows the databases, tables, queries, and formats that are associated with a map	• Click Search in the Load Record Maps window • Click Search in the Unload Record Maps window • Click Map in the Load Job window* • Click Map in the Unload Job window*	Figure 1
Database Views	Shows the tables in the database or the queries that are associated with the database	• Click Search in the Query window • Click Table in the Load Job window* • Click Query in the Unload Job window*	Figure 1
Filter Views	Shows the formats that are associated with a particular filter	• Click Search in the Filter window • Click Filter in the Load Job window*	Filter views

* These options display the Views window only if the corresponding text box is empty. If the text box includes the name of a component, the Component-Definition window is displayed.

Available options in a Views window

The four types of Views windows operate in a similar manner. When a Views window appears, you have the following options:

- Type in a component name and search for the component.
- Click a label associated with an icon to expand the view and see related components.
- Click an icon to open the Component-Definition window that allows you to edit the component values.
- Click Create to display the Component-Selection window that allows you to create a component.

- Search for a component in a Views window
  You can use the Search button in a Views window to locate a specific component.

- Expand the view in a Views window
  Three of the Views windows expand their views to show related components.

Search for a component in a Views window

You can use the Search button in a Views window to locate a specific component.

Type the component name in the search text box and then click Search. The view displays only the component names that match the text string.

You can use the following wildcard search characters in the search text string.

Table 1. Wildcard search characters

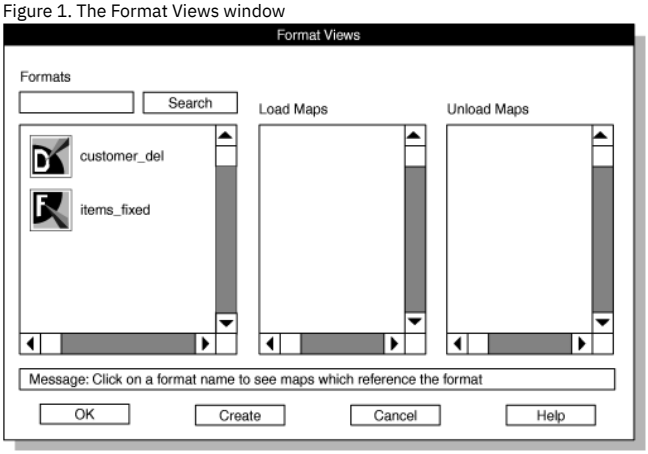
Wildcard symbol	Effect
?	Matches any single character
*	Matches any string of characters

Expand the view in a Views window

Three of the Views windows expand their views to show related components.

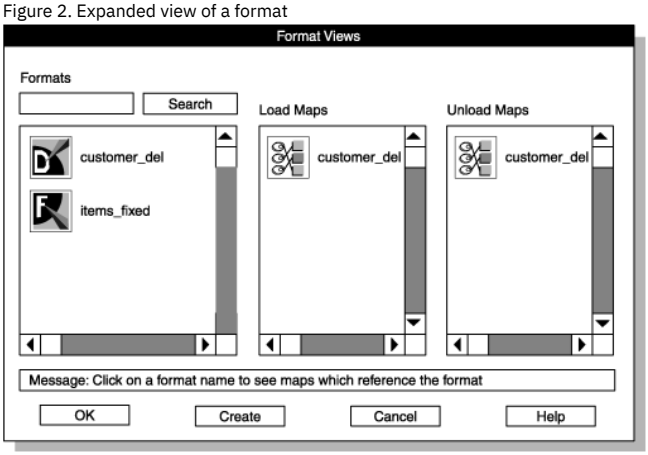
To expand the view, click an icon label (for example, **customer_del**) in the first pane. In the Database Views window, click an icon label in the second pane to expand the view further. The following figure shows the Format Views window.

Figure 1. The Format Views window



When you click an icon label in the **Formats** pane, the view expands to show maps that are related to your choice. The following figure shows the expanded view.

Figure 2. Expanded view of a format



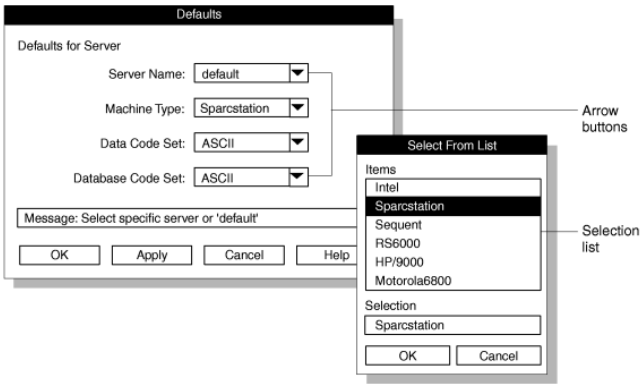
You can click the icon that you want to display a definition window for any format or map that is shown.

Selection-List windows

A Selection-List window lists the possible values for a text box. A down arrow that follows a text box indicates that you can use a selection list to see and select possible values for the text box. When you click the down arrow, the corresponding Selection-List window appears.

The following figure shows the selection list that is available for the Machine Type text box in the Defaults window. After you select an item in the list box, click OK, and the item appears in the text box on the original window.

Figure 1. The Defaults window and a selection list

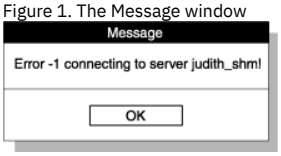


Tip: If your entry is rejected, look at the selection list. Your entry is invalid if it is not available in the selection list.
Selection-List windows are available for many text boxes throughout the HPL user interface. These windows have various names, but these topics refer to them as selection lists.

Message windows

A message window typically contains either a warning or an information update. A warning lets you verify or cancel the action that you have chosen. An update informs you about the successful completion of an operation or explains why an operation failed.

The following figure shows a typical error message.
Figure 1. The Message window



The HPL ipload utility buttons

After you move beyond the main window, every window has at least one button to help you move through the interface.

In general, buttons appear in three locations:

- Toolbar buttons appear across the top of the display.
- Icon buttons appear in the middle section of the display.
- Buttons appear across the bottom of the display.

- The HPL ipload utility toolbar buttons
  Toolbar buttons appear at the top of many windows. The function of the window determines which buttons appear.
- The HPL ipload utility icon buttons
  Icon buttons appear in the middle sections of the Load Job, *Unload Job*, and Views windows. The icon buttons represent various components. When you click it, each button opens another display.
- Buttons at the bottom of the HPL ipload utility display
  The buttons across the bottom of the display let you indicate the next action. Most windows have one or more of these buttons.

The HPL ipload utility toolbar buttons

Toolbar buttons appear at the top of many windows. The function of the window determines which buttons appear.
The following sections describe the toolbar buttons. Buttons that appear in only one window are described with the specific window.

Button	Button name	Purpose	See
	Browse	Displays the Browse window	The HPL browsing options
	Copy	Copies the selected component (format, map, query, filter, device, or project) to a new item	The Copy button
	Connect	Lets you reattach to an active unload (or load) job from the Unload (or Load) Job Select window	Figure 1
	Delete (trash can)	Deletes the selected component (format, map, and so on)	The Delete button
	Delete (eraser)	Breaks the association between a database column and a data-file field	Using the Delete button
	File	Displays the Import/Export File Selection window	Exporting and importing queries
	Find	Allows you to quickly locate a particular field or column in a Map window	Using the Find button

Button	Button name	Purpose	See
	Generate	Lets you generate jobs automatically	The Generate options of the ipload utility
	Notes	Allows you to type descriptive text for an item	The Notes button
	Options	Displays an options window where you can change default values or supply additional parameters	Changing the unload options, Changing the load options
	Print	Prints the parameters for the selected item	The Print button
	Save As	Saves a copy of the currently selected item (behaves in the same way as the Copy button)	The Copy button
	Search	Displays a Views window where you can see the relationships among components	Views windows
	Specs	Displays the Specifications window, where you can view the attributes for selected columns or fields	Using the Specs button

- The Browse button
  With the Browse button, you can look through the files that show information about the load or unload jobs and any problems that the **onpload** utility found.
- The Copy button
  You can copy a selected component with the Copy button. This feature can save you time when you are creating a component. You can copy an existing component and then modify the copy with your changes.
- The Delete button
  The Delete button lets you delete one or more selected components.
- The Notes button
  You can type descriptive text for an item with the Notes button. The text of the note is displayed in the Notes area in a window when you select the item. The Notes button is a useful tool for identifying **ipload** components, load jobs, unload jobs, and projects.
- The Print button
  You can print information that is associated with a component with the Print button. Before you start **ipload**, you must set your workstation so that it can find a printer.

The Browse button

With the Browse button, you can look through the files that show information about the load or unload jobs and any problems that the **onpload** utility found.

Related concepts:
Browsing options

The Copy button

You can copy a selected component with the Copy button. This feature can save you time when you are creating a component. You can copy an existing component and then modify the copy with your changes.

You can copy one component at a time, or you can select and copy multiple components at the same time. You can copy components that are grouped under a project (filters, formats, maps, and queries) within the same project, or to a different project.

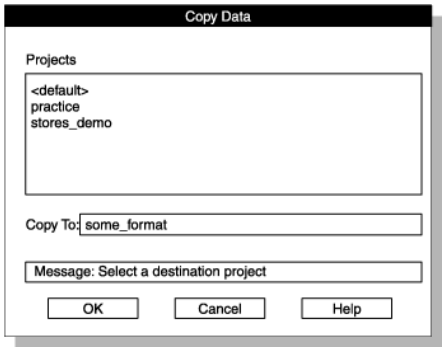
If you copy a component within a project, you must give the copy a different name. If you copy a component to a different project, you can retain the name for the copy or give the copy a different name. If you copy multiple components, you must copy them to a different project. When you copy multiple components, the components retain their names.
Important: Devices are not project specific. When you copy a device, you must give the copy a new name.

- Copying an existing format into a new format

Copying an existing format into a new format

To copy an existing format to a new format:

1. In the HPL main window, select the project that includes the format that you want to copy.
2. Choose Components > Formats to access the Record Formats window. For an example, see Creating a fixed format.
3. Select the format that you want to copy. This example assumes that the format to copy is **some_format**.
4. Click the Copy button.
   The Copy Data window appears, as the following figure shows. The Copy Data window displays a list of existing projects. The Copy To text box shows the name of the format that you are copying.
   Figure 1. The Copy Data window



5. Select the project to which you want to copy the format.
6. Type the name that you want to give to the copied format in the Copy To text box.
   If you are copying the format to another project, you can keep the same name. You must change the name, however, if you are copying the format to the same project.
7. Click OK. The display returns to the Record Formats window.
8. Click Cancel to return to the HPL main window.

The Delete button

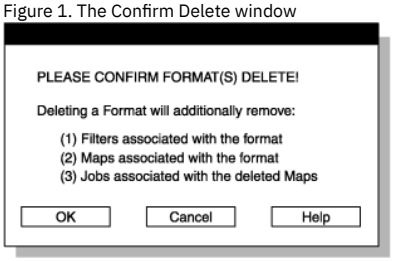
The Delete button lets you delete one or more selected components.

- Deleting an existing format

Deleting an existing format

To delete an existing format:

1. In the HPL main window, select the project that includes the format that you want to delete.
2. Choose Components > Formats to access the Record Formats window. For an example, see Creating a fixed format.
3. Select the format that you want to delete.
4. Click the Delete button.
   The Confirm Delete window appears, as the following figure shows. The Confirm Delete window describes the impact of deleting this format. The text in this window is different for each of the component types.
   Figure 1. The Confirm Delete window



5. Click OK to confirm the deletion, or click Cancel to cancel it.
   If you click OK, the format is deleted and any associated maps, filters, and jobs.
6. Click Cancel to return to the HPL main window.

The Notes® button

You can type descriptive text for an item with the Notes button. The text of the note is displayed in the Notes area in a window when you select the item. The Notes button is a useful tool for identifying **ipload** components, load jobs, unload jobs, and projects.

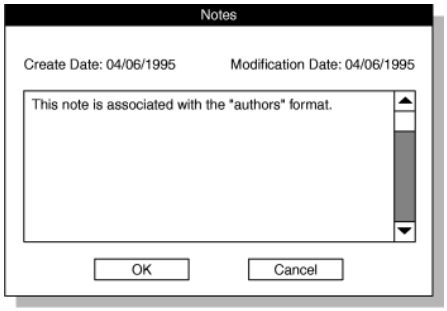
- Creating a note

Related reference:
The note table in the onpload database

Creating a note

To create a note:

1. Click the Notes button in a Component-Definition window.
   The Notes window appears, as the following figure shows.
   Figure 1. The Notes window

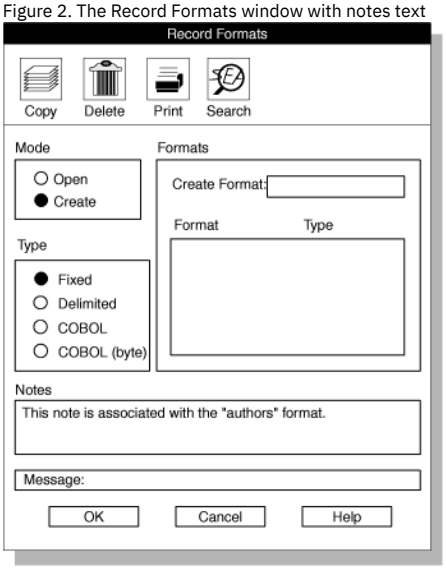


2. Type the descriptive text in the Notes text box.
3. Click OK to store the note and return to the Component-Definition window.
   When you select the component, the note text is displayed in the Notes area.

If you do not change a note, click Cancel instead of OK.

For example, the note created in the Notes window is associated with the **authors** format. The next time you go to the Record Formats window and select authors, **ipload** displays the note text, as the following figure shows.

Figure 2. The Record Formats window with notes text



The **ipload** utility stores the information that you type in the **note** table window of the onpload database. For a description of the **note** table, see The note table in the onpload database.

The Print button

You can print information that is associated with a component with the Print button. Before you start **ipload**, you must set your workstation so that it can find a printer.

For information about setting up a printer, see your operating-system publications.

If you click the Print button in the Map-Definition window in Figure 2, the following printout results:

```
--------------------------------------------------------------------------------
LOAD MAP REPORT
--------------------------------------------------------------------------------

Project : <default>
Name: a_map

OPTIONS
 Database Table  Format
 -------- ---------  --------------
 testdbtab1a_format

RECORD FORMAT MAP VIEW
 Format FieldTable Column Option Data
 ----------------------  -------------------  ------------------
 input1  col3
 input2  col2
 input3  col1
```

The HPL ipload utility icon buttons

Icon buttons appear in the middle sections of the Load Job, *Unload Job*, and Views windows. The icon buttons represent various components. When you click it, each button opens another display.

The following table shows and describes the icon buttons that are used in these windows.

Component	Description	Window	Action
 Device	The device or device array where the source files are located	Load Job, Unload Job	• If the text box is empty, click the Device button to display the Device Array Selection window, where you can create or open a device type. • If the text box has an entry, click the Device button to display the Device Array Selection window for that specific device or type the name of a different device in the text box.
 Filter	The filter that controls which records are selected from the data file for a database update (The use of a filter is optional.)	Load Job, Filter Views	• If the text box is empty, click the Filter button to display the Filter Views window, where you select a filter and associated format. You can also create a filter from this window. • If the text box has an entry, click the Filter button to display the Filter-Definition window for that specific filter or type the name of a different filter in the text box. • In the Filter Views window, click the Filter button to display the Filter-Definition window for a specific filter.
 Format	The format of the source data used for this load or unload	Load Job, Unload Job	• If the text box is empty, click the Format button to display the Format Views window, where you can select a format and associated map. You can also create a format from this window. • If the text box has an entry, click the Format button to display the Format-Definition window for that specific format or type the name of a different format in the text box. • In all Views windows, click the Format button to display the format definition for a specific format. In these windows, the button shows only one of the three symbols (F, D, C) to indicate whether the type of format is fixed, delimited, or COBOL.
 Map	The map that correlates fields of the data source to database columns	Load Job, Unload Job, Map Views, Format Views, Database Views	• If the text box is empty, click the Map button to display the Map Views window, where you can select a map and associated table and format. You also can create a map from this window. • If the text box has an entry, click the Map button to display the Map-Definition window for that specific map or type the name of a different map in the text box. • In a Views window, click the Map button to display the Map-Definition window for a specific map.
 Options	The options that let you specify characteristics of the load or unload	Load Job, Unload Job	• Click the Options button to display the Load Options window.
 Query	The query that selects data from the database table	Unload Job, Database Views, Map Views	• If the text box is empty, click the Query button to display the Database Views window from which you can select the table and associated map and format. • If the text box has an entry, click the Query button to display the Query-Definition window for that specific query or type the name of a different query in the text box. • In a Views window, click the Query button to display the Query-Definition window for a specific query.
 Table	The database table into which the converted data will be loaded	Load Job, Database Views, Query Definition	• Click the Table button to display the Database Views window from which you can select the table and associated map and format. If an association is not apparent, click Create to create one. • Click the Table button in the Query-Definition window to choose a table and columns for the Select entry.

Related tasks:
Changing the load options

Buttons at the bottom of the HPL ipload utility display

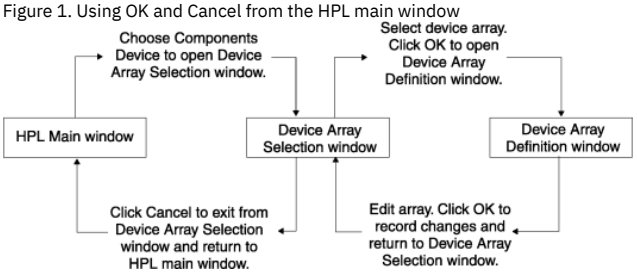
The buttons across the bottom of the display let you indicate the next action. Most windows have one or more of these buttons.

Table 1. Buttons that indicate the next action

Button name	Action
Apply	Saves changes but does not exit.
Cancel	Does not save any changes. Exit to the previous display.
Create	Displays the Component-Selection window.
Help	Displays context-sensitive help in a separate window. For information about the online help, see The HPL online help.
OK	Save changes and exit to the previous display.

Use OK only when you have changed the display. If you are exiting from a series of displays, use Cancel to exit from the display. The following figure shows the use of OK and Cancel.

Figure 1. Using OK and Cancel from the HPL main window



The HPL online help

The Help menu on the High-Performance Loader (HPL) main window has Glossary and Contents choices.

The **Glossary** option opens a scrolling list of items. Select an item to see its definition. The **Contents** option takes you to the main contents page. This page directs you to discussions of various HPL topics.

If you click Help in any window other than the HPL main window, Help displays information that is related to the current window. After the Help window opens, you can click its Help button for more information about using the Help window.

The UNIX keyboard commands to move the cursor

Instead of using the mouse to move from area to area in the High-Performance Loader (HPL) user interface, you can use UNIX keyboard commands to move the cursor. As you move around, the currently selected item is highlighted with a box.

The following table lists the cursor-moving keystrokes.

Table 1. Cursor moving keystrokes

Keystroke	Result
TAB	Move from area to area. Sometimes used to move from tab stop to tab stop.
SHIFT-TAB	Back up; that is, move from area to area in reverse order.
CONTROL-TAB	Move from area to area when `TAB` is reserved to move from tab stop to tab stop.
Cursor keys	Move from item to item within a functional area.
SPACEBAR	Select the current item or action.

Most displays in the HPL user interface are divided into functional areas, such as toolbar buttons, selection group, component-name text box, component list box, and so on. Depending on the nature of the specific display, sometimes `TAB` moves from item to item (or even from tab stop to tab stop) within a major area. On other displays, `TAB` moves only between functional areas, and you must use `SPACEBAR` to move around within the functional area.

Define HPL projects

This section explains how to create a project and how projects are related. The individual components that you store in projects are described in later sections.

- HPL projects
  You can organize your work by specifying *projects* with the High-Performance Loader (HPL). A project is a collection of individual pieces that you use to load and unload data. A project can include load and unload jobs and the maps, formats, filters, and queries that you use to build the load and unload jobs.
- Project organization
  The High-Performance Loader (HPL) uses only one database, onpload, to track the preparation that you do for loading and unloading data. By using projects, you can organize your work into functional areas.
- Select or create a project with the Projects window
  You can select or create a project from the Projects window. After you select a project, you can copy the project, delete it, print the project parameters, or make a note that describes the project.

Related reference:
Choose a project

HPL projects

You can organize your work by specifying *projects* with the High-Performance Loader (HPL). A project is a collection of individual pieces that you use to load and unload data. A project can include load and unload jobs and the maps, formats, filters, and queries that you use to build the load and unload jobs.

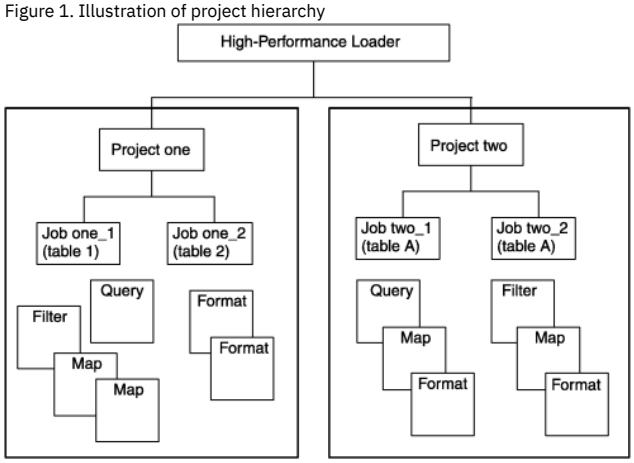
Project organization

The High-Performance Loader (HPL) uses only one database, onpload, to track the preparation that you do for loading and unloading data. By using projects, you can organize your work into functional areas.

For example, you might regularly transfer data to or from several unrelated databases. You could put all of the preparation for each database into a separate project.

When you first start **ipload**, **ipload** creates a project named **<default>**. If you prefer, you can select the **<default>** project and assign all of your work to that project. The HPL does not require that you create any additional projects. However, creating projects and putting separate tasks into distinct projects makes your work easier to maintain.

The following figure shows the relationships among projects, jobs, and components.
Figure 1. Illustration of project hierarchy

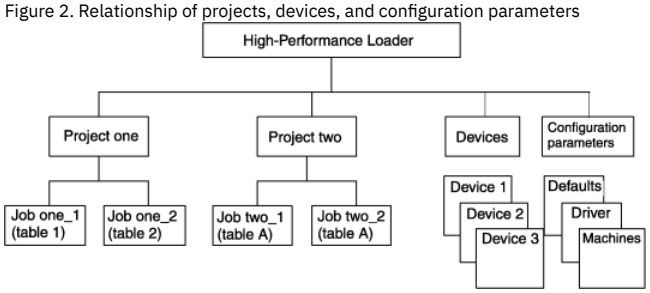


The preceding figure shows that jobs are linked directly to the projects. The format, map, filter, and query components belong to a project but are not directly linked to a job, as illustrated with Project One. In general, you create a format, map, and filter or query for each job, as shown with Project Two. However, in some cases, you might use the same component for more than one job within a project.

For example, for reports about a medical study, you might want to create three reports: one about subjects under 50 years of age, one about subjects over 50, and one about all subjects. In that case, the description of how to find the information (the format and map) is the same for all three reports, but the selection of information (the query) is different for each report. (Formats, maps, and queries are described in detail in later chapters.)

All components (maps, formats, queries, filters, and load and unload jobs) that you create in a project are associated with that project in the onpload database. Components that are associated with a project are visible (usable) only when the project is selected. When you select a different project, a different set of components becomes available.

Device-array definitions and configuration parameters are not included in project definitions. The following figure shows the components that the HPL uses. Each project is distinct, but the devices and configuration parameters apply to all projects.
Figure 2. Relationship of projects, devices, and configuration parameters

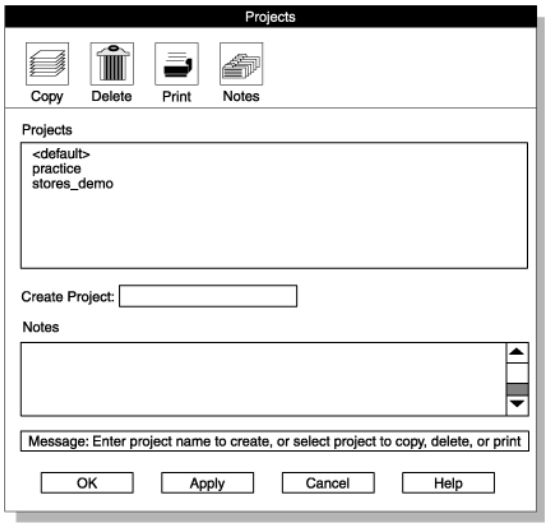


Related reference:
Define projects
The project table in the onpload database

Select or create a project with the Projects window

You can select or create a project from the Projects window. After you select a project, you can copy the project, delete it, print the project parameters, or make a note that describes the project.

The **ipload** utility stores project information in the **project** table of the onpload database.
Figure 1. The Projects window



- Defining a new project
- Selecting a project
  The HPL provides two methods for selecting a project.

Related reference:
The project table in the onpload database

Defining a new project

To define a new project:

1. Choose Configure > Project from the HPL main window.
   The Projects window appears, as Figure 1 shows.
2. Choose a name for the project and type it in the Create Project text box.
3. Click Apply.
   The **ipload** utility creates the project but does not exit from the Projects window. You can create another project, or you can use the toolbar buttons to manipulate the project.
4. Click Cancel to return to the HPL main window.

If you want to create one project and then exit, click OK instead of Apply.

Related concepts:
Initial options on the HPL main window

Selecting a project

The HPL provides two methods for selecting a project.

- Selecting a project for a load or unload job
- Selecting a project to edit

Selecting a project for a load or unload job

To select a project for a load or unload job or to edit components:

1. Select the project name from the Select Project list box in the HPL main window.
2. Choose the action that you want to take from one of the menus on the HPL main window.

Selecting a project to edit

To select a project to edit:

1. Choose Configure > Project from the HPL main window.
   The Projects window appears, as Figure 1 shows.
2. Select the project that you want to edit from the Projects list box.
3. Perform the edit actions (copy, delete, print, or describe with a note) you want.

4. Click Cancel to return to the HPL main window.

Configure the High-Performance Loader

This section describes the process of configure the High-Performance Loader (HPL).

- Configure the ipload utility
  When you configure the **ipload** utility, you describe the type of computer, code sets, and other aspects of your database server environment. Configuration information is stored in the **onpload** database.
- Select a database server
  The High-Performance Loader (HPL) needs to know the location of two databases: the onpload database and the *target database*.
- Modify the onpload default values
  You must describe the computer environments of your database servers. This information applies to database servers. If you change the description of a database server, the changes apply to all jobs that you run on that database server. You can prepare a default computing environment that applies to all database servers that are not explicitly described.
- Selecting a driver
- Modify the machine description
  The information that the Machines option of the Configure menu stores describes the characteristics of a specific computer. The High-Performance Loader (HPL) uses these characteristics to control the conversion of binary data formats. Unless you are converting binary data, you do not need to be concerned about the machine description.

Related reference:
Check the ipload utility default values

Configure the ipload utility

When you configure the **ipload** utility, you describe the type of computer, code sets, and other aspects of your database server environment. Configuration information is stored in the **onpload** database.

The configuration tasks include:

- Selecting a database server
- Modifying the onpload defaults
- Selecting or creating a driver
- Modifying the machine description

Related concepts:
The ipload utility
Related reference:
HPL performance

Select a database server

The High-Performance Loader (HPL) needs to know the location of two databases: the onpload database and the *target database*.

The target database is the IBM® Informix® database into which you load data or from which you unload data. When you start **ipload**, **ipload** assumes that both the onpload database and the target database are on the database server that the INFORMIXSERVER environment variable specifies. You can use the Connect Server window (Figure 1) to specify different database servers.

The sqlhosts file or registry controls connectivity to database servers. The **ipload** utility scans the sqlhosts information to derive the lists of available database servers that the Connect Server window displays.
Restriction: You cannot use the alias **loghost** as a machine name in the sqlhosts file. The **loghost** alias is present on all computers; consequently, onpload cannot correctly identify computers based on this name.

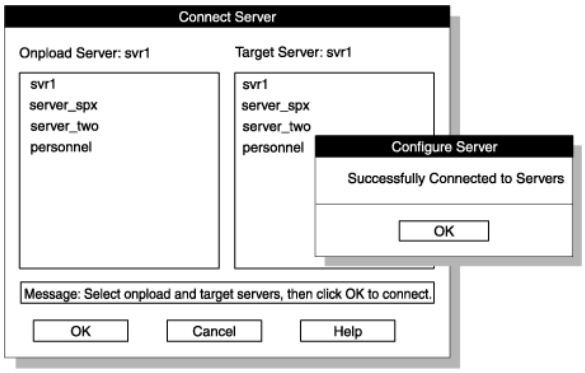
- Selecting a database server
- Create the onpload database
  When you first start **ipload**, **ipload** creates an onpload database. If you use the Connect Server window to choose a different onpload database server, **ipload** creates another onpload database on that database server.

Related concepts:
Initial options on the HPL main window

Selecting a database server

To select a database server:

1. Choose Configure > Server from the HPL main window.
   The Connect Server window appears, as the following figure shows.
   Figure 1. The Connect Server window



2. Select the database server where the onpload database is going from the Onpload Server list box.
3. Select the database server that includes the database that you will load or unload from the Target Server list box.
4. Click OK.
5. Click OK in the Configure Server window to return to the HPL main window.

Create the onpload database

When you first start **ipload**, **ipload** creates an onpload database. If you use the Connect Server window to choose a different onpload database server, **ipload** creates another onpload database on that database server.

The default name of the database that the High-Performance Loader (HPL) uses is onpload. To give some other name to the HPL database, set the DBONPLOAD environment variable.

Related tasks:
Preparing multiple onpload databases

Modify the onpload default values

You must describe the computer environments of your database servers. This information applies to database servers. If you change the description of a database server, the changes apply to all jobs that you run on that database server. You can prepare a default computing environment that applies to all database servers that are not explicitly described.

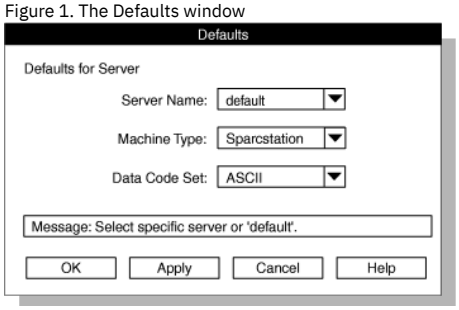
- The Defaults window
  You can change the server name, machine type, and data code set from the Defaults window.
- Specifying the onpload defaults

Related reference:
Look at the Defaults window
The defaults table in the onpload database

The Defaults window

You can change the server name, machine type, and data code set from the Defaults window.

The following figure shows the Defaults window.
Figure 1. The Defaults window



The **ipload** utility saves the information from the Defaults window in the **defaults** table of the onpload database. For more information about the **defaults** table, see The defaults table in the onpload database.
Tip: You can use DB-Acess to examine the default values. The following tables in the onpload database contain default values: **defaults**, **delimiters**, **driver**, and **machines**.

Table 1. The Defaults window options

Display option	Header

Display option	Header
Server Name	The Server Name text box specifies the database server with which the settings are associated. The information provided for the special server name **default** applies to all database servers for which no explicit information is provided. For example, if most the database servers on your network (that will be using the HPL) are BrandX computers, **default** should describe the BrandX computers. To describe the computing environment of the other database servers on the network, specify the database server name. The selection list that is associated with the Server Name text box lists the database servers that are identified in the sqlhosts information.
Machine Type	The Machine Type text box describes fixed-length, binary-format records. It defines the sizes and byte order of data in data files that the specified database server produces. The selection list that is associated with the Machine Type text box provides descriptions of several computers. You can use the Machines option on the Configure menu to add descriptions of other computers to this list (see Modify the machine description).
Data Code Set	The Data Code Set text box specifies the character set of the data file. When you load data into a database, you can convert the character set of the data file into the character set of the database. For example, you can convert EBCDIC to ASCII, or any other character set that your system supports. Conversely, you can convert the data from a database into a selected character set when you unload data. You can select the GLS code set you want from this selection list. The character set of the database is determined by the DB_LOCALE environment variable. For information about locales and code sets, see the *IBM® Informix® GLS User's Guide*.

Specifying the onpload defaults

To specify defaults for onpload:

1. Choose Configure > Defaults from the HPL main window.
   The Defaults window appears, as Figure 1 shows.
2. Update the values in each of the text boxes. Click the down arrows to display selection lists that show possible values for each text box.
3. Click Apply to save the values and prepare the defaults for another database server.
4. Click Cancel to return to the HPL main window.

If you want to prepare the defaults for only one database server, click OK instead of Apply.

Related concepts:
Modify the machine description

Selecting a driver

In a UNIX environment, you can use pipe-device arrays to connect custom programs to the input or output of the **onpload** utility. Although pipe-device arrays provide the simplest way to customize the input or output of **onpload**, connecting the standard I/O of programs is less efficient than directly incorporating the functionality into **onpload**.

The **ipload** utility identifies custom software by the driver name that you assign to the record-format definitions.

To incorporate custom file-handling software directly into the **onpload** program:

1. Use the API described in Available API support functions to write the driver.
2. Add the driver name to the onpload database. For details, see Adding a custom-driver name.
3. Select the driver from the Driver selection list in the Format Options window. For details, see Format options.

- The Drivers window
  You can add information about custom drivers from the Drivers window. After you add a custom driver name, you can assign the driver to the record-format definitions.
- Adding a custom-driver name

The Drivers window

You can add information about custom drivers from the Drivers window. After you add a custom driver name, you can assign the driver to the record-format definitions.

The following figure shows the Drivers window. The Driver Name list box displays the currently available drivers, their class, and type.
Figure 1. The Drivers window

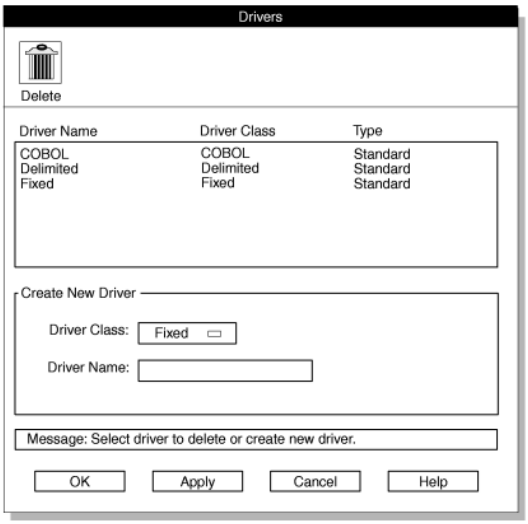


Table 1. The Drivers window options

Display option	Description
Driver Name	The Driver Name text box specifies the name of the custom-driver program. Before you can use the custom driver, you must add the program to your **onpload** shared library. For more information, see Rebuilding the shared-library file.
Driver Class	The Driver Class text box specifies the format type that the custom driver supports. The custom driver must produce data in a format that **onpload** supports. The **onpload** utility supports the following driver classes: • Fixed • Delimited • COBOL

Related reference:
The driver table in the onpload database

Adding a custom-driver name

To add a custom-driver name:

1. Choose Configure > Driver from the HPL main window.
   The Drivers window appears, as Figure 1 shows.
2. Type the name of the driver that you are adding in the Driver Name text box.
3. Select the driver class.
4. Click Apply to save this driver and add another driver.
   If you want to add only one driver, click OK instead of Apply.
5. When you finish, click OK to return to the HPL main window.

Modify the machine description

The information that the Machines option of the Configure menu stores describes the characteristics of a specific computer. The High-Performance Loader (HPL) uses these characteristics to control the conversion of binary data formats. Unless you are converting binary data, you do not need to be concerned about the machine description.

When you first start **ipload**, **ipload** stores the characteristics of several computers. You can select one of the existing computer types, modify an existing description, or create a machine description.

You use the information from the Machines window when you prepare the defaults for the database servers on your network. The information from the Machines window is stored in the **machines** table of the onpload database. The default information for the HPL includes descriptions of the binary data sizes for several computers. If the default data does not include the computer from which you are reading data, you can create a description for that computer.

- The Machines window
- Editing the description of a computer
- Adding a computer type to the Machines list

Related tasks:
Specifying the onpload defaults
Related reference:
Look at the Machines window
The machines table in the onpload database

The Machines window

The following figure shows the Machines window. If you select SPARCstation from the Machine Type selection list, the following values appear in the Machines window.

Figure 1. The Machines window

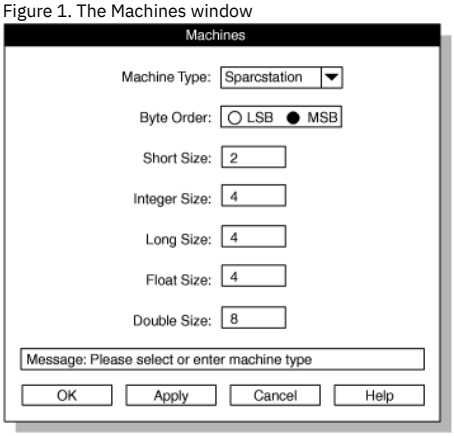


Table 1. The Machines window options

Display option	Description
Machine Type	Name for the computer type that this entry describes.
Byte Order	Bit ordering of binary information for this computer. The two possible formats are LSB and MSB. In the LSB format, the least-significant bits of a value are at lower memory addresses. In the MSB format, the most-significant bits of a value are at lower memory addresses.
Short Size	Number of bytes required to hold a short integer value.
Integer Size	Number of bytes required to hold an integer.
Long Size	Number of bytes required to hold a long-integer value.
Float Size	Number of bytes required to hold a floating-point value.
Double Size	Number of bytes required to hold a double-sized floating-point value.

Related reference:
The machines table in the onpload database

Editing the description of a computer

To edit the description of a computer:

1. Choose Configure > Machines from the HPL main window.
   The Machines window appears, as Figure 1 shows.
2. Click the down arrow and select a machine type from the selection list.
3. Modify the values as appropriate.
4. Click Apply to save the values and modify another machine description.
5. When you finish, click Cancel to return to the HPL main window.

If you want to modify the description of only one computer, click OK instead of Apply.

Adding a computer type to the Machines list

To add a computer type to the Machines list:

1. Choose Configure > Machines from the HPL main window.
   The Machines window appears, as Figure 1 shows.
2. Type the new name in the Machine Type text box.
3. Type an appropriate value in each text box.
4. Click Apply to save the values and add another computer type.
5. When you finish, click Cancel to return to the HPL main window.

If you want to add only one computer type, click OK instead of Apply.

Define device arrays

This section describes how to define and use device arrays with the High-Performance Loader (HPL).

- Device arrays
  A device array groups several I/O devices so that the High-Performance Loader (HPL) can perform parallel processing of the input and output. When you specify multiple devices in a device array, **onpload** sets up separate, parallel streams of input or output, when it performs a database load or unload.
- Multiple devices in a device array
- The Device-Array Selection window
  You can create a device array or select an existing array from the Device-Array Selection window. If you select an existing array, you can edit that array or use one of the toolbar buttons to copy, delete, or print the array.
- The Device-Array Definition window
  You can add, edit, or delete devices from an array from the Device-Array Definition window.

Related concepts:
Components of the unload job

Device arrays

A device array groups several I/O devices so that the High-Performance Loader (HPL) can perform parallel processing of the input and output. When you specify multiple devices in a device array, **onpload** sets up separate, parallel streams of input or output, when it performs a database load or unload.

Device arrays set up simultaneous access to one or more tape devices, files, or pipes (UNIX only) so that the **onpload** utility can take advantage of parallel processing.

Device arrays are not project specific. You can use the same device array for a load or unload job on any of the projects that you define.

Related reference:
Define device arrays

Multiple devices in a device array

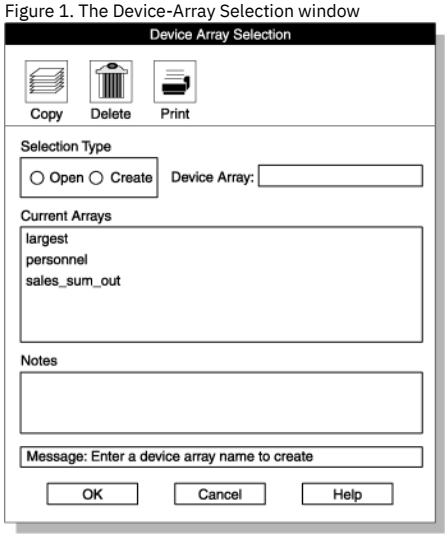
You can include files, pipes (UNIX only), and tape devices in a single array. Devices for the device array discusses factors that you should take into account when you decide what devices to assign to an array. If your array includes pipe commands, **onpload** starts the pipe when it begins execution.

When the High-Performance Loader (HPL) unloads data, it assigns records to the devices of a device array in a round-robin fashion.

The Device-Array Selection window

You can create a device array or select an existing array from the Device-Array Selection window. If you select an existing array, you can edit that array or use one of the toolbar buttons to copy, delete, or print the array.

The following figure shows the Device-Array Selection window.
Figure 1. The Device-Array Selection window



- Creating a device array
- Opening an existing device array

Creating a device array

To create a device array:

1. Choose Components > Devices from the HPL main window.
   The Device-Array Selection window appears, as Figure 1 shows.

2. Click Create in the Selection Type group.
3. Select a name for the new device and type it in the Device Array text box.
4. Click OK.
   The Device-Array Definition window appears, as Figure 1 shows.

Opening an existing device array

To open an existing device array:

1. Choose Components > Devices from the HPL main window.
   The Device-Array Selection window appears, as Figure 1 shows.

2. Click Open in the Selection Type group.
3. Select a device from the Current Arrays list box.
4. Click OK.
   The Device-Array Definition window appears, as Figure 1 shows.

The Device-Array Definition window

You can add, edit, or delete devices from an array from the Device-Array Definition window.

Figure 1. A partially completed Device-Array Definition window

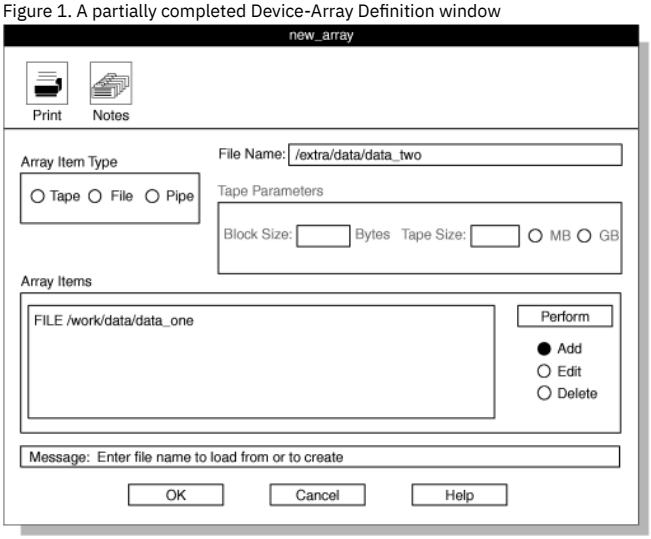
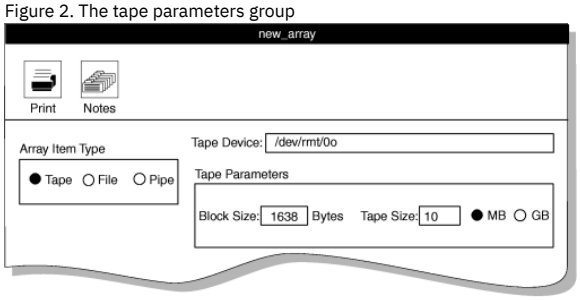


Table 1. The Device-Array Definition window options

Display option	Description
Array Item Type group	The Array Item Type group lists the types of devices that you can include in a device array. You can mix different types of devices in a single array.
Device text box	Depending on the array item type that you selected from the Array Item Type group, the label for the text box where you type a device name is **Tape Device**, **File Name**, or **Pipe Command** (UNIX only). Complete this text box as follows. **Device type** **What to type** Tape Device The full path name of the tape device (example: /dev/rmt/0) File Name The full path name of the file (example: /work/mydata) Pipe Command The full path name of the executable pipe command or shell script (example: /tmp/g)

Display option	Description
Tape Parameters group	When you select Tape as the array item type, the Tape Parameters group becomes active (not gray), as the following figure shows. You must type the block and tape size. The tape size must be greater than zero. The example shown in the following figure is for UNIX. An example of a tape device name for Windows is 11.1tape0. Windows does not support remote tape load and unload. Figure 2. The tape parameters group 

- Adding, editing, and removing devices
  You can add devices to the device array, you can edit the devices, and you can remove them.

Related reference:
The device table in the onpload database

Adding, editing, and removing devices

You can add devices to the device array, you can edit the devices, and you can remove them.

- Adding devices to the device array
- Editing a device in the device array
- Deleting a device from the device array

Adding devices to the device array

To add devices to the device array:

1. Click Add in the Device-Array Definition window.
2. Click the device type in the Array Item Type group.
3. Type the full path name of the device in the Device text box. If you specified a tape device, the Tape Parameters group becomes active, as Figure 2 shows.
   a. Type the block size in kilobytes.
   b. Click MB (megabytes) or GB (gigabytes) to specify the units to use for the tape size.
   c. Specify the tape size.
4. When you have included all of the information for the device, click Perform. The device that you added appears in the Array Items list box.
5. Repeat steps 2 through 4 to add other items to the device array.
6. When you have added all of the devices to the array, click OK to return to the Device-Array Selection window. Your new array appears in the Current Arrays list box.
7. Click Cancel to return to the HPL main window.

Editing a device in the device array

To edit a device in the device array:

1. Click Edit in the Device-Array Definition window.
2. Select a device from the Array Items list box. The selected item appears in the Device text box.
3. Edit the path name and tape parameters, as appropriate.
4. Click Perform.
5. Click OK to return to the Device-Array Selection window.
6. Click Cancel to return to the HPL main window.

Tip: When you edit a device, you can change the path name and the tape parameters, but you cannot change the array-item type (tape, file, pipe). If you need to change the device type, you must delete the item and then add a new item.

Deleting a device from the device array

To delete a device from the device array:

1. Click Delete in the Device-Array Definition window.
2. Select a device from the Array Items list box.
3. Click Perform.
4. Click OK to return to the Device-Array Selection window.

5. Click Cancel to return to the HPL main window.

Define formats

This section describes the formats that the High-Performance Loader (HPL) provides and shows how to prepare and edit the format component.

- Formats
  A format describes the structure of the data in a data file.
- Formats of supported datafile records
- Fixed-length records
- Delimited records
  Delimited records are records whose fields can vary in length. In a data file that contains delimited records, the records and fields are separated by a delimiter.
- COBOL records
- Other formats
  In addition to delimited, fixed, and COBOL formats, the High-Performance Loader (HPL) provides two other formats for loading and unloading data: fast format and fast job. These formats are not included on the Record Formats window because the format specifications are predefined; you do not need to make any choices.
- Format options
  The format options let you change the default driver, the character set, the default computer type, and the delimiters. Information about the format options is stored in the **formats** table of the **onpload** database.
- Format Views window
  You can display a list of the formats and load and unload maps that are associated with a project from the Format Views window. You can also create or edit a format.

Related concepts:
Components of the unload job

Formats

A format describes the structure of the data in a data file.

Before you can import records from a data file into IBM® Informix® databases or export records from a database to a data file, you must define a format that describes the data file. You do not need to define a format for the database because **ipload** already knows the schema (the organization) of the database table.

These topics use the term *format* in two ways:

- To refer to the arrangement of data fields in a record of a data file
- To refer to the HPL component that documents the arrangement of the data fields

After you familiarize yourself with the concepts in this section, you might save yourself some work by using one of the Generate options to create formats automatically.

Related reference:
The Generate options of the ipload utility

Formats of supported datafile records

Data files can be structured in various ways. The High-Performance Loader (HPL) supports data-file records of the following formats:

- Fixed-length
- Delimited
- COBOL
- Other formats

You can define new format components at any time. Also, you can test your format before you actually load or unload data.

The **ipload** utility includes options that let you modify the data before it is inserted into the database.

The **ipload** utility stores information about formats in the **formatitem** and **format** tables of the onpload database.
Important: To prepare the format component, you must know the format of the records in the data file. If you do not know the data-file format, you must get it from the person who provided the data file.
Related concepts:
Preview data-file records
Related reference:
Format options
The formatitem table in the onpload database
The formats table in the onpload database
Define formats

Fixed-length records

In fixed-length or fixed-format records, each field starts and ends at the same place in every record. A data file that contains data records of equal and constant length might be organized as follows.
Figure 1. Sample file with fixed-length records

```
aaabbbbcccddddggghhhh
```

The data file illustrated previously has three records. Each record has a field of three characters followed by a field of four characters, so the total record length is seven characters. The file does not contain any separation between records; delimiters are unnecessary because all fields have the same length. The VARCHAR data types are therefore always the fixed maximum size of the field.

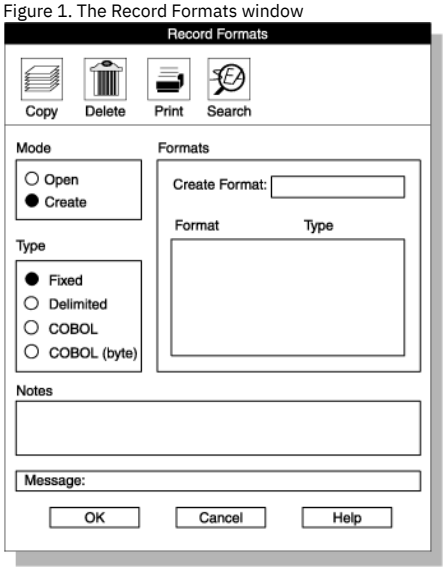
When you define a fixed-length format, you specify the length of each field. The **ipload** utility calculates the offset for each field and the total length of the record from the field lengths that you supply.

- Creating a fixed format
  You can create and define formats for fixed-length records from the Record Formats and the Fixed Format Definition windows.
- Editing a format
  After you create and save a format, you might need to add a new field, insert a new field, edit a field, or delete a field. The process for editing an existing format is essentially the same, regardless of the file type.
- Create a fixed format that uses carriage returns
- Create a fixed format that includes BYTE or TEXT data
  You can organize the byte or text data in a fixed-format data file with inline data or data in a separate file.
- Create a fixed format that includes Ext type or Simple LO data
  You can organize the Ext Type or Simple LO data in a fixed-format data file with fixed-length data or inline data.

Creating a fixed format

You can create and define formats for fixed-length records from the Record Formats and the Fixed Format Definition windows.

To create a format for fixed-length records:

1. Choose Components > Formats from the HPL main window.
   The Record Formats window appears, as the following figure shows.
   Figure 1. The Record Formats window
   

2. Click Create in the Mode group.
3. Click Fixed in the Type group.
4. Choose a name for the format and type it in the Create Format text box.
5. Click OK.
   The Fixed Format definition window appears. The title bar includes the name that you chose for the format. The following figure shows the Fixed Format Definition window as it might appear after you prepare the format for the file that Figure 1 shows.
   Figure 2. A completed Fixed-Format Definition window with an open selection list

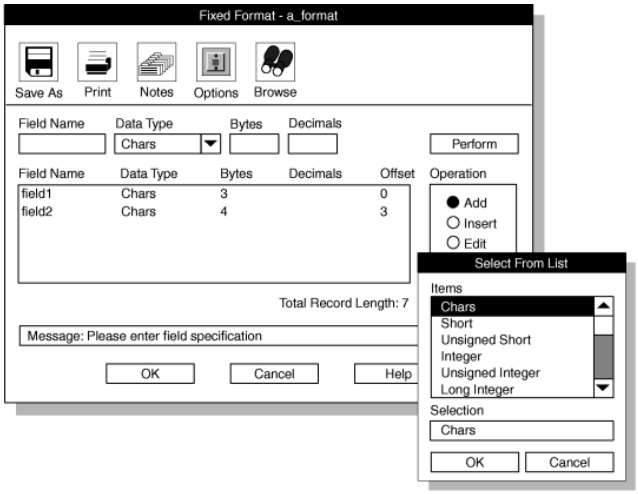


Table 1. The Fixed-Format Definition window options

Display option	Description
Bytes	The Bytes text box specifies the number of characters that the field occupies in the record. In the Bytes text box, you must set the number of bytes for your data types. Although **ipload** uses default information to calculate an offset if you create a format that has a new length, it does not readjust the lengths for existing formats. To change the default information, see The Machines window. The **ipload** utility automatically calculates the total length of the data file as you add each field description.
Decimals	The Decimals text box specifies the number of decimal places that are displayed when you convert floating-point types to ASCII. You can set the number of decimals only for the **Float** and **Double** data types.

6. Click Add in the Operation group.
7. Choose a name for the field and type the name in the Field Name text box.
8. Type the data type in the Data Type text box. For columns that contain user-defined types (UDTs) columns, you must choose an Ext Type format. For information about user-defined types, see *IBM® Informix® Guide to SQL: Reference*.
   The down arrow next to the Data Type text box displays the selection list that appears at the right in Figure 2. Data types allowed in a fixed format describes the data types that appear in the selection list.
9. Type the appropriate value in the Bytes text box (or in the Decimals text box, if appropriate). If you chose the Ext Type String in the Data Type box, you must specify a size, or you get an error. Field size depends on the type of data and its representation in the data file.
10. Click Perform.
    After you click Perform, **ipload** calculates the proper offset for this field in the record and displays the value under the Offset heading, as Figure 2 shows.
11. Repeat steps 7 through 10 for each field in your data file.
12. Click OK to save the format and return to the Record Formats window.
13. Click Cancel to return to the HPL main window.

Tip: Use the field name to map the data file to the database. You can type any name that you choose. You might find it easier to remember the names if you use the same name as the corresponding column of the database.

- Data types allowed in a fixed format

Related reference:
The formats table in the onpload database

Data types allowed in a fixed format

You can use the following data types when you are preparing a fixed format.

Data type	Description
Chars	ASCII format data
Short Unsigned Short Integer Unsigned Integer Long Integer Unsigned Long Float Double	The Machines description specifies the number of bytes required in fixed format for integers and floating-point values (see The Machines window.) When you select one of these data types, **ipload**sets the number of bytes.
Date	Date string
UNIX Date	A long integer interpreted as the system date from a UNIX system
Blob Length	The number of bytes of BYTE and TEXT (binary large object) that follow this record
Blob File	A file that contains BYTE and TEXT data
BLOB and CLOB	A file that contains smart large object (BLOB or CLOB) data
Simple LO Length	The number of bytes of simple large object data (BYTE and TEXT data) that follow this record
Simple LO File	The name of a file that contains a Simple LO data (BYTE and TEXT data)
INT8	An eight-byte integer type
SERIAL8	An eight-byte serial column
BIGINT	An ASCII character string or an eight-byte integer type

Data type	Description
BIGSERIAL	An ASCII character string or an eight-byte integer type
Ext Type String	The ASCII representation of an extended data type (Ext Type) value
Ext Type String Length	The length of an ASCII Ext Type value. The Ext Type value follows at the end of the input/output record. Use Ext Type String Length data type if you have data that contains null UDT values.
Ext Type Binary	The binary representation of an Ext Type value
Ext Type Binary Length	The length of the binary representation of an Ext Type value. The binary value follows at the end of the input/output record. Use Ext Type Binary Length data type if you have data that contains null UDT values.

The HPL supports several data types under the Ext Type mechanism. As a result, the specific names of these data types do not appear in the data-type selection list. For the following data types, choose the appropriate Ext Type data type:

- INT8
- LVARCHAR
- SERIAL8
- BLOB
- BOOLEAN
- CLOB
- Collection data types
- Distinct data type
- Opaque data types
- Row types

Editing a format

After you create and save a format, you might need to add a new field, insert a new field, edit a field, or delete a field. The process for editing an existing format is essentially the same, regardless of the file type.

The following example uses a fixed-format file, but the same procedure applies to COBOL and delimited files also.

- Adding a new field description to the format
- Inserting a new field into the format
- Editing the description of a field
- Deleting a field description from the format

Related concepts:
Preview data-file records
Related tasks:
Editing a format

Adding a new field description to the format

To add a new field description to the format:

1. Open the Fixed Format Definition window. For more information, see Creating a fixed format.
2. Click Add in the Operation group.
3. Type the field specifications in the text boxes at the top of the window.
4. Click Perform to add the new field at the end of the list.
5. Click OK to save your changes and return to the Record Formats window.
6. Click Cancel to return to the HPL main window.

Inserting a new field into the format

To insert a new field into the format:

1. Open the Fixed Format Definition window. For more information, see Creating a fixed format.
2. Click Insert in the Operation group.
3. Select the field before which you want to insert the new field.
4. Type the field specifications in the text boxes at the top of the window.
5. Click Perform to insert the new field before the selected field.
6. Click OK to save your changes and return to the Record Formats window.
7. Click Cancel to return to the HPL main window.

Editing the description of a field

To edit the description of a field:

1. Open the Fixed Format Definition window. For more information, see Creating a fixed format.

2. Click Edit in the Operation group.
3. Select the field from the list of fields.
4. Change the information that you want.
5. Click Perform.
6. Click OK to save your changes and return to the Record Formats window.
7. Click Cancel to return to the HPL main window.

Deleting a field description from the format

To delete a field description from the format:

1. Open the Fixed Format Definition window. For more information, see Creating a fixed format.
2. Click Delete in the Operation group.
3. Select the field that you want to delete.
4. Click Perform to delete the field.
5. Click OK to save your changes and return to the Record Formats window.
6. Click Cancel to return to the HPL main window.

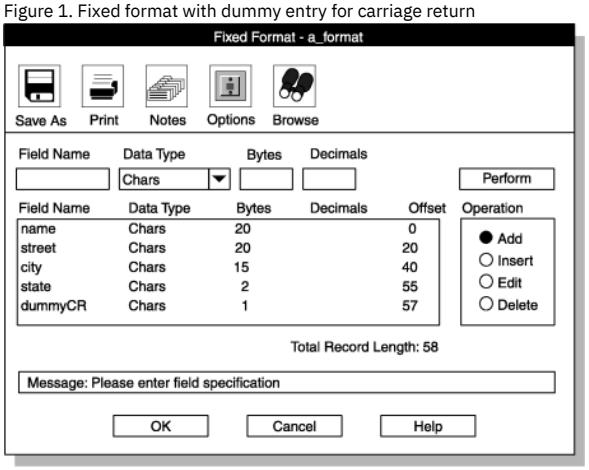
Create a fixed format that uses carriage returns

A fixed-format data file often includes a carriage return (new line) at the end of each record, such as the data file in the following example file.

```
20 chars            20 chars            15 chars2         chars

John Brown          100 Main St.        Citadel           LA
Mary Smith          3141Temple          WayChesapeake     AZ
Larry Little        44 Elm Rd. #6       Boston            MA
```

When you prepare the format for this data file, you must include a dummy field for the carriage return. When you create the load map for this format, do not link the dummy field to a database column. The following figure shows the format for the data file illustrated previously.

Figure 1. Fixed format with dummy entry for carriage return



Related tasks:
Creating a load map

Create a fixed format that includes BYTE or TEXT data

You can organize the byte or text data in a fixed-format data file with inline data or data in a separate file.

- Inline data
  BYTE or TEXT data that is included as part of a fixed-format data file is called inline data. When byte or text data is inline, the data-file record has two parts: a fixed-length part and a variable-length part.
- Data in a separate file
  You can also store BYTE and TEXT data in separate files.

Inline data

BYTE or TEXT data that is included as part of a fixed-format data file is called inline data. When byte or text data is inline, the data-file record has two parts: a fixed-length part and a variable-length part.

For example, a record with two fields and byte or text data might be organized as follows.

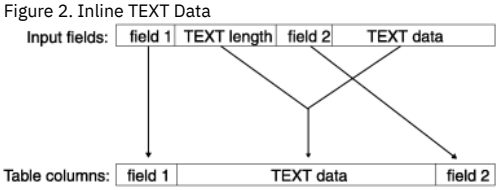
Figure 1. Organization of a record that includes inline TEXT data

```
field1textlengthfield2textdata
```

The length of the TEXT data is included in the fixed-length part of the record. The actual TEXT data is inserted at the end of the fixed-length part of the record. The High-Performance Loader (HPL) reads the TEXT length from the fixed-length part of the record and uses that length to read the actual TEXT data. The HPL also uses the TEXT length to calculate the offset to the beginning of the next record.

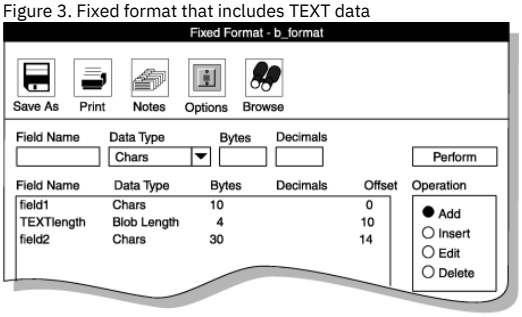
The following figure shows the format definition of a record with inline BYTE and TEXT data. The arrows show how the HPL puts the record into the database. The arrows from **field 1** and **field 2** indicate entries in fixed-length format. The split arrow shows that the HPL uses the **TEXT length** information to find the **TEXT data** and insert it into the table. The HPL does not insert the **TEXT length** into the database.

Figure 2. Inline TEXT Data



When you define the format in the Format-Definition window, select **Blob Length** as the data type for the **textlength** field. The following figure shows the format for the example in Figure 1. The format does not include an entry for TEXT data.

Figure 3. Fixed format that includes TEXT data



When you create a map to link the input fields that are defined by the format to the columns of a database table, connect the **textlength** input field to the table column that contains the TEXT data.

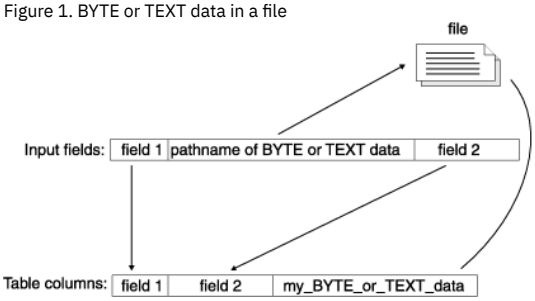
Related reference:
Define maps

Data in a separate file

You can also store BYTE and TEXT data in separate files.

During a load, BYTE and TEXT data files are read and inserted into the database. During an unload job, the file is created, and BYTE and TEXT data is written to the file. When the fixed-format input contains the path name of a data file, the HPL uses that path name to insert data into a column of the database table, as the following figure shows. When you prepare the format, select **Blob File** for the data type.

Figure 1. BYTE or TEXT data in a file



When you create a map to link the fields of the input record to the columns of a database table, link the name of the BYTE or TEXT file with the BYTE or TEXT column. The arrows in Figure 1 illustrate how the HPL inserts the BYTE or TEXT data into the column.

Related reference:
Define maps

Create a fixed format that includes Ext type or Simple LO data

You can organize the Ext Type or Simple LO data in a fixed-format data file with fixed-length data or inline data.

- Fixed-length data
  When you designate a field as an Ext Type String or Ext Type Binary data type, you specify that the data is going occupy a fixed amount of space, similar to the behavior of a fixed-length Chars data type. With fixed-length data format, you must specify the number of bytes that the field occupies in the record.
- Inline data
  Simple LO data or varying-sized Ext Type data that is included as part of a fixed-format data file is called inline data. When Ext Type or Simple LO data is inline, the data-file record has two parts: a fixed-length part and a variable-length part.
- Simple LO data in a separate file
  You can also store Simple LO data in separate files.

Fixed-length data

When you designate a field as an Ext Type String or Ext Type Binary data type, you specify that the data is going occupy a fixed amount of space, similar to the behavior of a fixed-length Chars data type. With fixed-length data format, you must specify the number of bytes that the field occupies in the record.

Inline data

Simple LO data or varying-sized Ext Type data that is included as part of a fixed-format data file is called inline data. When Ext Type or Simple LO data is inline, the data-file record has two parts: a fixed-length part and a variable-length part.

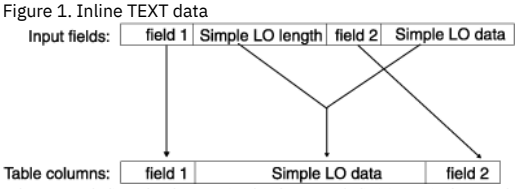
For example, a record with two fields and a Simple LO might be organized as follows.

```
field1Simple LO lengthfield1Simple LO data
```

The data-type length of the Ext Type or Simple LO data is included in the fixed-length part of the record. The actual TEXT data is inserted at the end of the fixed-length part of the record. The HPL reads the Ext Type or Simple LO length from the fixed-length part of the record and uses that length to read the actual Ext Type or Simple LO data. The HPL also uses the Ext Type or Simple LO length to calculate the offset to the beginning of the next record.

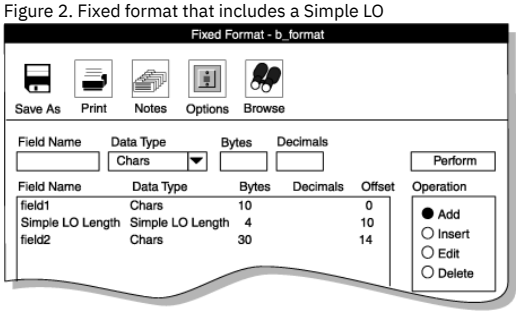
The following figure shows the format definition of a record with inline Ext Type or Simple LO data. The arrows show how the HPL puts the record into the database. The arrows from **field 1** and **field 2** indicate entries in fixed-length format. The split arrow shows that the HPL uses the **Simple LO length** information to find the Simple LO data and insert it into the table. The HPL does not insert the Simple LO length into the database.

Figure 1. Inline TEXT data



When you define the format in the format-definition window, select the appropriate data-type length data type (**Ext Type String Length**, **Ext Type Binary Length**, or **Simple LO Length**) for the data-type length field. The following figure shows the format for the example previously illustrated. The format does not include an entry for Simple LO data.

Figure 2. Fixed format that includes a Simple LO



Important: Ext Type binary-length format is not supported for complex types.

When you create a map to link the input fields that are defined by the format to the columns of a database table, connect the data type length input field to the table column that contains that particular data. In this case, connect the **Simple LO length** input field to the table column that contains the Simple LO data.

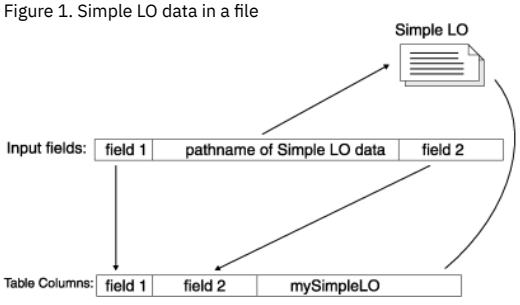
Related reference:
Define maps

Simple LO data in a separate file

You can also store Simple LO data in separate files.

During a load, Simple LO data files are read and inserted into the database. During an unload job, the file is created and BYTE and TEXT data is written to the file. When the fixed-format input contains the path name of a data file, the High-Performance Loader (HPL) uses that path name to insert data into a column of the database table, as the following figure shows. When you prepare the format, select **Blob File** for the data type.

Figure 1. Simple LO data in a file



When you create a map to link the fields of the input record to the columns of a database table, link the name of the Simple LO file with the Simple LO column. The arrows in Figure 1 illustrate how the HPL inserts the Simple LO data into the column.

Related reference:
Define maps

Delimited records

Delimited records are records whose fields can vary in length. In a data file that contains delimited records, the records and fields are separated by a delimiter.

The following data file uses a vertical bar (|) as the field delimiter and a carriage return as the record delimiter:

```
John Brown|100 Main St.|Citadel|LA|215/887-1931
Mary Smith|3141 Temple Way|Chesapeake|AZ|415/812-9919
Larry Little|44 Elm Rd. # 6|Boston|MA|617/184-1231
```

The **ipload** utility uses the vertical bar and carriage return as the default field and record delimiters.

- Create a delimited format
- Create a delimited format that includes BYTE or TEXT data
  In a delimited format, BYTE or TEXT data can be characters, hexadecimal data, or a separate file.
- Create a delimited format with extended data types

Related tasks:
Modifying delimited-format options

Create a delimited format

To create a format for delimited records, follow the same steps as in Creating a fixed format, with the following modifications:

- In step 3, click Delimited in the Type group.
- Omit step 9. When you use delimited records, **ipload** does not need byte or decimal information.

- Data types allowed in a delimited format

Data types allowed in a delimited format

You can use the following data types when you prepare a delimited format.

Table 1. Data types allowed in delimited format

Data type	Description
Chars	Normal ASCII data
BYTE or TEXT File	File that contains BYTE or TEXT data
TEXT Data	TEXT data is formatted as ASCII text. If the text includes carriage returns (new lines) or delimiters, a backslash (\) must precede those characters.
BYTE or TEXT HexASCII	BYTE or TEXT data that is formatted in ASCII hexadecimal. The **onpload** utility translates the data into binary format before it loads the data into the database.
BLOB or CLOB	File that contains smart large object (BLOB or CLOB) data
Simple LO File	The name of a file that contains Simple LO data (BYTE and TEXT data)
Simple LO Text	Simple LO data that is formatted as ASCII text. If the text includes carriage returns, newline characters, or delimiters, a backslash (\) must precede those characters
Simple LO HexASCII	Simple LO data that is formatted in ASCII hexadecimal. The **onpload** utility translates the data into binary format before it loads the data into the database.
Ext Type String	The ASCII representation of a Data Type (Ext Type) value
Ext Type HexASCII	The HexASCII representation of an Ext Type value

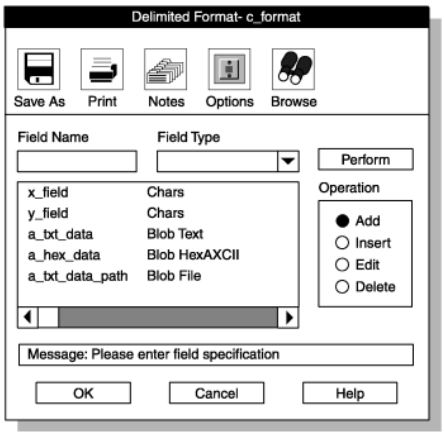
Create a delimited format that includes BYTE or TEXT data

In a delimited format, BYTE or TEXT data can be characters, hexadecimal data, or a separate file.

The following sample data-file record shows a data record that has two fields of character data followed by a field of character BYTE or TEXT data, a field of hexadecimal byte or text data, and the path name of a file that contains BYTE or TEXT data.

```
field1|field2|TEXT data|BEEEF6699|/bbs/kaths/data2jn95
```

The following figure shows a format for the sample data-file record previously illustrated.
Figure 1. Delimited format with BYTE or TEXT entries



Create a delimited format with extended data types

Extended data types include the following data types:

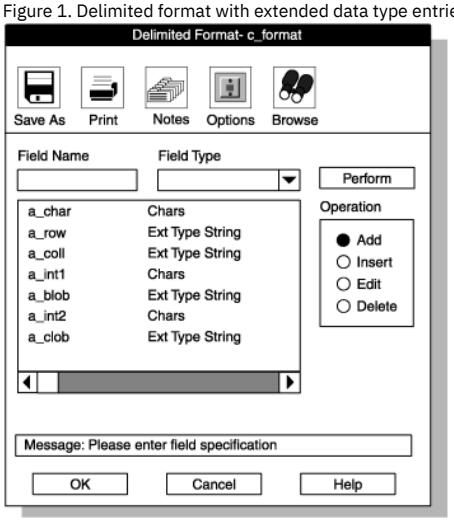
- User defined (distinct and opaque)
- Collection
- CLOB and BLOB
- Row

In a delimited format, CLOB and BLOB data is always written to a file. CLOB data can be ASCII or hexadecimal data. BLOB data can be binary data. The path name of the file, the file size, and the offset are embedded in the data file during unload. However, when you perform a load, you only need to specify the path name.

The following sample data-file record shows a data record that has a field of character data (field1), a row-type field (ROW('abcd', NULL)), a collection-type field (SET{1}), an integer field (10), a BLOB-type field (/work/data/photo.jpg), an integer field (20), and a CLOB-type field (work/data/text.txt).

```
field1|ROW('abcd', NULL|SET{1}|10|/work/photo.jpg|20|
/work/text.txt
```

The following figure shows a sample format for the sample data-file record that was previously illustrated.
Figure 1. Delimited format with extended data type entries



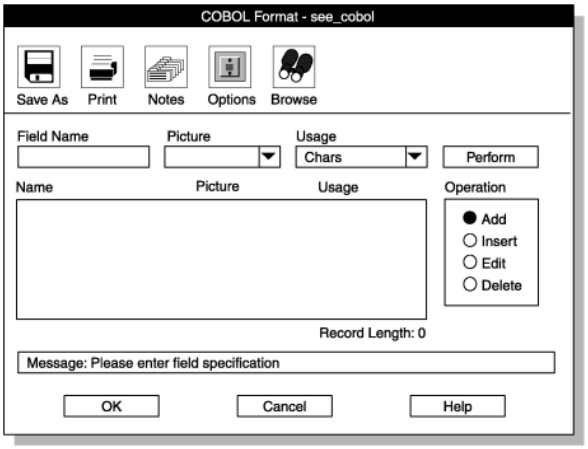
If you unload by using a format that is similar to Figure 1, the unloaded data might resemble the following sample data-file record. You can use this data to load the same file again, instead of using the path name.
Figure 2. Sample data-file record that includes extended data types

```
field1|ROW('abcd', NULL|SET{1 }|10|0,51f4,blob67e9.8ac|20|
0,c 692,clob67e9.9ad
```

By default, the clob67e9.8ad and blob67e9.8ac files in Figure 2 are written to /tmp. To change the default, modify the path in the PLOAD_LO_PATH environment variable.

COBOL records

The HPL supports COBOL sequential data files that do not contain internal indexing. The following figure shows the COBOL-Format Definition window for preparing a COBOL format.
Figure 1. Fixed-format definition window for a COBOL format



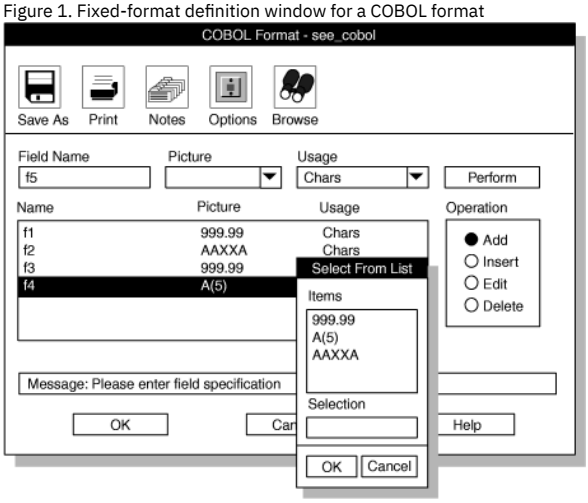
- Create a COBOL format
- Picture and usage descriptions
  The picture and usage description must conform to ANSI-COBOL 85 specifications.

Create a COBOL format

To create a format for COBOL records, follow the same steps as in Creating a fixed format, with the following modifications:

- In step 3, click COBOL in the Type group in the Record Formats window.
- In step 8, type the COBOL picture description in the Picture text box.
- In step 9, type the data type in the Usage text box.
- The arrow displays the selection list of available data types for the Usage text box.

Figure 1. Fixed-format definition window for a COBOL format



Important: Ext types are not supported in COBOL format.

Picture and usage descriptions

The picture and usage description must conform to ANSI-COBOL 85 specifications.

For information about COBOL picture strings, see the documentation for the COBOL compiler.

Picture description

The picture description must match the record file descriptor (FD) from the COBOL program that generates or use the data. For information about COBOL formats, see your COBOL programmer's publication.

Usage description

The usage description must match the data-field type described in the FD descriptor of the COBOL program. If the COBOL program does not include a usage clause, select the Chars (character) option for the usage.

Packed-decimal conversions

When values are converted to packed-decimal formats, supply a picture clause that matches the picture clauses in the COBOL programs that use the data. Otherwise, the COBOL interpretation of the values is wrong.

The following table lists some examples of appropriate picture clauses.

Picture	Input data	Output data	COBOL value =
9999999	123	0000123C	123
9999V99	123	0000123C	1.23
9999.99	123	0000123C	1.23
9999V99	-123.22	0012322D	-123.22

Other formats

In addition to delimited, fixed, and COBOL formats, the High-Performance Loader (HPL) provides two other formats for loading and unloading data: fast format and fast job. These formats are not included on the Record Formats window because the format specifications are predefined; you do not need to make any choices.

Fast format and fast job are the most efficient ways to load and unload data because their formats are predefined.

Fast format

Fast format loads or unloads data in which each individual column uses IBM® Informix® internal format. You can reorder, add, or delete columns, but you cannot conduct conversion on the column itself.

Select Fixed internal in the Generate window to get this type of load. For information about the Generate window, see The Generate options of the ipload utility.

Fast job

A fast job loads or unloads an entire row of the IBM Informix database table in internal format. A fast job is also called a raw load or a no conversion job. For more information, see Format type group.

The **-fn** flag of the **onpload** command-line utility specifies a fast job. For information about the **onpload** utility, see The onpload utility.

Format options

The format options let you change the default driver, the character set, the default computer type, and the delimiters. Information about the format options is stored in the **formats** table of the **onpload** database.

- Modifying fixed and COBOL formats
- Modifying delimited-format options

Related concepts:
Formats of supported datafile records
Related reference:
The formats table in the onpload database
Format type group

Modifying fixed and COBOL formats

You can modify the following options for fixed and COBOL formats.

Character set
: The code set that is used to translate the data in the data table

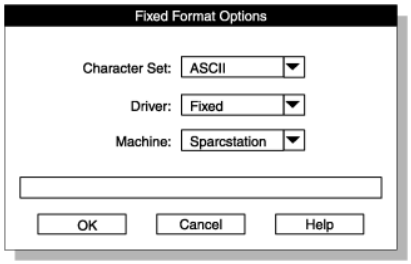
Driver
: The driver that is used with the delimited format. For more information, see Selecting a driver.

Machine
: The machine type that produced the data files. For more information, see Modify the machine description.

For a fixed format, you can select the GLS code set you want from the Character Set selection list. For information about locales and code sets, see the *IBM® Informix® GLS User's Guide*.

To modify the options for fixed and COBOL formats:

1. Display the Format-Definition window for the format that you want.
   To display the window, follow the steps in Creating a fixed format.
2. Click Options.
   The Options window (in this example, the Fixed Format Options window) appears, as following figure shows.
   Figure 1. The Fixed Format Options window



3. Modify the options as appropriate.
4. Click OK to save your options and return to the Format-Definition window.

Modifying delimited-format options

You can modify the following options of the delimited format from the Delimiter Options window.

Character set
: The code set used to translate the data in the data table.

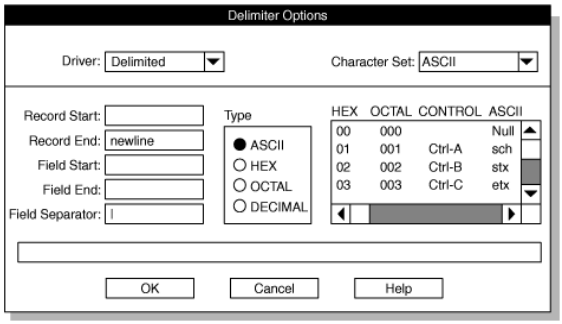
Driver
: The driver that is used with the delimited format. For more information, see Selecting a driver.

Delimiting characters
: The delimiting characters, which are sometimes called record separators and field separators, indicate the beginning and end of records and fields. You can specify the delimiting characters in ASCII, HEX, OCTAL, or DECIMAL format.

You can select the GLS code set you want from the Character Set selection list. For information about locales and code sets, see the *IBM Informix GLS User's Guide*.

To modify the options for delimited formats:

1. Display the Delimited Format Definition window.
   To display the window, complete the steps in Creating a fixed format with the following modification: in step 3, click Delimited.
2. Click the Options button in the Delimited Format Definition window.
   The Delimiter Options window appears, as the following figure shows.
   Figure 1. The Delimiter Options window



3. Modify the options that you want to change, or add information.
   For example, add Field Start and Field End information to mark the beginning and end of a field (column).

   The default escape character is **** (the backslash).

   If the **Field Start** and **Field End** values contain both single or double quotation marks and specified characters are not present in a field, an error occurs. The error occurs because the input does not match the expected layout. If one of these characters appears in a field, you must mark the character with the escape key (to escape the character).

   For comma-separated values (CSV), enter , (a comma) in the Field Separator field. For more information, see Testing the import of a CSV file.

   An error can also occur if the **Field Separator** value appears inside a quoted field or any special character (such as the end field, end record, or escape character) appears in a field. In this case, you must also escape the character.

4. Click OK to save your changes and return to the Delimited Format Definition window.

Tip: You can use the DBDELIMITER environment variable to set the field delimiter for the **dbexport** utility and the LOAD and UNLOAD statements. However, do not use DBDELIMITER with the HPL because the **onpload** utility does not use this environment variable.
For more information about environment variables, see the *IBM Informix Guide to SQL: Reference*.

- Testing the import of a CSV file
  This procedure shows how to test importing sample data from a spreadsheet into IBM Informix.

Related concepts:
Delimited records
Related reference:
The delimiters table in the onpload database

Testing the import of a CSV file

This procedure shows how to test importing sample data from a spreadsheet into IBM® Informix®.

Files containing comma-separated values (CSV) are a commonly used for transferring simple text data between programs. The CSV format uses a comma as a field separator and a new line as a record separator. Double quotation marks are used to embed commas, new lines, or double quotation marks within strings. Some applications (for example, some spreadsheet applications) provide the option of exporting text data in the CSV format.

To test importing a CSV file:

1. Create a test database and simple table in .
2. In your spreadsheet application:
   a. Create a spreadsheet with columns that correspond to the columns in your table.
   b. Populate the columns with some test data.
   c. Export the data by saving it in .csv format.
3. Copy the resulting .csv file and paste it in a location that you can access from **ipload**.
4. Start **ipload** and choose Components > Generate Job to create a job.
5. In the Generate Job window:
   a. Select Load/Unload Job in the Generate group.
   b. Select Delimited in the Format Type group.
   c. Specify the .csv file as the source for input to the new job you are creating. Specify a name in the Generate Name field, select your database in the Database field, select you table in the Table field, and then insert the full path to the .csv file in the Device field.
   d. Click OK.
6. Load the job by choosing Jobs > Load.
7. While you can do many things (such as filtering and mapping) on the Load Job window, for this test, select the Format button.
   The Delimited Format window appears. You use this window to define the format of the input file.
8. In the Delimited Format window, click the Options button to display the Delimited Options window.
9. In the Delimiter Options window, change the value in the Field Separator field to , (a comma) and click OK.
   The **ipload** utility returns you to the Delimited Format window.
10. In the Delimited Format window, click OK.
    The **ipload** utility returns you to the Load Job window.
11. Click Run to run the job.
    The **ipload** utility displays status information while the job runs, and then it displays the results of the job.
12. Click OK twice to return to the main **ipload** window.

As an evaluation, you can use DBAccess to verify that the data was successfully loaded.

For more information, see Generating load and unload components.

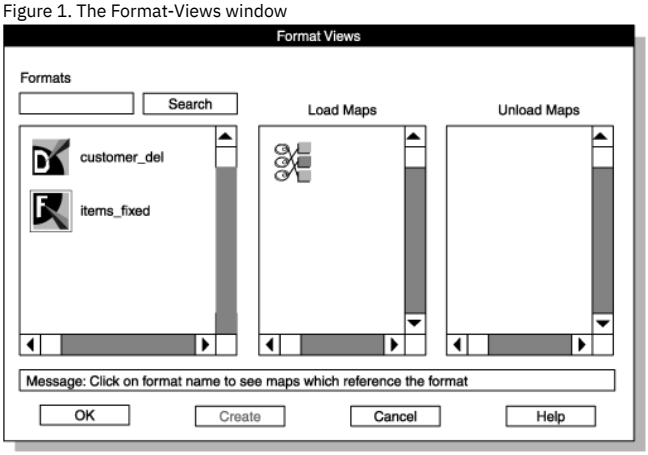
Format Views window

You can display a list of the formats and load and unload maps that are associated with a project from the Format Views window. You can also create or edit a format.

The Format Views window appears in the following situations:

- When you click Format in the Unload Job window and no format name is in the Formats text box
- When you click Search in the Query window

The following figure shows a Format Views window. Views windows discusses how to use Views windows.
Figure 1. The Format-Views window



Define queries

This section describe how to define queries including how to create, edit, and export and import queries.

- HPL queries
  You can build an SQL statement with the **ipload** query component.
- Creating a query
  Use the Query window to create a query.
- Editing a query
  To edit a query, follow the same steps as for creating a query, but open an existing query in the Query window.
- Exporting and importing queries
  You can use the File button on the Query-Definition window to export a query to a file or to import a query that you prepared with some other tool.
- The Database Views window
  You can display a list of the queries, maps, and formats that are associated with a project from the Database Views window. You can also create or edit a query.

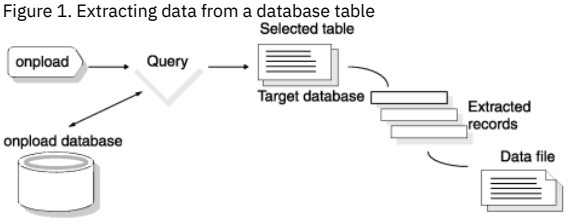
Related concepts:
Components of the unload job

HPL queries

You can build an SQL statement with the **ipload** query component.

The **ipload** utility stores query information in the **query** table of the onpload database. The SQL statement is stored as TEXT data.

During the unload process, the High-Performance Loader (HPL) uses a query to select data from a database table (or tables), as the following figure shows. The HPL can process any valid SQL statement.
Figure 1. Extracting data from a database table



These topics use the term *query* in two ways:

- To refer to the SQL statement that selects information from the database
- To refer to the HPL component that lets you build and store the SQL statement

Related reference:
The query table in the onpload database

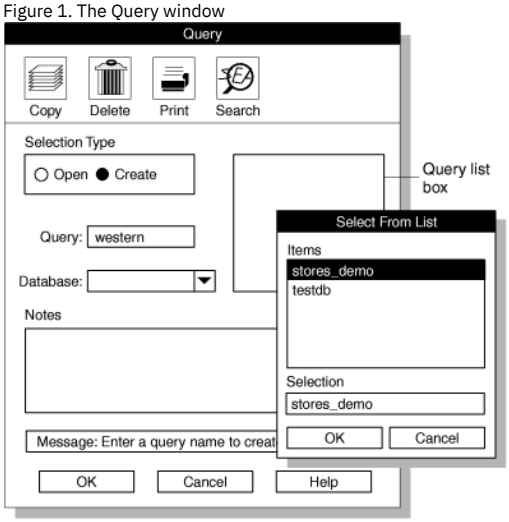
Creating a query

Use the Query window to create a query.

To create a query:

1. Choose Components > Query from the HPL main window.
   The Query window appears, as Figure 1 shows.

2. Click Create in the Selection Type group.
3. Choose a name for your query and type it in the Query text box.
4. In the Database text box, type the name of the database that contains the tables from which you want to extract data. Or, click the down arrow to select from a database selection list.
   The following figure shows the Query window with the Query text box completed and **stores_demo** selected from the selection list.
   Figure 1. The Query window

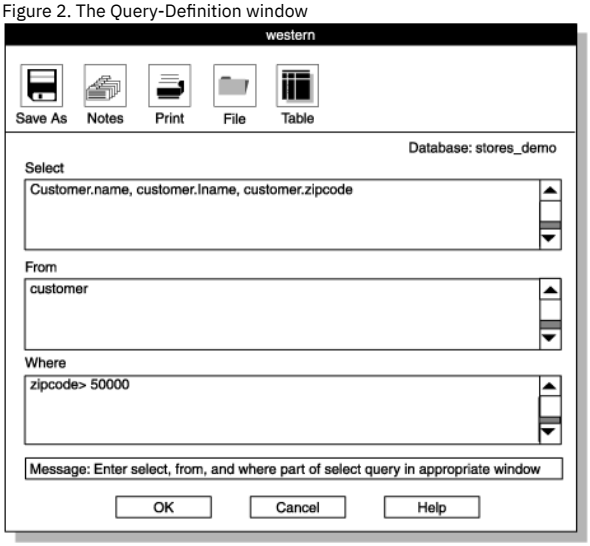


5. Click OK.
   The Query-Definition window appears, as Figure 2 shows. The name that you chose for your query appears in the title bar.

6. Type your query in the Select, From, and Where text boxes.
   The following figure shows the following simple query of the **customer** table of the **stores_demo** database:

```
SELECT customer.fname, customer.lname,
customer.zipcode
FROM customer
WHERE zipcode > 50000
```

   Figure 2. The Query-Definition window



   If you prefer, you can type the entire query into the Select text box. If you later edit the query, **ipload** divides the query into SELECT, FROM, and WHERE clauses.

7. Click OK to save the query and return to the Query window.
   The query that you just created now appears in the Query list box at the right side of the Query window.

8. Click Cancel to return to the HPL main window.

- Using the Column Selection window
  The Table button displays the Column Selection window. You can use the Column Selection window to build queries by selecting tables and columns. The **ipload** utility inserts the selected columns and tables into the appropriate text boxes of the Query-Definition window.
- Editing the WHERE clause
  When you use the Column Selection window to select a column or columns for the WHERE clause, the selected columns appear in the Where text box.

Related reference:
The query table in the onpload database

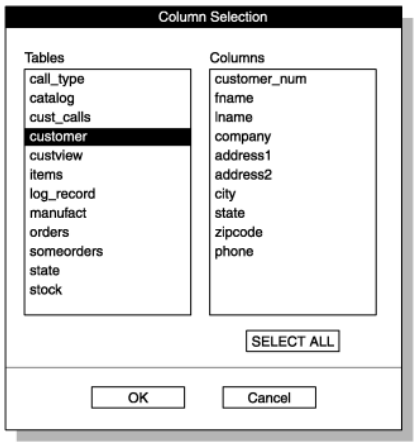
Using the Column Selection window

The Table button displays the Column Selection window. You can use the Column Selection window to build queries by selecting tables and columns. The **ipload** utility inserts the selected columns and tables into the appropriate text boxes of the Query-Definition window.

To use the Column Selection window:

1. Follow the steps in Creating a query to display the Query-Definition window.
2. Click Table.
   The Column Selection window appears, as Figure 1 shows. The Tables list box includes synonyms and views that are valid for the local database server.

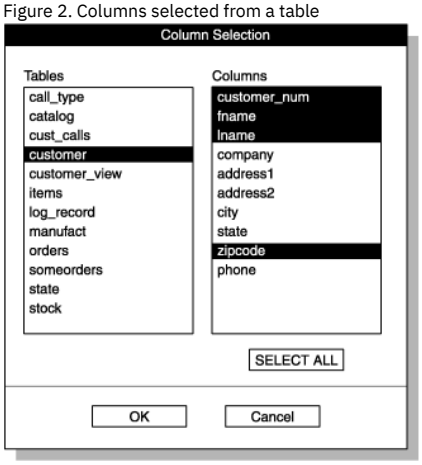
3. Select a table.
   After you select a table, the right pane displays a list of the columns in that table. The following query shows the Column Selection window with the **customer** table selected.
   Figure 1. The Column Selection window after selecting a table



4. Select one or more columns to use in the query.
   - To select a single column, select that column.
   - To select all columns, click Select All.
   - To select consecutive columns, select the first column. Move to the final column and hold down SHIFT while you select that column.
   - To select nonconsecutive columns, select a column. Hold down CONTROL while you select additional items.

   The following figure shows the Column Selection window with several columns selected.

   Figure 2. Columns selected from a table

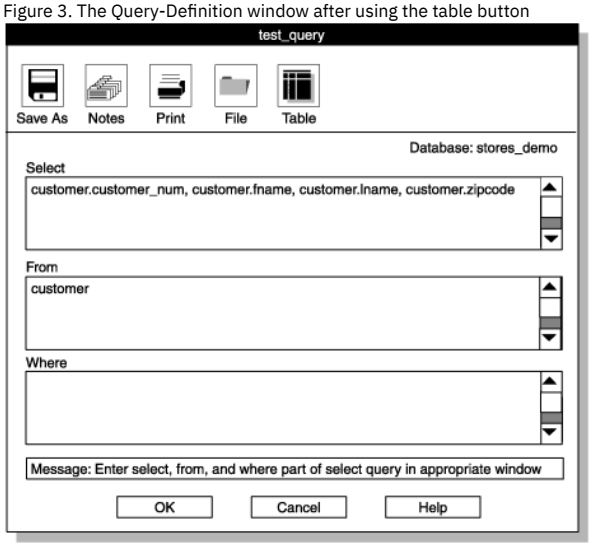


5. When you finish selecting columns, click OK to return to the Query-Definition window.
   When the Query-Definition window reappears, the mouse cursor changes to a pointing hand and the message line reads:

   ```
   Position Cursor Where Column Data to be Inserted
   ```

6. Select the Select text box or the Where text box.
   The following figure shows columns inserted into the Select text box. The **ipload** utility also inserts the table name into the From text box.

   Figure 3. The Query-Definition window after using the table button



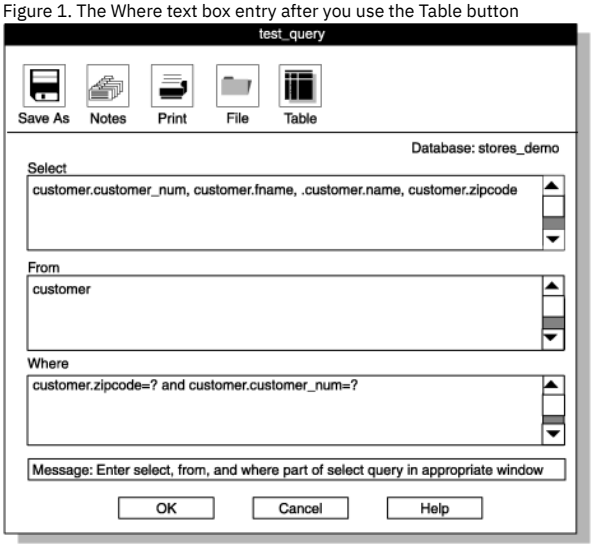
7. Repeat steps 2 through 6 to add columns from other tables.
8. Modify the text in the Where text box so that it is a valid WHERE clause. See Editing the WHERE clause.
9. Click OK to save the query and return to the Query window.
10. Click Cancel to return to the HPL window.

Editing the WHERE clause

When you use the Column Selection window to select a column or columns for the WHERE clause, the selected columns appear in the Where text box.

In the Where text box, as shown in the figure below, the `=?` symbols indicate where you must provide match conditions. The following figure shows the result when you choose **zipcode** and **customer_num** from the **customer** table.

Figure 1. The Where text box entry after you use the Table button



To edit the WHERE clause:

1. Select `=?` and change it to the match condition that you want.
   For example, the Where text box in Figure 1 contains the following text:

   ```
   customer.zipcode =? and customer.customer_num =?
   ```

   You must change both occurrences of `=?` to valid match conditions. You might change the text as follows:

   ```
   customer.zipcode > 50000 and
   customer.customer_num > 150
   ```

   For a full description of match conditions, see Match condition operators and characters.

2. Check the comparison operators.
   When you select multiple columns from the Column Selection window, **ipload** inserts `and` into the expression between each column. You might need to change `and` to `or`.

Editing a query

To edit a query, follow the same steps as for creating a query, but open an existing query in the Query window.

To edit a query:

1. Choose Components > Query from the HPL main window.
   The Query window appears, as Figure 1 shows.

2. Click **Open**.
3. Select a query from the list of queries.
4. Click OK.
   The Query-Definition window appears, as Figure 2 shows. The name of the query that you are editing appears in the title bar.

5. Modify the Select, From, and Where text boxes.
6. Click OK to save the modified query.
7. Click Cancel to return to the HPL main window.

Exporting and importing queries

You can use the File button on the Query-Definition window to export a query to a file or to import a query that you prepared with some other tool.

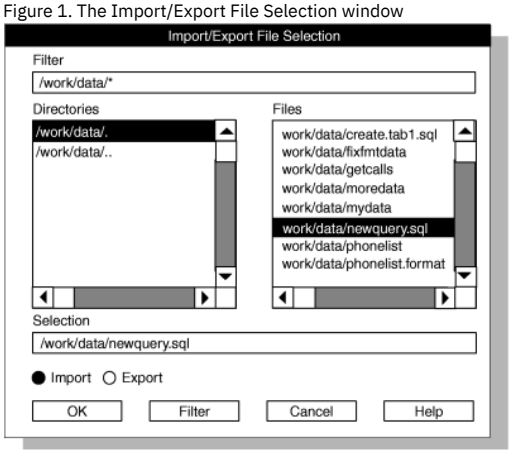
For example, you might use DB-Acess to prepare and test a query and to save the query to a file. You can then import that query into the High-Performance Loader (HPL).

- Importing a query
  You can use the Import/Export File Selection window to import a query that you prepared outside of the High-Performance Loader (HPL).
- Exporting a query
  The File button also allows you to export the query as an SQL statement.

Importing a query

You can use the Import/Export File Selection window to import a query that you prepared outside of the High-Performance Loader (HPL).

To import a query:

1. Display the Query-Definition window by following the steps in Creating a query.
2. Click File.
   The Import/Export File Selection window appears, as the following figure shows.
   Figure 1. The Import/Export File Selection window


3. Click Import.
4. Specify the file that you want to import.
   a. Type a path name and appropriate wildcards in the Filter text box and click Filter. Use an asterisk (*) to list all of the files in the directory. Then select a file and click OK or double-click a file name.
   b. Type the full path name in the Selection text box and then click OK.

   The text from the imported file appears in the Query-Definition window.

   If **ipload** can interpret the SQL statement, the SQL statement is inserted into the appropriate Select, From, and Where text boxes.

   If **ipload** cannot interpret the SQL statement, the entire content of the imported file appears in the Select text box.
5. Edit the query so that it meets your needs.
6. Click OK.
   If the query is a valid SQL query, the display returns to the Query window.

   If the query is not a valid SQL query, **ipload** highlights the portion of the query that it cannot interpret and provides an error message.
7. From the Query window, click Cancel to return to the HPL main window.

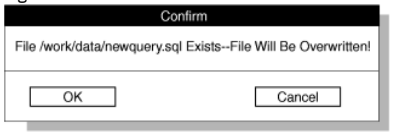
Exporting a query

The File button also allows you to export the query as an SQL statement.

You can prepare a query for export in the following ways:

- Create a query. (See Creating a query.)
- Open an existing query.
- Import an already prepared query and modify it. (See Importing a query.)

To export a file:

1. Follow the steps in Creating a query to prepare a query in the Query-Definition window (see Figure 2).
2. Click File.
   The Import/Export File Selection window appears (see Figure 1).
3. Click Export.
4. Select the directory and file where the query is to be stored.
   a. Add the name of a new file to a path name in the Selection text box and click OK.
   b. Type a path name and appropriate wildcard in the Filter text box and click Filter. Then select a file name.
5. Click OK.
   If the file that you specified exists, **ipload** asks if you want to overwrite the existing file, as the following figure shows.
   Figure 1. The Confirm File Overwrite window


6. You now have two choices:
   - Click OK to overwrite. The display returns to the Query window.

- Click Cancel to choose a different file name.

The **ipload** utility writes the text from the Select, From, and Where text boxes into the specified file as a single SQL statement.

7. Click OK. The display returns to the Query window.

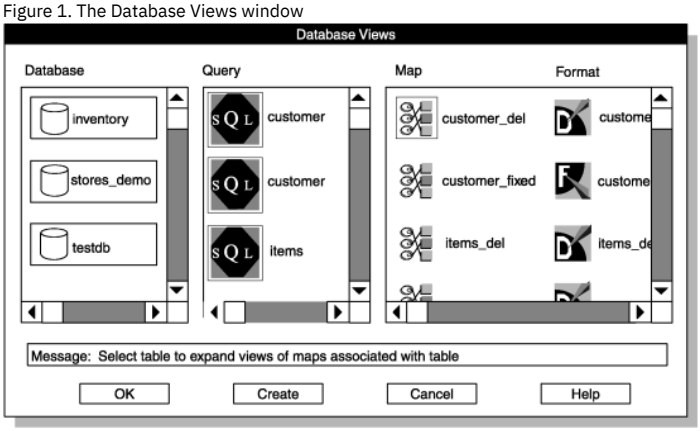
The Database Views window

You can display a list of the queries, maps, and formats that are associated with a project from the Database Views window. You can also create or edit a query.

The Database Views window appears in the following situations:

- When you click Query in the Unload Job window and no query name is in the Query text box
- When you click Search in the Query window

The following figure shows the Database Views window. Views windows discusses how to use Views windows.
Figure 1. The Database Views window



Define maps

This section describes how to define maps with the High-Performance Loader (HPL). It also describes the options that are available for defining maps.

- Load and unload maps
  You can use the **ipload** utility to build a *map*. A map specifies the relationship between the fields of a data file and the columns of a database.
- Unload maps
  An unload map associates columns extracted from a database by a query with the fields in a data-file record. You can create an unload map from the Load Job window or from the Components menu of the HPL main window. After you define an unload map, you use it with the Unload Job window or the **onpload** utility.
- Mapping options
  The mapping options define conversions that **onpload** applies to the data before it inserts the data into the database (for a load job) or into the data file (for an unload job). These conversions can include case conversion, text justification, data masking through picture strings, default values, and fill characters. The mapping options also allow you to replace imported data with data from other database tables.
- Editing options
  This section discusses specialized options in the Map-Definition window.
- Map Views window
  You can display a list of the components that are associated with a database in a specific project from the Map Views window. You can also create or edit a map.

Related concepts:
Inline data
Data in a separate file
Simple LO data in a separate file
Components of the unload job
Related reference:
Inline data

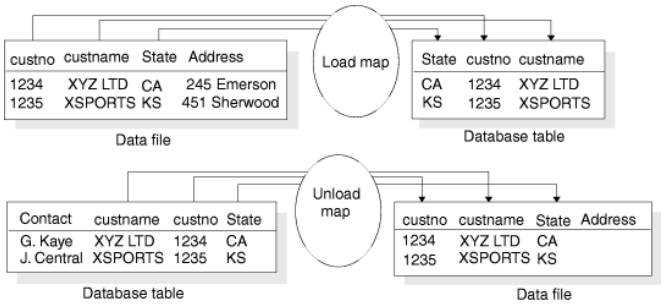
Load and unload maps

You can use the **ipload** utility to build a *map*. A map specifies the relationship between the fields of a data file and the columns of a database.

To load data into a database, you define a load map, which associates fields from records in a data file to columns in a database table.

To unload data, you define an unload map. The unload map associates the columns that a query retrieves from one or more tables to the fields in a data file.

The following figure shows the relationship between load and unload maps.
Figure 1. Using a map



The **ipload** utility stores information about maps in the **maps**, **mapitem**, **mapoption**, and **mapreplace** tables of the **onpload** database.

You can define a map at any time. After you define a map, you use it with the Load Job window or the **onpload** utility.

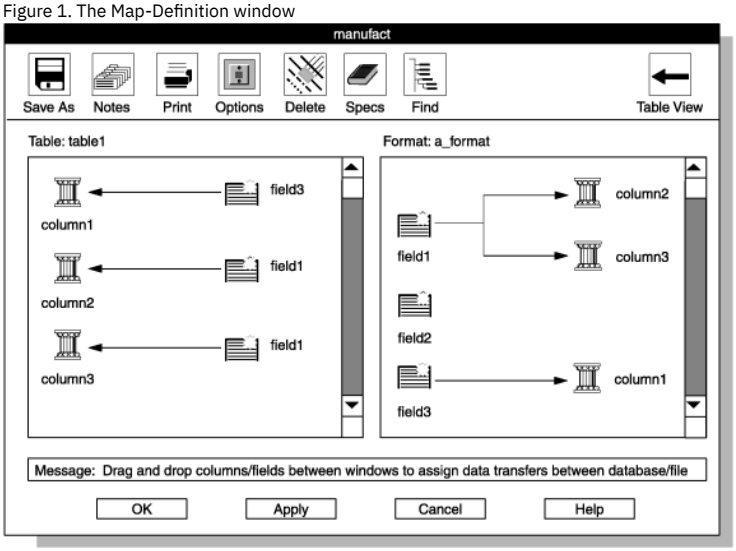
- The Map-Definition window
  You can associate an input item with a table column from the Map-Definition window.
- Creating a load map
  You can create a load map from the Load Job window or from the Components menu of the High-Performance Loader (HPL) main window.

Related reference:
The onpload database
Define maps
The mapitem table in the onpload database
The maps table in the onpload database

The Map-Definition window

You can associate an input item with a table column from the Map-Definition window.

The following figure shows a Map-Definition window for a load map.
Figure 1. The Map-Definition window



The map specifies which fields of the data file are loaded into database columns. The data moves from the fields of a data file into the columns of a database.

- The Table and Format panes
  The Map-Definition window contains two panes: the Table pane and the Format pane.
- Unassigned or multiple-assigned fields and columns
- Identical field names and column names
  When you create a format, you can assign arbitrary names to the fields of the data file. You might find it convenient to assign names that correspond to the names of the columns in the database. When you create a map, **ipload** automatically links columns and fields that have the same name and type, thus saving you work.

The Table and Format panes

The Map-Definition window contains two panes: the Table pane and the Format pane.

The window has two panes so that you can take the following actions:

- Scroll the panes to see all of the columns or fields of a long data file or database table.
- Connect an input field to more than one column.

The left column of icons in each pane represents the active elements of the display. These left columns do not change. In a load map, the columns in the Table pane receive the input. In the Format pane, data from the fields moves into the columns of the database table.

The right column of icons in each pane represents the associations that you make. These columns change as you build the map. A field might be listed more than once in the right column of the Table pane because you can store a field from the data file in more than one database column. This field is mapped (with a split arrow) to two columns in the Format pane. A column never appears more than once in the list to the right of the Format pane because a column can only receive input from one database field.

By scanning the left pane, you can easily see which columns are receiving data from the data file. By scanning the right pane, you can see which fields of the data file are providing data and which fields are not being used.

Unassigned or multiple-assigned fields and columns

The High-Performance Loader (HPL) does not require a one-to-one connection between the fields and columns. You can map a field to multiple columns. Figure 2 shows a map where the data from one field is placed into two columns.

You can also have a column that has no mapping association. Field 1 in the Format pane in Figure 2 does not have an association. If a column does not receive input, **onpload** sets the column to null.

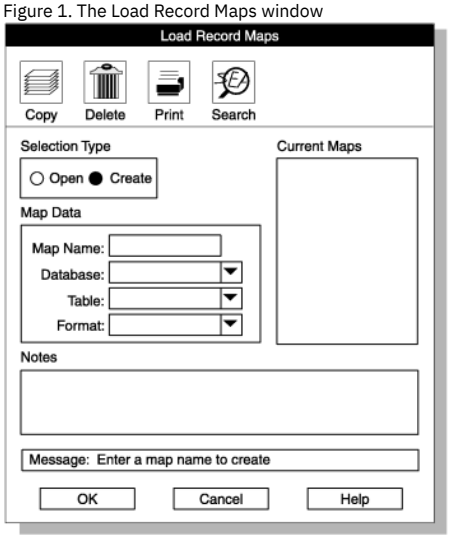
Identical field names and column names

When you create a format, you can assign arbitrary names to the fields of the data file. You might find it convenient to assign names that correspond to the names of the columns in the database. When you create a map, **ipload** automatically links columns and fields that have the same name and type, thus saving you work.

Creating a load map

You can create a load map from the Load Job window or from the Components menu of the High-Performance Loader (HPL) main window.

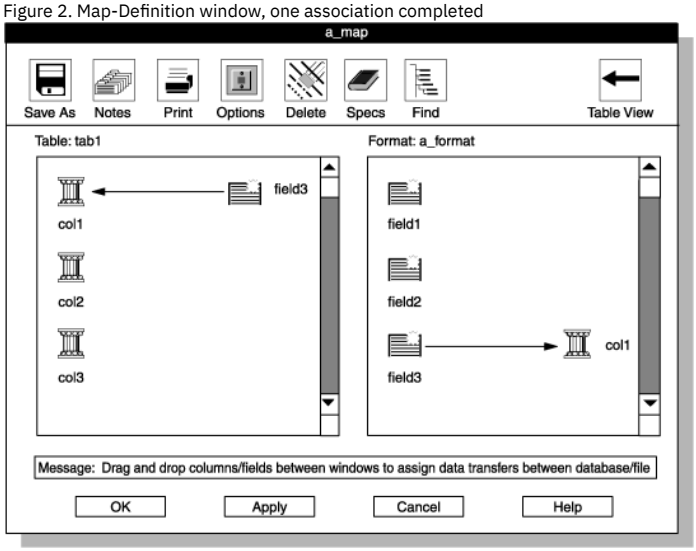
Before you can create a load map, you must create a format that describes the data file that you plan to load. For information about how to create a format, see Define formats.
Important: The HPL does not support conversion from extended type data and smart-large-object data (Ext Type data types) to non-Ext Type data types. A field that is defined as an Ext Type data type can be mapped only to an Ext Type column. For more information about Ext Type data types, see Data types allowed in a fixed format or Data types allowed in a delimited format
To create a load map:

1. Choose Components > Maps > Load Map from the HPL main window.
   The Record Maps window appears, as the following figure shows.
   Figure 1. The Load Record Maps window
   

2. Click Create in the Selection Type group.
3. Choose a name for the map and type it in the Map Name text box.
4. Type the names of the database and table where the data will be loaded in their corresponding text boxes.
   You can also click the down arrow to choose the names from a selection list. The Tables selection list includes synonyms that are valid for the local database server.
5. Type the format that describes the data file in the Format text box.
   You can also click the down arrow to choose the format from a selection list.
6. Click OK to open the Map-Definition window.
   A Map-Definition window similar to Figure 2 appears.
7. Click a column icon in the left column in the Table pane and hold the mouse button down. A box appears around the icon and its name.
8. Drag the box to a field icon in the Format pane.
   When you connect columns to fields, it does not matter whether you drag a column to a field or drag a field to a column, but you must always connect items from the left column of each pane.

   The following figure shows a Map-Definition window with this step completed.

Figure 2. Map-Definition window, one association completed



9. Repeat steps 7 and 8 for each field that you want to transfer into the database.
10. Add the options that you want, if any. For instructions, see Defining the mapping options.
11. Click OK to return to the Load Record Maps window.

Related reference:
Create a fixed format that uses carriage returns

Unload maps

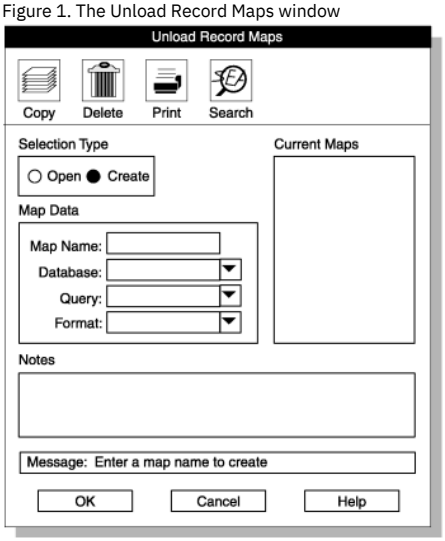
An unload map associates columns extracted from a database by a query with the fields in a data-file record. You can create an unload map from the Load Job window or from the Components menu of the HPL main window. After you define an unload map, you use it with the Unload Job window or the **onpload** utility.

- Creating an unload map
- Unload data by using functions
  If you use a function in a query to unload data, you must associate a name with the result of that function.

Creating an unload map

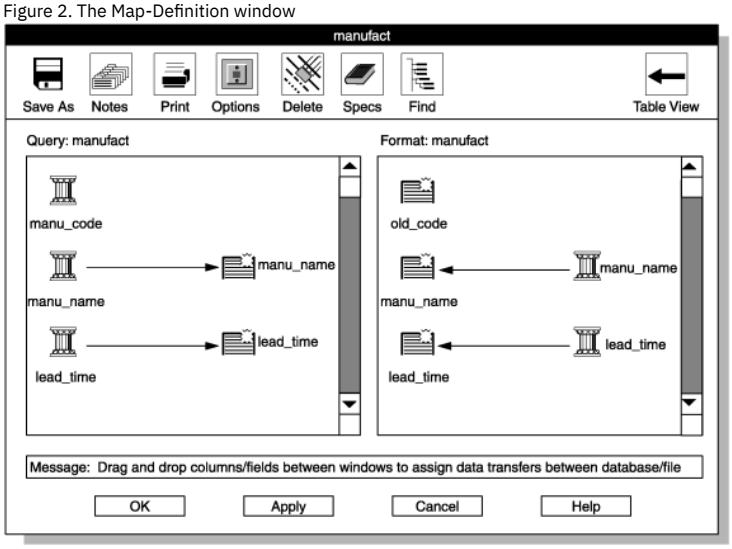
Before you can create an unload map, you must define a query on the table that will be unloaded. For instructions on how to define a query, see HPL queries.
Important: The HPL does not support conversion from extended type data and smart-large-object data (Ext Type data types) to non-Ext Type data types. An Ext Type column can be mapped only to an Ext Type field. For more information about Ext Type data types, see Data types allowed in a fixed format or Data types allowed in a delimited format.
To create an unload map:

1. Choose Components > Maps > Unload Map from the HPL main window.
   The Unload Record Maps window appears, as the following figure shows.
   Figure 1. The Unload Record Maps window



2. Click Create in the Selection Type group.
3. Choose a name for the map and type the name in the Map Name text box.
4. Type the name of the database in the Database text box. You can click the down arrow to choose a database from a selection list of databases.

5. Type the name of a query in the Query text box. You can click the down arrow to choose a query from a selection list of queries.
6. Type the format name in the Format text box. You can click the down arrow to choose a format from a selection list of formats.
7. Click OK.
   A Map-Definition window is similar to the following figure. In this figure, some of the field names match column names. The **ipload** utility automatically maps columns to fields of the same name. The direction of the arrows indicates the flow of data, as shown.
   Figure 2. The Map-Definition window



8. To map a database column to a data-file field, click the database-column icon. Drag the column to the data-file field icon that you want. An arrow links the column icon to the field icon.
9. Repeat step 8 until you have mapped all the columns you want to fields.
10. Define any mapping options as appropriate. For information about mapping options, see Mapping options.
11. Click OK to save the map and return to the Unload Record Maps window.
12. Click Cancel to return to the HPL main window.

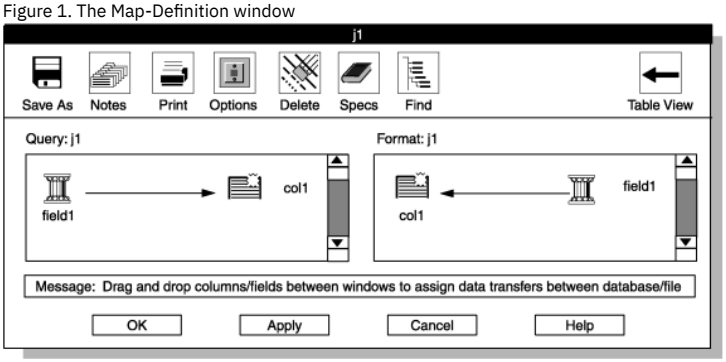
Unload data by using functions

If you use a function in a query to unload data, you must associate a name with the result of that function.

In the following example, the returned value of the function TRIM is assigned the name `field1`.

```
SELECT TRIM(col1) field1 FROM tab1
```

After submitting the query, you must attach `field1` to `col1` of the unload file manually, as the following figure shows.
Figure 1. The Map-Definition window



Mapping options

The mapping options define conversions that **onpload** applies to the data before it inserts the data into the database (for a load job) or into the data file (for an unload job). These conversions can include case conversion, text justification, data masking through picture strings, default values, and fill characters. The mapping options also allow you to replace imported data with data from other database tables.

The information from the Mapping Options window is stored in the **mapoption** table of the onpload database.

- Defining the mapping options
  This procedure describes how to specify mapping options.
- Set the mapping options
  You can set as many of the choices on the Mapping Options window as you need.

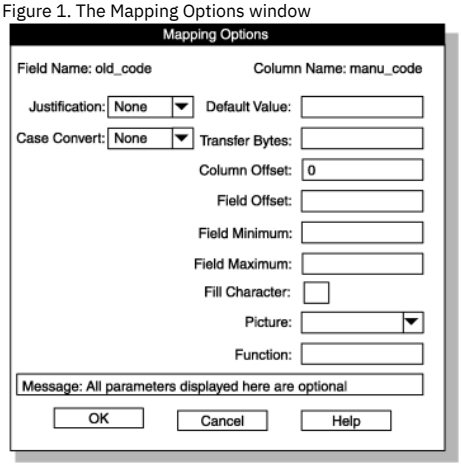
Related reference:
The onpload database

Defining the mapping options

This procedure describes how to specify mapping options.

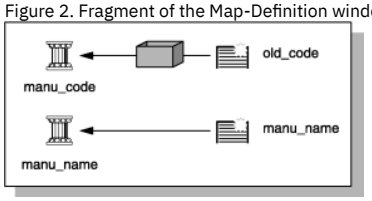
To define mapping options:

1. Display the Map-Definition window by following the steps for Creating a load map, or Creating an unload map.
2. Select the field or column (in the right column of a pane) that you want to modify.
3. Click Options.
   The Mapping Options window appears, as the following figure shows.
   Figure 1. The Mapping Options window



4. Change the options that you want.
5. When you have set all the options that you want, click OK to return to the Map-Definition window.

When you return to the map window, an options symbol (a small box) appears between the field and the column, as the following figure shows. The options symbol indicates that mapping options are in effect.
Figure 2. Fragment of the Map-Definition window showing an options symbol



Related reference:
The mapoption table in the onpload database

Set the mapping options

You can set as many of the choices on the Mapping Options window as you need.

Justification option

The Justification option positions text within a record. You can justify the text to the left or right, or you can center it.

Case Convert option

The Case Convert option converts the case of the data to the selected case. The High-Performance Loader (HPL) supports upper, lower, and proper-name conversions. For example, you can make the following conversions.

Input	Conversion type	Result
JOHN LEE SMITH	Proper Name	John Lee Smith
john lee smith	Proper Name	John Lee Smith
john lee smith	Upper	JOHN LEE SMITH
JOHN LEE SMITH	Lower	john lee smith

Default Value option

The Default Value option specifies the value that is inserted into the column when no field is mapped into that column.

Transfer Bytes option

The Transfer Bytes option specifies the number of bytes in the record field to transfer to the database column.

For variable-length format records, this number reflects the maximum size of the field. The actual number of bytes to transfer is determined by the record or field delimiters.

Column Offset option

The Column Offset option specifies the offset from the beginning of a column field at which to start transferring the data from the field of the data record. Offsets are zero based.

Field Offset option

The Field Offset option specifies the offset from the beginning of a record field at which to start transferring data to the column. Offsets are zero based.

Field Minimum and Field Maximum options

The Field Minimum and Field Maximum options specify the smallest and largest acceptable values for a numeric column. If the data in the field is outside that range, the HPL rejects the record. This option is available only for fields with numeric formats, such as integer, short, or float.

Fill Character option

The Fill Character option lets you specify a character that you use to pad the contents of a field. The fill character can be any character that you can type on the keyboard. You can specify a fill character for fixed ASCII and COBOL loads or unloads. The fill character is filled in as a trailing character.

Picture option

The Picture option lets you reformat and mask data from the field of a record before the data is transferred to the database. Picture strings, explains picture strings.

Function option

The Function option specifies a user-defined function that is called for every record that is processed. You must add the function to the dynamically linked library. For information about using custom functions, see the API interface documentation in Custom-conversion functions.

Editing options

This section discusses specialized options in the Map-Definition window.

- Using the Delete button
  You can break the association between a column and a field with the Delete button in the Map-Definition window.
- Using the Find button
  You can find a column or field in a pane with the Find button in the Map-Definition window. The **ipload** utility scrolls the selected item into view and puts a box around it. This option is useful when the list of columns or fields is so long that the pane cannot display all of the items.
- Using the Specs button
  With the Specs button, you can display the Specifications window, where you can examine the characteristics of the columns and fields in your map.

Using the Delete button

You can break the association between a column and a field with the Delete button in the Map-Definition window.

To use the Delete button:

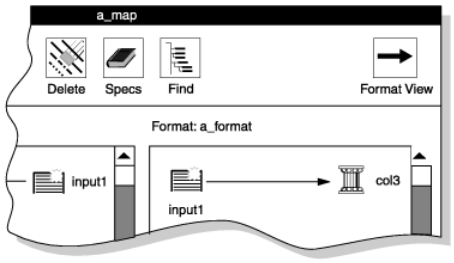
1. Click an icon in the right column of either of the panes in the Map-Definition window.
2. Click Delete to remove the arrow that connects the item to another item.

Using the Find button

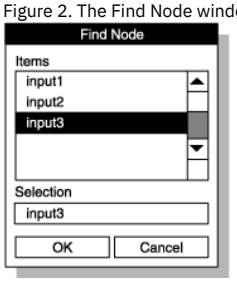
You can find a column or field in a pane with the Find button in the Map-Definition window. The **ipload** utility scrolls the selected item into view and puts a box around it. This option is useful when the list of columns or fields is so long that the pane cannot display all of the items.

To use the Find button:

1. In the Map-Definition window, select either the Table pane or the Format pane.
   When you select a pane, the view indicator in the upper right corner of the window changes to show which pane you selected. The following figure shows the upper portion of the Map-Definition window after you select the Format pane.
   Figure 1. The view indicator



2. Click Find.
   The Find Node window appears, as the following figure shows.
   Figure 2. The Find Node window



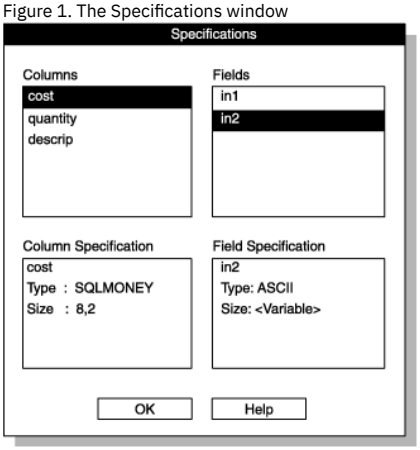
   Because the view indicator shows Format View, the Find Node window lists the fields of the data file. To see the columns of the database table, make sure that the view indicator shows Table View.

3. To select the item to find, you can use either of these methods:
   - Scroll through the list box to locate the item that you want to find and then select the item.
   - Type the name of the item that you want to find in the Selection text box.
4. Click OK. The Map-Definition window appears again. The selected field or column is highlighted with a box.

Using the Specs button

With the Specs button, you can display the Specifications window, where you can examine the characteristics of the columns and fields in your map.

The following figure shows a sample Specifications window.
Figure 1. The Specifications window



To use the Specifications window:

1. Click Specs in the Map-Definition window to display the Specifications window.
2. Select a column from the Columns list box or a field from the Fields list box or both.
   The specification boxes in the lower part of the screen display the characteristics of the selected items.
3. When you finish examining the specifications, click OK to return to the Map-Definition window.

The Specifications window displays the attributes of columns and fields. The Specifications window does not allow you to edit the attributes it displays. To change the attributes of a field, you must modify the format of the data file. (See Format options.) To change the attributes of a column, you must use appropriate SQL statements to modify the database table.

Map Views window

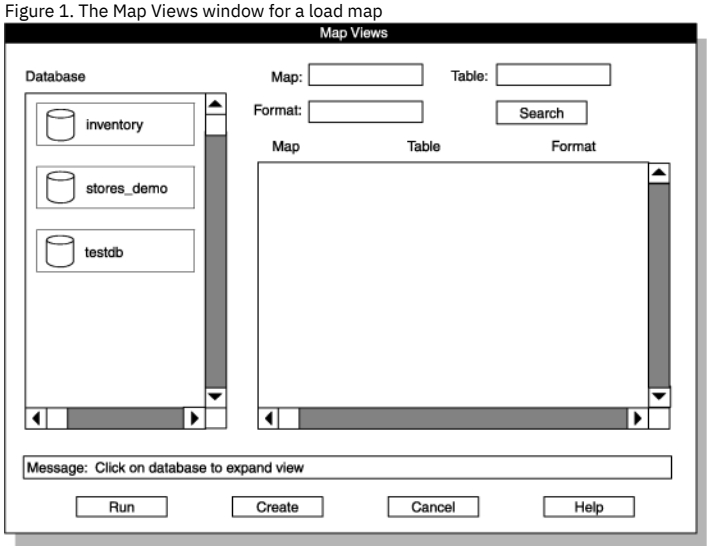
You can display a list of the components that are associated with a database in a specific project from the Map Views window. You can also create or edit a map.

The Map Views window appears in the following situations:

- If you click Map in the Load Job or Unload Job window when no map name is in the Map text box

- If you click Search in the Load Record Maps or Unload Record Maps window

The following figure shows the Map Views window for a load map.
Figure 1. The Map Views window for a load map



- Seeing the load maps of a database
- Seeing selected load maps

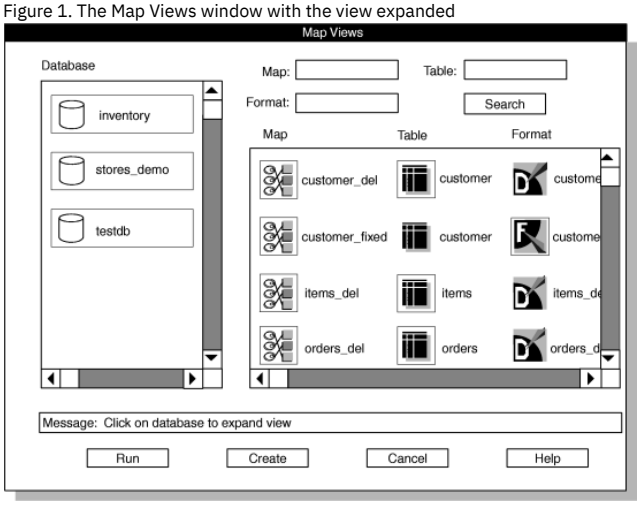
Seeing the load maps of a database

To see the load maps of a database:

1. Select a project in the HPL main window.
2. Choose Components > Maps from the HPL main window.
3. Choose Load > Map or Maps > Unload Map.
4. After the Load Record Maps or Unload Record Maps window appears, click Search.
   The Map Views window appears, as Figure 1 shows.

5. Select a database.
   The **ipload** utility displays a list of the maps associated with that database, as the following figure shows. The Table and Format columns show the database column and the format associated with each map.

   If you want to edit a specific map or format, click its button and the corresponding definition window appears.
   Figure 1. The Map Views window with the view expanded

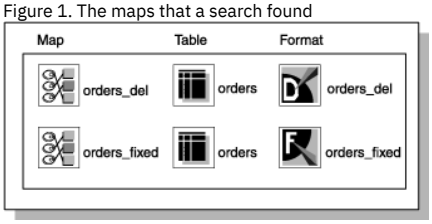


Seeing selected load maps

To see selected load maps:

1. Open the Map Views window.
2. Select a database.
3. Type the name or partial name of a map, table, or format in the Map, Table, or Format text box. You can use wildcards in the name.
4. Click Search.
   The following figure shows the maps that you find when you search for any table that includes **orders** in its name.

Figure 1. The maps that a search found



Define filters

This section describes how to create, edit, and delete filters.

- Filters
  Filters are similar to queries. However, queries select data from database tables, whereas filters select data from a data file. During the load process, **ipload** loads all of the records from a data file into a database table unless you use a filter to exclude some of the records.
- Example of using filter
- Creating a filter
- Modifying a filter
  After you create a filter, you might need to modify that filter.
- Filter views
  You can display a list of the filters and formats that are associated with a project from the Filter Views window. You can also create or edit a filter.
- Filters with code-set conversion (GLS)

Filters

Filters are similar to queries. However, queries select data from database tables, whereas filters select data from a data file. During the load process, **ipload** loads all of the records from a data file into a database table unless you use a filter to exclude some of the records.

A filter is a mechanism for prescreening data-file records for eligibility as database table entries. You can use the filter to include or exclude records explicitly during the load process. You define *match conditions* to filter the records. Match conditions are selection criteria that test one or more data-file fields for certain values or text.

You can define filters at any time. After you define a filter, you can specify it in the Load Job window. The Load Job window is illustrated in Figure 2.

Filters have the following restrictions:

- You cannot use filters with Ext Type data types.
- The DATE and DATETIME data filters can only be applied to the fixed ASCII and delimited format types.

Example of using filter

Suppose that you have a worldwide telemarketing data file that contains the name, country, yearly salary, and age of potential contacts, as the following example shows:

```
John BrownUS  125,00057
Mary SmithArgentina83,00043
Larry Little US  118,00042
Ann SouthCanada  220,00053
David PetersonFrance  175,00072
Richard NorthSpain350,00039
Nancy RichardsJapan150,00054
William ParkerEgypt200,00064
```

To create a database that includes people who earn over $100,000 a year, are over the age of 50, and live outside the United States:

1. Use the match condition **discard salary < 100,000** to exclude people who earn less than $100,000 a year. The selected records are as follows:

   ```
   John BrownUS125,00057
   Larry Little US118,00042
   Ann SouthCanada220,00053
   David PetersonFrance175,00072
   Richard NorthSpain350,00039
   Nancy RichardsJapan150,00054
   William ParkerEgypt200,00064
   ```

2. Use the match condition **keep age > 50** to include people over the age of 50. The remaining records are as follows:

   ```
   John BrownUS125,00057
   Ann SouthCanada220,00053
   David PetersonFrance175,00072
   Nancy RichardsJapan150,00054
   William ParkerEgypt200,00064
   ```

3. Use the match condition **discard country = US** to exclude people living in the United States. The remaining records are the records that match all of the restrictions:

   ```
   Ann SouthCanada220,00053
   David PetersonFrance175,00072
   Nancy RichardsJapan150,00054
   William ParkerEgypt200,00064
   ```

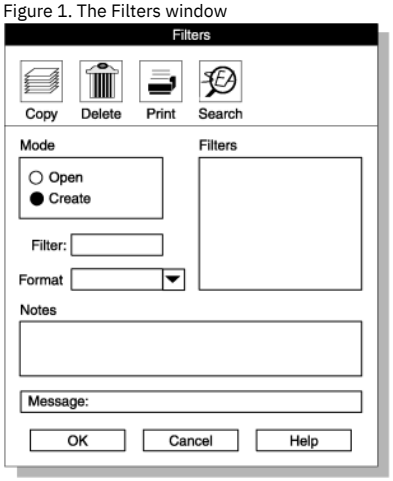
If you want to use the same data file to create a database of only those people who live in the United States, or only those people under the age of 30, simply define another filter. There is no limit to the number of filters that you can define for a data file.

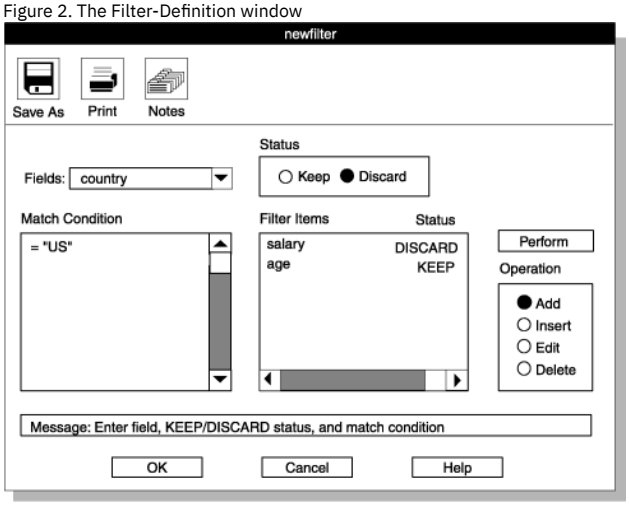
Related reference:
Define filters

Creating a filter

Before you can create a filter, you must create a format that describes the data file. For information about how to create a format, see Define formats.
The **ipload** utility stores the filter information in the **filters** table of the onpload database. For more information about the **filters** table, see The filters table in the onpload database.

To create a filter:

1. Choose Components > Filter from the HPL main window.
   The Filters window appears, as the following figure shows.
   Figure 1. The Filters window
   

2. Click Create in the Mode group.
3. Choose a name for the filter and type the name in the Filter text box.
4. Type the name of an existing format in the Format text box, or click the down arrow and choose a format from the selection list.
5. Click OK.
   The Filter-Definition window appears. The following figure shows a partially completed Filter-Definition window.
   Figure 2. The Filter-Definition window
   

   From the Filter-Definition window, you can prepare a filter that specifies which data from the input file is be loaded into the database table.

   The Filter-Definition window has the following parts.

Table 1. Parts of the Filter-Definition window

Section	Description
Fields	Specifies the data-file field used in a match condition
Status	Indicates whether you want to keep or discard records that meet the match condition
Match Condition	Specifies the criteria for keeping or discarding a record
Filter Items/Status	Lists existing filter items and their status As you add match conditions, the conditions are added to this list.

- Preparing the filter definition

Related reference:

The filteritem table in the onpload database
The filters table in the onpload database

Preparing the filter definition

To prepare the filter definition:

1. Click Add in the Operation group to specify that you want to add a new match condition.
2. Type the name of the record field that you want to match in the Fields text box. You can also click the down arrow to see a selection list.
3. Click Keep or Discard in the Status group. This selection indicates whether the matching record should be entered into the database or discarded.
4. Type the match condition in the Match Condition text box by using the appropriate logical operators and match characters.
   See Match condition operators and characters for a list of the logical operators and match characters.
5. Click Perform.
6. Repeat steps 2 through 5 for each additional filter item.
7. Click OK to save the filter and return to the Filters window.
8. Click Cancel to return to the HPL main window.

Modifying a filter

After you create a filter, you might need to modify that filter.

1. Choose Components > Filter from the HPL main window to display the Filters window.
2. Click Open in the Mode group.
3. Select the filter that you want to modify.
4. Click OK to display the Filter-Definition window.
5. Click Add in the Operation group.
6. Type the name of the record field in the Fields text box.
7. Type the match condition in the Match Condition text box.
8. Click Keep or Discard in the Status group to indicate the filter status.
9. Click Perform.
10. Click OK to save your changes and return to the Filters window.
11. Click Cancel to return to the HPL main window.

- Editing an existing filter
- Adding an item to the filter
- Inserting an item into the filter sequence
- Deleting a filter

Editing an existing filter

To edit an existing filter:

1. Choose Components > Filter from the HPL main window tp display the Filters window.
2. Click Open in the Mode group.
3. Select the filter that you want to modify.
4. Click OK to display the Filter-Definition window.
5. Click Edit in the Operation group.
6. Select the filter item you want from the list of items. The field, status, and match conditions appear in their respective areas on the screen.
7. Change the information that you want.
8. Click Perform.
9. Click OK to save your changes and return to the Filters window.
10. Click Cancel to return to the HPL main window.

Adding an item to the filter

To add an item to the filter:

1. Choose Components > Filter from the HPL main window to display the Filters window.
2. Click Open in the Mode group.
3. Select the filter that you want to modify.
4. Click OK to display the Filter-Definition window.
5. Click Add in the Operation group.
6. Type the name of the record field in the Fields text box.
7. Type the match condition in the Match Condition text box.
8. Click Keep or Discard in the Status group to indicate the filter status.
9. Click Perform.
10. Click OK to save your changes and return to the Filters window.
11. Click Cancel to return to the HPL main window.

Inserting an item into the filter sequence

To insert an item in the filter sequence:

1. Choose Components > Filter from the HPL main window to display the Filters window.
2. Click Open in the Mode group.
3. Select the filter that you want to modify.
4. Click OK to display the Filter-Definition window.
5. Click Insert in the Operation group.
6. From the list of items, select the filter item before which you want to insert the new item.
7. Type the name of the record field in the Fields text box.
8. Type the match condition in the Match Condition text box.
9. Click Keep or Discard in the Status group to indicate the filter status.
10. Click Perform to insert the new item before the selected filter item in the Filter Items list box.
11. Click OK to save your changes and return to the Filters window.
12. Click Cancel to return to the HPL main window.

Deleting a filter

To delete a filter:

1. Choose Components > Filter from the HPL main window to display the Filters window.
2. Click Open in the Mode group.
3. Select the filter that you want to edit.
4. Click OK to display the Filter-Definition window.
5. Click Delete in the Operation group.
6. Select the item that you want to delete from the list of filter items.
7. Click Perform.
8. Click OK to save your changes and return to the Filters window.
9. Click Cancel to return to the HPL main window.

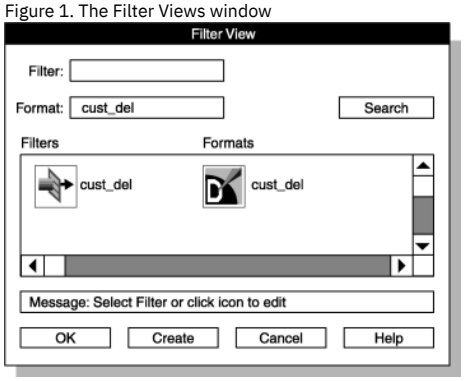
Filter views

You can display a list of the filters and formats that are associated with a project from the Filter Views window. You can also create or edit a filter.

The Filter Views window appears in the following situations:

- If you click Filter in the Load Job window when no filter name is in the Filter text box
- If you click Search in the Filters window

The following figure shows the Filter Views window. Views windows discusses the use of Views windows.
Figure 1. The Filter Views window



Filters with code-set conversion (GLS)

When you use a filter to select or discard data during the load, the High-Performance Loader (HPL) interprets the filter specification in the code set of the database server. The filtering process on data that undergoes code-set conversion occurs in the following order:

1. The **onpload** utility converts the input data to the code set of the database server.
2. The **onpload** utility performs the filtering operation.

If the code-set conversion process creates lossy errors, then the output of the filter operation can be unexpected. For information about lossy errors and how to define or evaluate a code-set conversion specification, see the *IBM® Informix® GLS User's Guide*.

Unload data from a database

This section describes the Unload Job window.

- Unload jobs
  An unload job converts IBM Informix database records to a specified format and then unloads those records to a file, tape, pipe (UNIX only), or device array.
- Components of the unload job
- The Unload Job windows
  You can create an unload job or select an existing job for editing from the Unload Job Select window.
- Generate options for an unload job
  Instead of individually creating the components that are required on the Unload Job window, you can use the Generate options to create an unload job.

Related concepts:
The HPL data-unload process
Load Job and Unload Job windows

Unload jobs

An unload job converts IBM® Informix® database records to a specified format and then unloads those records to a file, tape, pipe (UNIX only), or device array.

You can run an unload job from the Unload Job window of **ipload**, or you can run the **onpload** utility from the command line.

Related reference:
The onpload utility

Components of the unload job

Before you can unload data, you must first define the following components of the unload job:

- The device array that receives the unloaded data.
- The format that describes the organization of the data file into which you are unloading data.
- The query that extracts the records that you want from the database.
- The unload map that describes the relationship between the columns of a database table and the fields of the data-file record.
  The map also specifies any necessary data translations, such as case conversion and justification.

You can define these components in the following ways:

- Define each component from the Unload Job window.
- Define each component individually from the Components menu.
- Use the Generate Job option from the Components menu.
- Use the Generate button in the Unload Job window.

- Choose the database server
  You must run the unload job on the *target server*. The target server is the database server that contains the database from which you unload the data. The database must be on the same database server as onpload that extracts data from it. You can run **ipload** on any database server on your network.
- Run multiple jobs
  You can run multiple unload jobs concurrently.

Related concepts:
Define device arrays
Related reference:
Define formats
Define queries
Define maps
The Generate options of the ipload utility

Choose the database server

You must run the unload job on the *target server*. The target server is the database server that contains the database from which you unload the data. The database must be on the same database server as onpload that extracts data from it. You can run **ipload** on any database server on your network.

Run multiple jobs

You can run multiple unload jobs concurrently.

However, because the High-Performance Loader (HPL) is designed to use as many system resources as possible, running concurrent jobs might overload the system. If you are using a UNIX **cron** job to run the load and unload jobs, let one job finish before you start the next.

The Unload Job window displays the target and onpload database servers in the upper right corner of the display.

The Unload Job windows

You can create an unload job or select an existing job for editing from the Unload Job Select window.

From the Unload Job window, you can also create or modify the components of an unload job and run the unload job. You can change unload options before you run the unload job. The unload options include the isolation level and the maximum number of errors to permit before **onpload** stops the unload job.

The **ipload** utility stores the information about the unload job in the **session** table of the onpload database. The **session** table draws information from other onpload tables, such as **maps**, **formats**, and so on.

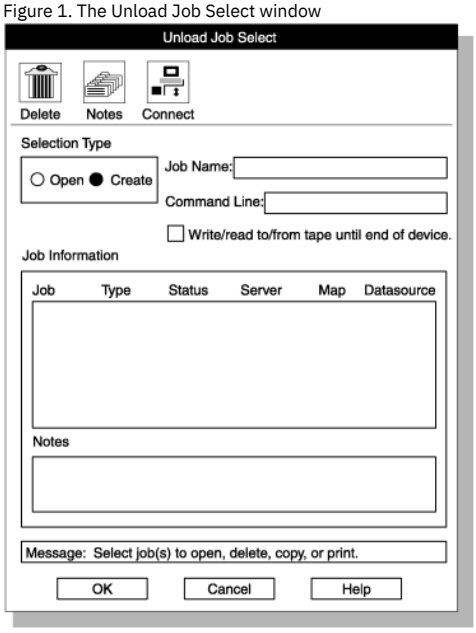
- Creating an unload job
  Use the Unload Job Select and Unload Job windows to create an unload job.
- Run the unload job
- Specify to write to the end of the tape
  On the Unload Job Select window, you can specify to write to the tape until the end of the device with the Write/read to/from tape until end of device check box. When a tape device is full, you are prompted to provide the next tape, until the unload job is complete.
- The command-line information
  If you select an existing job in the Unload Job Select window, the Command Line text box shows the **onpload** command that **ipload** generated for that unload job.
- Changing the unload options
- Editing an unload job
  After you save an unload job, you can return to the unload job and modify it.

Related reference:
The onpload database

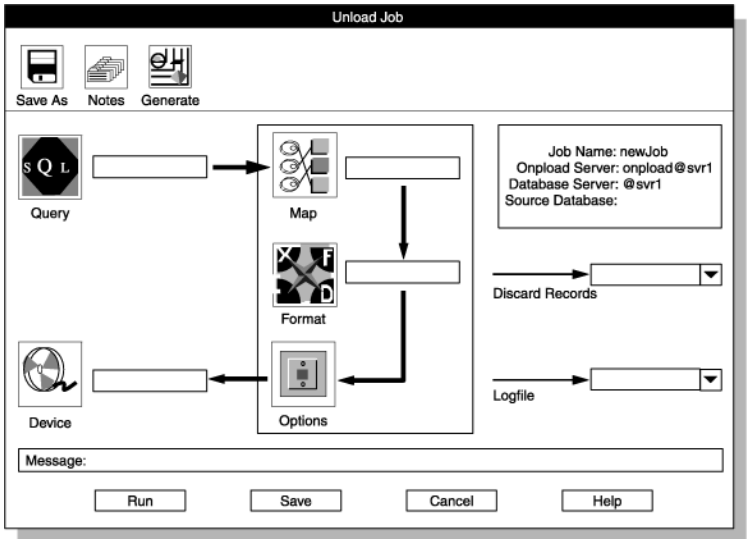
Creating an unload job

Use the Unload Job Select and Unload Job windows to create an unload job.

To create an unload job:

1. Choose Jobs > Unload from the HPL main window.
   The Unload Job Select window appears, as the following figure shows.
   Figure 1. The Unload Job Select window
   

2. Click Create in the Selection Type group.
3. Choose a name for this unload job and type the name in the Job Name text box.
4. Optionally check the Write/read to/from tape until end of device check box. For more information, see Specify to write to the end of the tape.
5. Click OK.
   The Unload Job window appears, as the following figure shows. The ipload utility windows, provides detailed descriptions of the buttons in the Unload Job window.
   Figure 2. The Unload Job window



6. Type appropriate values for all of the unload components. If you click a component button, the corresponding view window opens, and you can create or select the component.
7. Specify the file that contains rejected record by using one of these methods:
   - Type the name of the rejected file in the Discards Records text box.
   - Click the down arrow next to the Discard Records text box to select the file name from the file-selection list.
8. Select the file that contains the unload status log by using one of these methods:
   - Type the name of the log file in the Logfile text box.
   - Click the down arrow next to the Logfile text box to select a file name from the File-Selection window.
9. Click Options to change unload options. For more information, see Changing the unload options.
10. Click Save to save this unload job. (If you click Run to run the job immediately, the job is saved automatically.)
11. Now you can either run the unload job or exit and run the job later.
    - Click Run to run the job.
    - Click Cancel to exit to the Unload Job Select window.

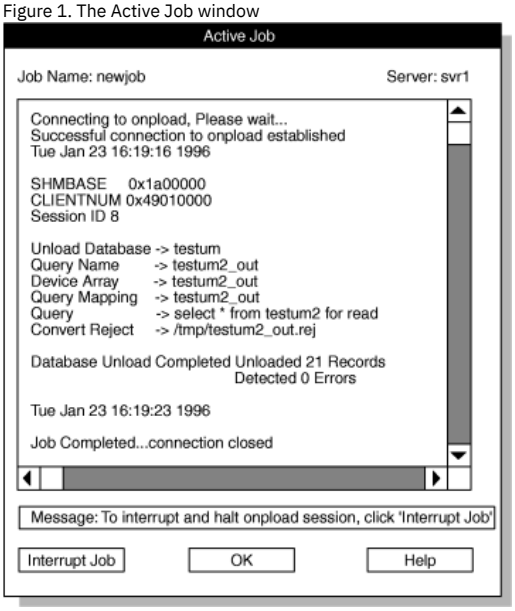
Important: Use Ext Type String Length data type or Ext Type Binary Length data type if you unload data that contains null UDT values.

Run the unload job

If you click Run in the Unload Job window, the Active Job window appears, as the following figure shows. The Active Job window displays the progress of your job and indicates when the job completes. When the Active Job window indicates that the job is complete, click OK to return to the Unload Job Select window.

The information that **onpload** displays in the Active Job window is also stored in the log file whose name you selected in step 8.

Figure 1. The Active Job window



- Problems during an unload job
  If you encounter any problems during the unload, examine the various files that **onpload** creates.

Related reference:
View the status of a load job or unload job

Problems during an unload job

If you encounter any problems during the unload, examine the various files that **onpload** creates.

Important: If a write to a file fails because a disk is out of space, the operating system does not return information about how much of the write succeeded. In this situation, the **onpload** utility cannot accurately report the number of records that were written to disk. Thus, the number of records that are logged as unloaded in the log file is imprecise.
Related reference:
The HPL browsing options

Specify to write to the end of the tape

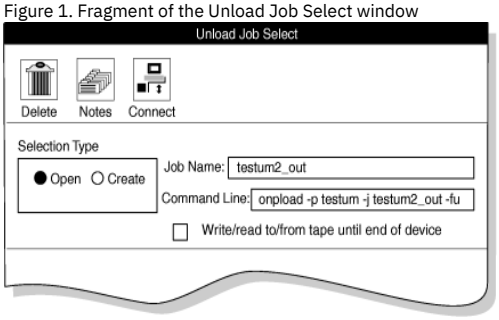
On the Unload Job Select window, you can specify to write to the tape until the end of the device with the Write/read to/from tape until end of device check box. When a tape device is full, you are prompted to provide the next tape, until the unload job is complete.

Important: If you select this option, you must also select it when loading from the Load Job window or specify the **-Z** option from the command line; otherwise, the loaded and unloaded data might be inconsistent.
If you check this check box, it supersedes any tape size information you enter in the Device-Array Definition window or at the command line.
Important: You must provide the same tapes in the same order on the same devices for both unload jobs and load jobs to ensure consistency.

The command-line information

If you select an existing job in the Unload Job Select window, the Command Line text box shows the **onpload** command that **ipload** generated for that unload job.

The following figure shows the Command Line portion of an Unload Job Select window. The Command Line text box displays the **onpload** command generated for the job shown in Figure 1.
Figure 1. Fragment of the Unload Job Select window



The command line, **onpload -p testum -j testum2_out -fu**, contains the following arguments.

-p testum
: The project where the job is stored.

-j testum2_out
: The name of the job.

-fu
: The job that unloads (rather than loads) data.

You can copy the **onpload** command from the Command Line text box and paste it at a system prompt to run the unload job. If you need to run the unload job multiple times (for example, every evening at 5:00 p.m.), you can save the **onpload** command and run it later.

You do not need to start **ipload** to run a job from the system prompt. Both **ipload** and **onpload** use the **onpload** database, but each one uses it independently.

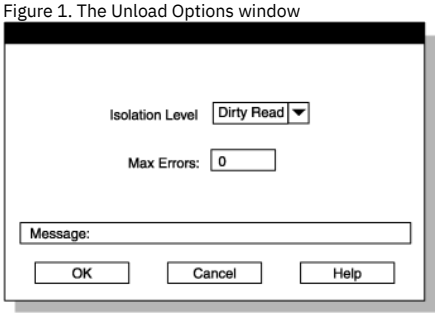
Changing the unload options

The Unload Options window contains the following options.

Table 1. The Unload Options window options

Option	Description
Isolation Level	The criteria for how the query selects records. The four levels of isolation (from highest to lowest) are as follows: • Committed • Cursor Stability • Repeatable Read • Dirty Read The higher the isolation level, the lower the unload performance. For a more detailed definition of isolation levels, see the *IBM® Informix® Guide to SQL: Syntax*.
Max Errors	The maximum number of error conditions to be encountered. If the number of unload errors exceeds this number, the unload job stops.

To change unload options:

1. Display the Unload Job window. See the instructions for Creating an unload job.
2. Click Options.
   The Unload Options window appears, as the following figure shows.
   Figure 1. The Unload Options window
   

3. Change the options that you want.
4. Click OK to return to the Unload Job window.

Editing an unload job

After you save an unload job, you can return to the unload job and modify it.

To edit an unload job:

1. Choose Jobs > Unload from the HPL main window.
2. Click Open in the Selection Type group.
3. Select a job from the Job Information list box.
4. Click OK to display the Unload Job window.
5. Make the appropriate changes to the entries in the Unload Job window.
6. Click Options to change unload options. For more information, see Changing the unload options.
7. Click Save to save this unload job.
8. Now you can either run the unload job or exit and run the job later.
   - Click Run to run the job.
   - Click Cancel to exit.

Generate options for an unload job

Instead of individually creating the components that are required on the Unload Job window, you can use the Generate options to create an unload job.

You can click Generate in the Unload Job window, or you can choose Components > Generate from the HPL main window.

The Generate options do not give you as much flexibility as the Unload Job window, but the options let you create the components quickly. In addition, the Generate options let you create formats (Binary, Fixed Internal, and No Conversion) that are not available from the Format-Definition window.

Related reference:
The Generate options of the ipload utility

Load data to a database table

This section describes the load process.

- Load jobs
  A load job loads data from a set of one or more files into a single database table. A record format, which defines each field of a record, specifies the layout of the input data. A load map specifies how the record fields are mapped to the columns of the target table. During the load process, the **onpload** utility converts data from record field to table column.
- Components of the load job
  The High-Performance Loader (HPL) lets you define the individual components of a data load individually or lets you use the generate option to define the components automatically.
- The Load Job windows
  You can create, save, and run a load job from the Load Job Select window and the Load Job window. The Load Job window visually represents the various components of a load. After you select the components, you can save the load job for future use or run it immediately.
- Generate options for a load job
  Instead of individually creating the components that are required on the Load Job window, you can use the Generate options to create a load job.

Related concepts:
Load Job and Unload Job windows

Load jobs

A load job loads data from a set of one or more files into a single database table. A record format, which defines each field of a record, specifies the layout of the input data. A load map specifies how the record fields are mapped to the columns of the target table. During the load process, the **onpload** utility converts data from record field to table column.

Components of the load job

The High-Performance Loader (HPL) lets you define the individual components of a data load individually or lets you use the generate option to define the components automatically.

The components of the load job specify:

- The device array where the source data files resides
- The format of the data files
- The filter that accepts or rejects source-file records for the load
- The map that specifies the relationship between the data-file format and the database table schema

When you run a load job, you select which individual components to use. The collection of the various components for a specific load is called the load job. You can assign a name to a load job, save the job, and then retrieve and rerun it as often as you need to. You can modify an existing job or save it under another job name.

You can define as many different load jobs as you need. You can group your load jobs under one or more projects to make the tasks easier to manage.

- Choose the database server
  You must run the load job on the *target server*. The target server is the database server that contains the database into which you load the data. The target database must be on the same database server as the **onpload** program that updates it.
- Run multiple jobs
  You can run only one express-load job at a time on the same table, although you can run multiple unload jobs concurrently. Because the High-Performance Loader (HPL) is designed to maximize the use of system resources, running concurrent jobs might overload the system.
- Prepare user privileges and the violations table
  You must make sure that the user who runs a load job has sufficient privileges to manage the constraints and the violations table.

Related reference:
Define onpladm utility jobs

Choose the database server

You must run the load job on the *target server*. The target server is the database server that contains the database into which you load the data. The target database must be on the same database server as the **onpload** program that updates it.

Tip: The onpload database and the **ipload** interface can be on different computers. You can run the **ipload** interface on any computer that can connect to the database server that contains the onpload database.

Run multiple jobs

You can run only one express-load job at a time on the same table, although you can run multiple unload jobs concurrently. Because the High-Performance Loader (HPL) is designed to maximize the use of system resources, running concurrent jobs might overload the system.

If you are using a UNIX **cron** job to run the load or unload jobs, let one job finish before you start the next.

Prepare user privileges and the violations table

You must make sure that the user who runs a load job has sufficient privileges to manage the constraints and the violations table.

The following table summarizes the actions that you must take. The following sections discuss these actions in more detail.

Table Status	User Privileges	Action
Owned by user		No further action is required.
Not owned by user	User has DBA privileges on the table.	No further action is required.
Not owned by user	User does not have DBA privileges on the table.	User must have: • Resource privileges on database. • Alter privileges on table. Owner must start violations table.

For detailed information about user privileges and violations tables, see the *IBM® Informix® Guide to SQL: Syntax* and the *IBM Informix Guide to SQL: Reference*.

- Set user constraints
  To modify any constraint, index, or trigger, a user must have both Alter privileges on the table and the Resource privilege on the database. The user must also have these privileges to start or stop a violations table. You use the GRANT statement to set these privileges.
- Manage the violations and diagnostics tables

Set user constraints

To modify any constraint, index, or trigger, a user must have both Alter privileges on the table and the Resource privilege on the database. The user must also have these privileges to start or stop a violations table. You use the GRANT statement to set these privileges.

Manage the violations and diagnostics tables

You can turn on or turn off the generation of constraint-violation information. If you turn on the generation of constraint-violation information, **onpload** writes the information to the violations and diagnostics tables.

The High-Performance Loader (HPL) manages the violations and diagnostics tables in the following manner:

1. Starts the load job.
2. Starts the violations and diagnostics tables if they do not exist. (If a violations and diagnostics table exists, the HPL uses that table).
   The HPL uses the following SQL statement to start the violations table:

   ```
   START VIOLATIONS TABLE FOR tablename
   ```

3. Performs the load job.
4. Stops the violations and diagnostics tables if they were started at step 2.
   The HPL uses the following SQL statement to stop the violations and diagnostics tables:

   ```
   STOP VIOLATIONS TABLE FOR tablename
   ```

5. Drops the violations table if the violations table is empty.

The START VIOLATIONS TABLE statement creates the violations and diagnostics tables and associates them with the load table. The STOP VIOLATIONS TABLE statement dissociates the violations and diagnostics tables from the load table.

The violations table (***tablename*_vio**) and the diagnostics table (***tablename*_dia**) are always owned by the owner of the table with which they are associated. The Resource privilege lets a user start and stop a violations table, but it does not let the user drop a table that the user does not own. Thus, the HPL cannot drop the violations table in step 5 if the user is not the owner.

Failure to drop the violations table does not cause the load job to fail. However, this failure leaves in the database a violations table that is not associated with a table. If the user tries to run the job again, the START VIOLATIONS TABLE statement in step 2 fails because the table ***tablename*_vio** exists.

To solve this problem, the owner of the table or the database administrator must explicitly create the violations and diagnostics tables by using the START VIOLATIONS statement. When the owner creates the violations table, the following actions take place:

- In step 2, the HPL uses the existing violations table.
- In step 4, the HPL does not stop the violations table because the table was not started in step 2.
- In step 5, the HPL does not drop the violations table because the user does not own the table.

After the load job is complete, an active violations table remains in the database. This table might be empty, but does no harm. When the user runs the load job a second time, the violations table is available, and the load job succeeds.

Related tasks:
Changing the load options
Related information:
START VIOLATIONS TABLE statement
STOP VIOLATIONS TABLE statement

The Load Job windows

You can create, save, and run a load job from the Load Job Select window and the Load Job window. The Load Job window visually represents the various components of a load. After you select the components, you can save the load job for future use or run it immediately.

The **ipload** utility assigns path names for the log files that document the load and that capture records that do not pass the specified filter or that do not pass conversion.

When you use **ipload** to create a job, **ipload** stores information for the job in a row in the **session** table of the onpload database. The **ipload** utility stores information about the components of the load job in other tables of the onpload database, including **format**, **maps**, **filters**, and so on. When you use the **onpload** command, columns in the **session** table reference the components to assemble the information necessary for the job.

- Creating a load job
  Use the Load Job Select window to create a load job or select an existing job to edit.
- Run the load job
- Specify to read to the end of the tape
  From the Load Job Select window, you can specify to read from the tape until the end of the device with the Write/read to/from tape until end of device check box. When a tape device is empty, you are prompted to provide a new tape, until the load job is complete.
- The command-line information
  If you select an existing job in the Load Job Select window, the Command Line text box shows the **onpload** command that **ipload** generated for that load job.
- Changing the load options
- Editing a load job
  After you create and save a load job, you can later return and modify that load job.

Related reference:
The onpload database

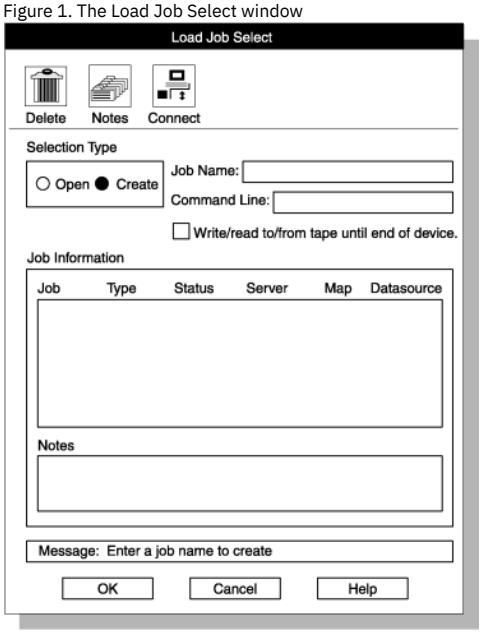
The session table in the onpload database

Creating a load job

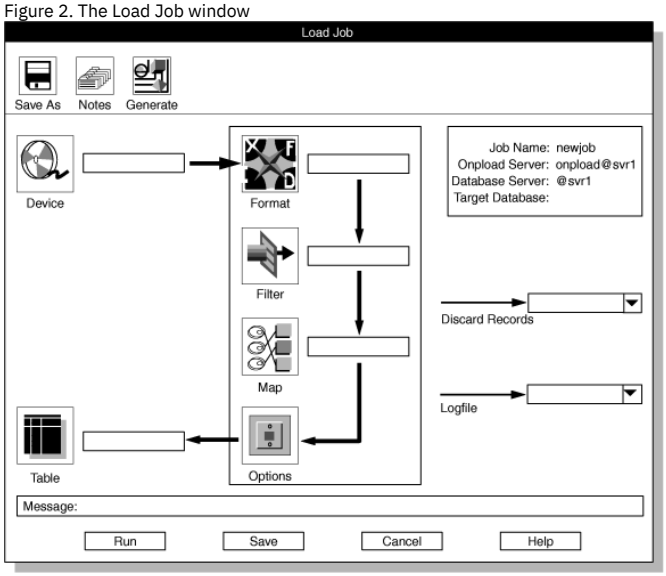
Use the Load Job Select window to create a load job or select an existing job to edit.

To create a load job:

1. Choose Jobs > Load from the HPL main window.
   The Load Job Select window appears, as the following figure shows.
   Figure 1. The Load Job Select window



2. Click Create in the Selection Type group.
3. Select a name for the job and type it in the Job Name text box.
4. Optional: Check the Write/read to/from tape until end of device check box. For more information, see Specify to read to the end of the tape.
5. Click OK.
   The Load Job window appears, as the following figure shows.
   Figure 2. The Load Job window



6. Type the appropriate values for the components of the load.
   The HPL ipload utility icon buttons describes the icons that represent the components of the load. For detailed information about these components, see the individual topics on device arrays, formats, filters, and maps.
7. Select a base name for the files that contain rejected records and type it in the Discard Records text box. Reviewing records that the conversion rejected gives information about rejected records.
8. Choose a name for the file that contains the load job status log and type it in the Logfile text box. For more information about the log file, see View the status of a load job or unload job.
9. Click Options to change the load options. For more information about these options, see Changing the load options.
10. Click Save to save this load job. (If you click Run to run the job immediately, the job is saved automatically.)
11. Now you can either run the load job or exit and run the job later.
    - Click Run to run the job.

- Click Cancel to exit to the Load Job Select window.

Run the load job

If you click Run in the Load Job window, the Active Job window appears, as Figure 1 shows. The Active Job window displays the progress of your job and indicates when the job completes. When the Active Job window indicates that the load job is complete, click OK to return to the Load Job Select window.
Tip: Before you run a load job, you might want to view the data-file records according to a specified format to check your definitions.
After you run an express-mode load, you must make a level-0 backup before you can access the table that you loaded.

- Make a level-0 backup
  Express-mode loads do not log loaded data. After an express-mode load, **onpload** sets the table to read-only as a protective measure. To make the table available for write access, you must do a level-0 backup for all the dbspaces that the fragments of the loaded table occupy. The level-0 backup allows data recovery for the table in case of future database corruption.
- Problems during a load job
  If you encounter any problems during the load, examine the various files that **onpload** creates.

Related concepts:
Preview data-file records

Make a level-0 backup

Express-mode loads do not log loaded data. After an express-mode load, **onpload** sets the table to read-only as a protective measure. To make the table available for write access, you must do a level-0 backup for all the dbspaces that the fragments of the loaded table occupy. The level-0 backup allows data recovery for the table in case of future database corruption.

If you do not need to provide for data recovery, you can use /dev/null as the backup device for the level-0 backup. This strategy makes the table available for write access without actually backing up the data. If a user attempts to write into the table before you make a level-0 backup, the database server issues an error message.

If you run several express-load jobs on different tables in a database, you can complete all of the loads before you perform the level-0 backup. However, if you try to do a second load on the same table without making a level-0 backup, the database server issues an error message.

For discussions of table fragments and dbspaces, see the *IBM® Informix® Administrator's Guide*. For information about making backups, see the *IBM Informix Backup and Restore Guide*.

Related concepts:
How the express and deluxe modes work

Problems during a load job

If you encounter any problems during the load, examine the various files that **onpload** creates.

Important: Because of operating-system limitations, the **onpload** utility cannot load successfully from a file (on disk) that is larger than 2 GB on a platform that is less than a 64-bit platform. If you try to read a file that is larger than 2 GB, **onpload** HPL fails only after it processes the first 2 GB of data. The HPL log file reports the following error:

```
Cannot read file /some_dir/a_long_file - aio error code 27
```

Related reference:
The HPL browsing options

Specify to read to the end of the tape

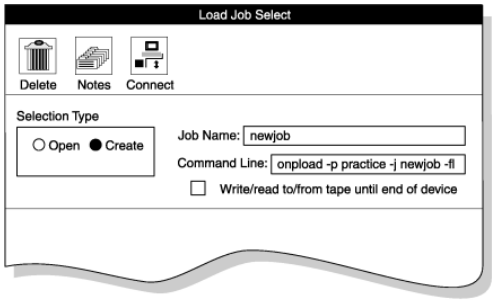
From the Load Job Select window, you can specify to read from the tape until the end of the device with the Write/read to/from tape until end of device check box. When a tape device is empty, you are prompted to provide a new tape, until the load job is complete.

Important: If you select this option, you must also select it when unloading from the Unload Job window, or specify the **-Z** option from the command line; otherwise, the loaded and unloaded data might be inconsistent.
If you check this check box, it supersedes any tape size information you enter in the Device-Array Definition window or at the command line.
Important: You must provide the same tapes in the same order on the same devices for both the unload jobs and the load jobs to ensure consistency.

The command-line information

If you select an existing job in the Load Job Select window, the Command Line text box shows the **onpload** command that **ipload** generated for that load job.

The following figure shows the Command Line portion of a Load Job Select window. The Command Line text box displays the **onpload** command generated for the load job that Figure 3 shows.
Figure 1. Fragment of the Load Job Select window



The command line, **onpload -p practice -j newjob -fl**, contains the following arguments.

-p practice
: The project where the job is stored.

-j newjob
: The name of the job.

-fl
: The job that loads (rather than unloads) data.

You can copy the **onpload** command from the Command Line text box and paste it at a system prompt to run the load job. If you need to run the load job multiple times, you can save the **onpload** command and run it later.

You do not need to start **ipload** to run a job from the system prompt. The **ipload** and **onpload** utilities both use the onpload database, but each utility uses it independently.

Changing the load options

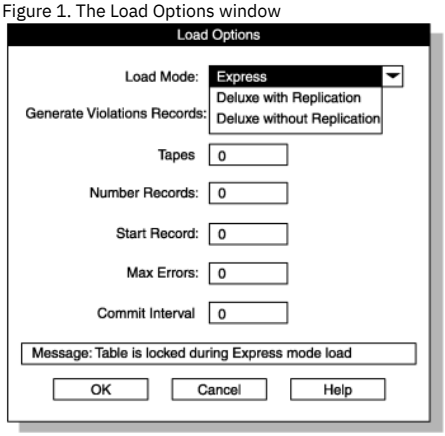
Before you run a load job, you can review or change any load options. The load options include specifying the number of records to load, the starting record number, and the loading mode.
The **ipload** utility stores option information in the **session** table of the onpload database.

The Load Options window contains the following option text boxes.

Table 1. The Load Options window options

Option	Description
Load Mode	The mode for the load: express, deluxe, or deluxe without replication
Generate Violations Records	Whether or not to generate violations records
Tapes	The number of tapes that contain source data
Number Records	The number of records to process in the data file
Start Record	The record number in the data file from which to start loading
Max Errors	The maximum number of error conditions to be encountered If the number of load errors exceeds this number, the load stops.
Commit Interval	The number of records to load before logging the transaction If you set the commit interval to `0`, **onpload** uses the default value of `10`. You can use this option only with deluxe mode.

To change load options:

1. Display the Load Job window. See Creating a load job.
2. Click Options.
   The Load Options window appears, as the following figure shows.
   Figure 1. The Load Options window
   

3. Change the options that you want.
4. Click OK to return to the Load Job window.

Related concepts:

The HPL ipload utility icon buttons
Manage the violations and diagnostics tables
Related reference:
The onpload database

Editing a load job

After you create and save a load job, you can later return and modify that load job.

To edit a load job:

1. Choose Jobs > Load from the HPL main window to display the Load Job Select window.
2. Click Open in the Selection Type group.
3. Select a job from the Job Information list box.
4. Click OK to display the Load Job window.
5. Make appropriate changes to the entries in the Load Job window.
6. Click Options to change load options.
7. Click Save to save this load job.
8. Run or cancel the load, as follows:
    - Click Run to run the data load.
    - Click Cancel to exit.

Generate options for a load job

Instead of individually creating the components that are required on the Load Job window, you can use the Generate options to create a load job.

You can click Generate in the Load Job window, or you can choose Components > Generate Job from the HPL main window.

The Generate options do not give you as much flexibility as creating each component individually, but these options let you create the components quickly. (After you generate the components, you can edit the components individually by accessing them through the Components menu.) In addition, the Generate options let you create formats (Fixed Internal and No Conversion) that are not available from the Format-Definition window.

Related reference:
The Generate options of the ipload utility

The Generate options of the ipload utility

This section describes the Generate options for the **ipload** utility.

- Overview of the ipload Generate options
  Use the generate options of **ipload** to automatically generate components of a load or unload job. The generate options can save you time when you create formats, maps, queries, and load and unload jobs.
- Tasks that generate load or unload components
- Generate from the Load Job window
  Use the Generate button in the Load Job window to save time when the format of the data file corresponds to the format of the database table.
- Generate from the Unload Job window
  Use the Generate button in the Unload Job window to save time when the format of the data file is similar to the format of the database table.
- Generate from the Components menu
  To generate all of the components for both load and unload jobs in one operation, choose Components > Generate Job from the main HPL window. This Generate option lets you choose formats that are not available from the Format-Definition window.

Related concepts:
Formats
Components of the unload job
Related reference:
The ipload utility Generate options
Generate options for an unload job
Generate options for a load job

Overview of the ipload Generate options

Use the generate options of **ipload** to automatically generate components of a load or unload job. The generate options can save you time when you create formats, maps, queries, and load and unload jobs.

When you generate a load or unload job for a database, **ipload** creates a format for the data file and a map that associates the columns of the table with the fields of the data-file records. Although the generated components might not match your database schema or data-file records exactly, the components created by the generate options provide useful starting points for building HPL components. After you generate default components, you can modify the components to match your specific needs.

Tasks that generate load or unload components

Use the **ipload** utility to perform the following tasks:

- Generate load components from the Load Job window.
- Generate unload components from the Unload Job window.
- Generate both load and unload components from the Components menu.

Generate from the Load Job window

Use the Generate button in the Load Job window to save time when the format of the data file corresponds to the format of the database table.

When you generate from the Load Job window, **ipload** makes the following assumptions about the file (or device array) that contains the data:

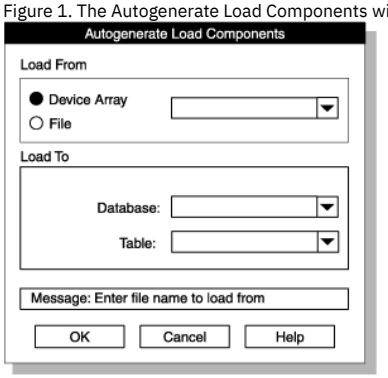
- The file is an ASCII file.
- The file uses the same locale as the database.
- The file uses a vertical bar (|) for the field delimiter and a new line for the record delimiter.
- The fields in each record of the file correspond one-to-one to the columns of the target table.
- All records in the file should be loaded.

- Generating a job from the Load Job window
  When you generate from the Load Job window, **ipload** creates a format, a map, a job, and, if needed, a device array.

Generating a job from the Load Job window

When you generate from the Load Job window, **ipload** creates a format, a map, a job, and, if needed, a device array.

To generate a job from the Load Job window:

1. Choose Jobs > Load from the HPL main window to display the Load Job Select window.
2. Click Create in the Selection Type group.
3. Select a name for the load job and type it in the Job Name text box.
4. Click OK to display the Load Job window.
5. Click Generate.
   The Autogenerate Load Components window appears, as the following figure shows.
   Figure 1. The Autogenerate Load Components window
   

6. Click Device Array or File to indicate the location of the source data
   - To load from an existing device array, click Device Array and type the name of the device array.
   - To load from a file, click File and type the full path name of the file. The **ipload** utility automatically generates a device array that includes the file.
7. In the Load To group, type the name of the database and table that will receive the data.
8. Click OK to generate the components of the load and return to the Load Job window.
9. If needed, click Filter to prepare a filter.
10. If you want, change the path names in the Discard Records and Logfile text boxes.
11. Click Save to save the components and the job.
12. Click Run to run job or Cancel to exit.

Generate from the Unload Job window

Use the Generate button in the Unload Job window to save time when the format of the data file is similar to the format of the database table.

When you generate from the Unload Job window, **ipload** makes the following assumptions about the file (or device array) into which the data is unloaded:

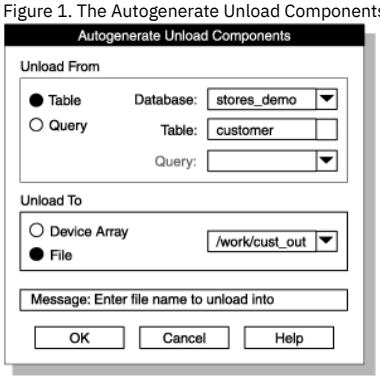
- The file is an ASCII file.
- The file uses the same locale as the database.
- The file uses a vertical bar (|) for the field delimiter and a new line for the record delimiter.

- Generating a job that uses a query
  When you generate from the Unload Job window, **ipload** creates a format, a map, a job, and, if needed, a device array. You can generate an unload job that uses a query to select from one or more tables or that unloads an entire table.
- Generating a job that unloads an entire table

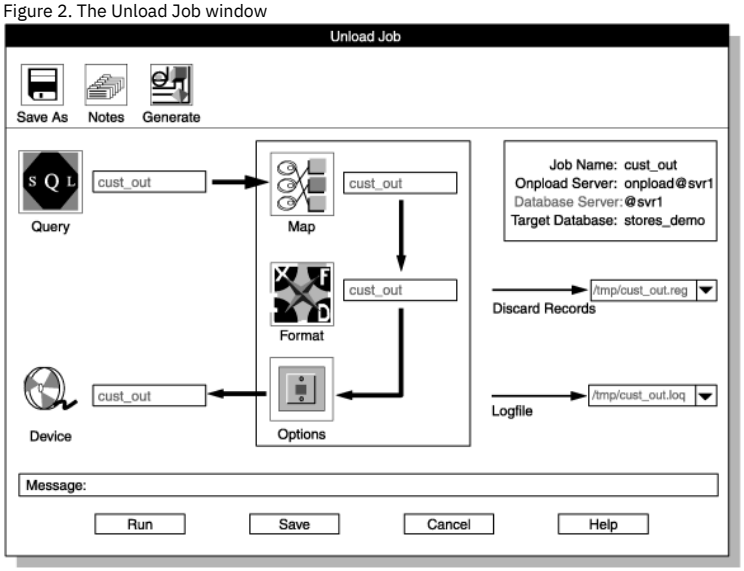
Generating a job that uses a query

When you generate from the Unload Job window, **ipload** creates a format, a map, a job, and, if needed, a device array. You can generate an unload job that uses a query to select from one or more tables or that unloads an entire table.

To generate a job that uses a query

1. Follow the instructions in Creating a query to create a query.
2. Choose Job > Unload from the HPL main window to display the Unload Job Select window.
3. Click Create in the Selection Type group.
4. Select a name for the unload job and type it in the Job Name text box.
5. Click OK to display the Unload Job window.
6. Click Generate.
   The Autogenerate Unload Components window appears, as the following figure shows.
   Figure 1. The Autogenerate Unload Components window
   

7. Click Query in the Unload From group.
8. Enter the name of the query. You can use the down arrow to see selection lists. When you unload from a table, you do not enter a query.
9. Click Device Array or File in the Unload To group.
   - If you click Device Array, you can use the down arrow to see a list of the available device arrays.
   - If you click File, **ipload** creates a device array of the same name as the unload job and inserts the specified file into that device array.

   Important: If you are executing **onpload** from the command line during an unload job, you must first create the file.
10. Click OK to generate the components of the unload job.
    The display returns to the Unload Job window. The **ipload** utility completes the Unload Job window. If you chose **cust_out** for the unload job name (step 4), the Unload Job window appears as the following figure shows.
    Figure 2. The Unload Job window
    

11. Click Save to save this Unload Job. You can click Run to run the job, or click Cancel to exit and run the job later.
12. To run the job, click Run to display the Active Job window.
13. When the Active Job window displays `Job Completed`, click Cancel to return to the main HPL window.

Generating a job that unloads an entire table

To generate a job that unloads an entire table:

1. Choose Jobs > Unload from the HPL main window to display the Unload Job Select window.
2. Click Create in the Selection Type group.
3. Choose a name for the unload job and type it in the Job Name text box.
4. Click OK to display the Unload Job window.
5. Click Generate.

The Autogenerate Unload Components window appears, as Figure 1 shows.

6. Click Table in the Unload From group.
7. Enter the database that you want in the Database text box and the table that you want in the Table text box. You can use the down arrows to see selection lists. When you unload from a table, you do not enter a query.
8. Click Device Array or File in the Unload To group.
   - If you click Device Array, you can use the down arrow to see a list of the available device arrays.
   - If you click File, **ipload** creates a device array of the same name as the unload job and inserts the specified file into that device array.
9. Click OK to generate the components of the unload job. The Unload Job window reappears, with the components of the job completed.
10. Click Save to save this Unload Job. You can click Run to run the job, or click Cancel to exit and run the job later.
11. To run the job, click Run and the Active Job window appears.
12. When the Active Job window displays `Job Completed`, click Cancel to return to the main HPL window.

Generate from the Components menu

To generate all of the components for both load and unload jobs in one operation, choose Components > Generate Job from the main HPL window. This Generate option lets you choose formats that are not available from the Format-Definition window.

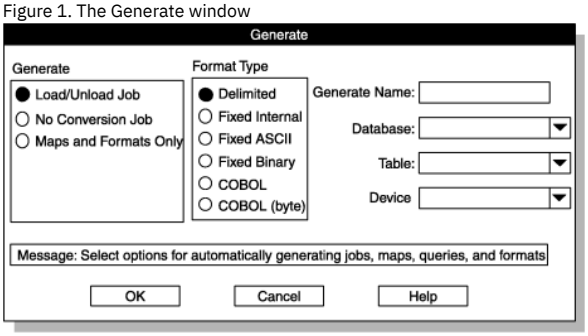
- The Generate window
  You can specify the characteristics of the components that **ipload** creates from the Generate window.
- Generating load and unload components
  When you choose Components > Generate, you can generate all of the components required for a load job and an unload job: format, load map, unload map, query, and device array.
- Using the No Conversion Job option
  The No Conversion Job option uses the IBM® Informix® internal format to unload data from a table. Jobs loaded or unloaded with this option are sometimes called raw loads or raw unloads.

The Generate window

You can specify the characteristics of the components that **ipload** creates from the Generate window.

The following figure shows the Generate window. This window generates all of the components required for a load job and an unload job: format, load map, unload map, query, and device array.

Figure 1. The Generate window



- Generate group
  The Generate group specifies the type of generate to perform.
- Format type group
  The Format Type group specifies the format of the data file.

Generate group

The Generate group specifies the type of generate to perform.

The Generate group has the following choices.

Choice	Effect	See
Load/Unload Job	Generates both load and unload jobs	Generating load and unload components
No Conversion Job	Generates a job that treats an entire database record as one entity	Using the No Conversion Job option
Maps and Formats only	Generates only a format, a load map, and an unload map	

Format type group

The Format Type group specifies the format of the data file.

The Format Type group has the following choices.

Choice	Description	See
Delimited	The fields of a data-file record are separated by a field delimiter, and records are separated by a record delimiter. The default delimiter for fields is a vertical bar (|). The default delimiter for records is a new line.	Modifying delimited-format options
Fixed Internal	The data file uses IBM® Informix® internal format. The only changes to the data that you can make when you use this format are ALTER TABLE changes: modify the order of columns, delete or add columns, or change the data type. The HPL loads and unloads data in this format more efficiently than data in the Delimited and Fixed ASCII formats.	Other formats
Fixed ASCII	All records are the same length. Each record contains characters in fixed-length fields. This format is the same as the Fixed format choice of the Record Formats window.	Fixed-length records
Fixed Binary	The data file records contain data in fixed-length fields. Character-oriented data is in character fields. Numeric data (integer, float, and so on) is in machine-dependent binary values. Use this format for loading or unloading data for an application that has or requires data in binary format. Data in binary format is much more compact than data in ASCII format.	Fixed-length records
COBOL	The data file is formatted according to COBOL 86 standards. All COBOL data types are supported.	COBOL records
COBOL (byte)	The data file is formatted with byte alignments for COMP-4 data type. All COBOL data types are supported.	

Tip: To generate EBCDIC data, select the Delimited or Fixed ASCII format and use the format options to change the code set.

Related reference:

Format options

Alter the schema of a table

Generating load and unload components

When you choose Components > Generate, you can generate all of the components required for a load job and an unload job: format, load map, unload map, query, and device array.

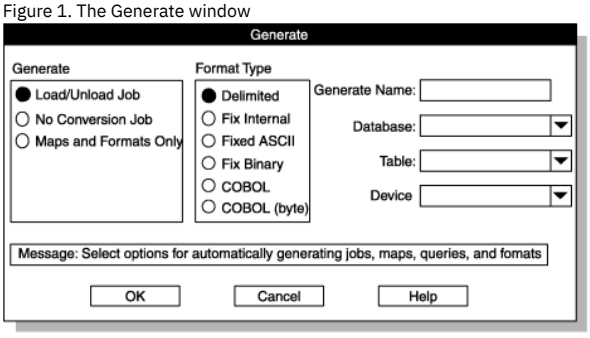
To generate the components for loading or unloading a database:

1. Choose Components > Generate Jobs from the main HPL window to display the Generate window.
2. Click Load/Unload Job in the Generate group.
3. Select a format for the data file in the Format Type group.
4. Select a name for the generated components and type it in the Generate Name text box. This name is used for each of the components that this option creates.
5. Type the name of the database that you will load or unload in the Database text box. Or, click the down arrow to select a database from the selection list.
6. Type the name of the table within the database in the Table text box. Or, click the down arrow to select a table from the selection list.
7. Type the name of a device array in the Device text box. Or, click the down arrow to select a device from the selection list.
   If you enter the name of a device (file) instead of a device array, **ipload** creates a device array of the same name as the unload job and inserts the specified device into that device array.

   When you specify a file instead of a device array in the Device text box, **ipload** makes the following assumptions about the data file:
   - The file is an ASCII file.
   - The file uses the same locale as the database.
   - The file uses a vertical bar (|) for the field delimiter and a new line for the record delimiter.
   - The fields in the data file are in the same order as the columns of the target table.
8. Click OK.

The following figure shows appropriate choices for generating load and unload jobs for delimited output from the **state** table of the **stores_demo** database. After **ipload** creates the components, you can run the job or use the Component-Definition windows to make any necessary changes.

Figure 1. The Generate window



For information about generating a job from a .csv file, see Testing the import of a CSV file.

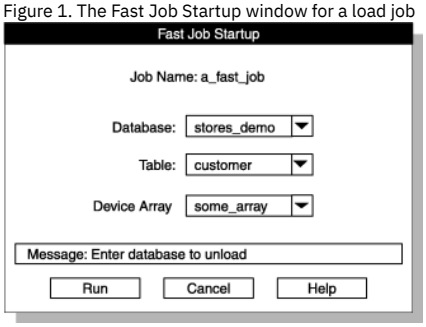
Using the No Conversion Job option

The No Conversion Job option uses the IBM® Informix® internal format to unload data from a table. Jobs loaded or unloaded with this option are sometimes called raw loads or raw unloads.

The No Conversion Job option treats an entire database record as one entity by using the internal format. It does not generate formats or maps. The No Conversion Job option is the fastest option that you can use for loading and unloading data. Use this option to transport data or when you need to reorganize the disks on your computer. No-conversion jobs are always completed in express mode. You cannot use the **onpload** command line to convert the running job to deluxe mode when using a no-conversion job.

When you run a job that you created with the No Conversion Job option, **ipload** displays a Fast Job Startup window instead of the usual Load Job or Unload Job window.

To use the Fast Job Startup window:

1. Choose Components > Generate Job from the main HPL window.
2. Click No Conversion Job in the Generate group.
3. Select a name for the job and type it in the Generate Name text box.
4. Type the names of the database, table, and device array in the Database, Table, and Device text boxes.
5. Click OK. The display returns to the HPL main window.
6. Choose JobsLoad (or Jobs > Unload) from the main HPL window.
   The Load Job Select or Unload Job Select window appears.
7. Select the job from the Job Information list box.
8. Click OK.
   The Fast Job Startup window appears, as the following figure shows.
   Figure 1. The Fast Job Startup window for a load job
   

9. Click Run to run the job and the Active Job window appears.
10. When the Active Job window displays `Job Completed`, click Cancel to return to the main HPL window.

Related reference:
Settings for a no-conversion load or unload job

The HPL browsing options

This section describes the browsing options that are available for the High-Performance Loader (HPL).

- Browsing options
  Use the browsing options of the High-Performance Loader (HPL) to preview records from the data file, review records after you perform a load or unload job, review various files associated with the HPL, and view the status of a load or unload job.

Related reference:
Problems during an unload job
Problems during a load job

Browsing options

Use the browsing options of the High-Performance Loader (HPL) to preview records from the data file, review records after you perform a load or unload job, review various files associated with the HPL, and view the status of a load or unload job.

- Preview data-file records
  Before you actually run a load job, you can use the Record Browser window to check your definition of the format. The display shows errors such as incorrect field lengths or missing fields. You can edit the format to correct your format definitions.
- Reviewing records that the conversion rejected
  When you run a load job, **onpload** creates a file that contains information about records of the data file that the conversion rejected.
- Viewing the violations table
  The load job also creates two tables in the target database that contain information about records that passed the conversion but that the database server rejected. The tables are named ***tablename*_vio** and ***tablename*_dia**, where *tablename* is the name of the table being loaded.
- View the status of a load job or unload job
  When a load or unload job is complete, **onpload** writes a record of the load or unload job into a log file.

Related concepts:
The Browse button
Related reference:
View error records

Preview data-file records

Before you actually run a load job, you can use the Record Browser window to check your definition of the format. The display shows errors such as incorrect field lengths or missing fields. You can edit the format to correct your format definitions.

- Using the Record Browser window
  You can review records in a specified format, search the list of available formats, or edit a format from the Record Browser window.

Related concepts:
Formats of supported datafile records
Related tasks:
Editing a format
Related reference:
Run the load job

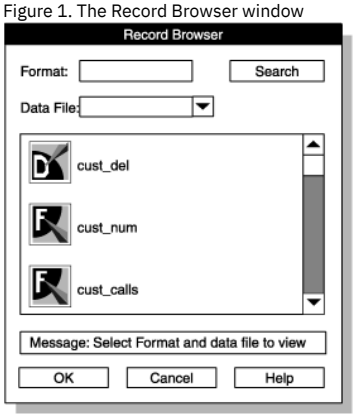
Using the Record Browser window

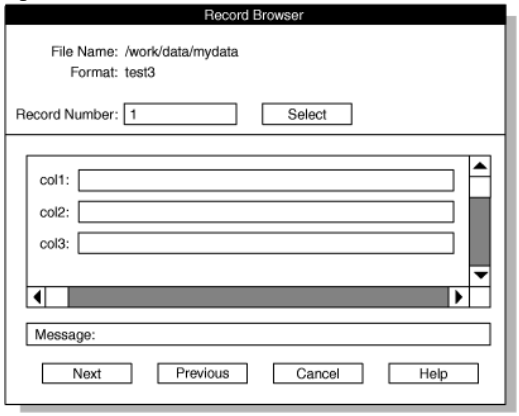
You can review records in a specified format, search the list of available formats, or edit a format from the Record Browser window.

- Reviewing data-file records in a selected format
- Searching and editing a format
- Editing a format

Reviewing data-file records in a selected format

To review data-file records in a selected format:

1. In the HPL main window, select the project that contains your load job.
2. Choose Browsers > Record.
   The Record Browser window appears, as the following figure shows.
   Figure 1. The Record Browser window


3. Type the name of the format to be applied to the source data file or click the format name in the list box.
4. In the Data File text box, type the name of the data file that you plan to load, or click the down arrow and select a file from the selection list.
5. Click OK.
   The second Record Browser window appears, as the following figure shows. This Record Browser window displays each of the fields in the format, followed by the value of the field for the given Record Number.
   Figure 2. The Record Browser window


6. You can take the following actions:
   - Type the record number that you want to view. Click Select.

- Click Next to display the next record.
- Click Previous to display the previous record.

7. When you finish browsing, click Cancel to return to the HPL main window.

Searching and editing a format

To search for and edit a format:

1. In the HPL main window, select the project that contains your load job.
2. Choose Browsers > Record to display the Record Browser window.
3. In the Format text box, type the format name or partial format name that you want to find. You can use wildcards (for example, *`cust`*).
4. Click Search. The **ipload** utility displays all formats of the current project that include the letters *cust*.
5. Click Cancel to return to the HPL main window.

Editing a format

To edit a format:

1. Select the project that contains your load job.
2. Choose Browsers > Record from the HPL main window.
3. Click a format button to edit the format. The **ipload** utility displays the Format-Definition window.

Related tasks:
Editing a format

Reviewing records that the conversion rejected

When you run a load job, **onpload** creates a file that contains information about records of the data file that the conversion rejected.

This file is named *basename*.rej where *basename* is the base name that you selected in step 7 of Creating a load job. When you use a generate option to create the components for a load job, *basename* is /tmp/*jobname* where *jobname* is the name that you selected for the unload job.

To review rejected records:

1. On the HPL main window, select the project that contains your load job.
2. Choose Browser > Record to display the Record Browser window.
3. Type the name of the format to be applied to the rejected-records file in the Format text box. Or, click the format name in the list box.
4. Type the name of the rejected-records file in the Data File text box. Or, click the down arrow and select a data file from the selection list.
5. Click OK to display the second Record Browser window.
6. You can take the following actions:
   - Type the record number that you want to view. Click Select.
   - Click Next to display the next record.
   - Click Previous to display the previous record.
7. Click Cancel to return to the HPL main window.

Viewing the violations table

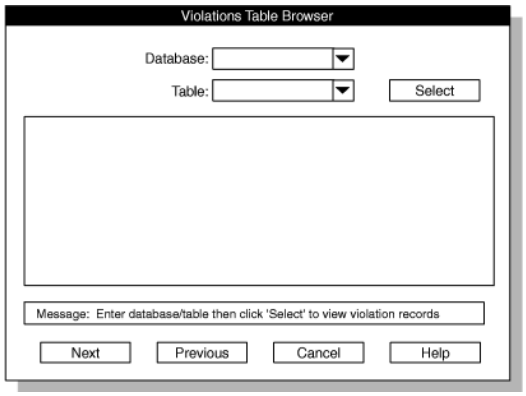
The load job also creates two tables in the target database that contain information about records that passed the conversion but that the database server rejected. The tables are named ***tablename*_vio** and ***tablename*_dia**, where *tablename* is the name of the table being loaded.

View the violations table to browse through records that passed the filter and conversion but that the database server rejected. The High-Performance Loader (HPL) writes these records into the violations table (***tablename*_vio**). The data in the violations table has the same format as the database table.

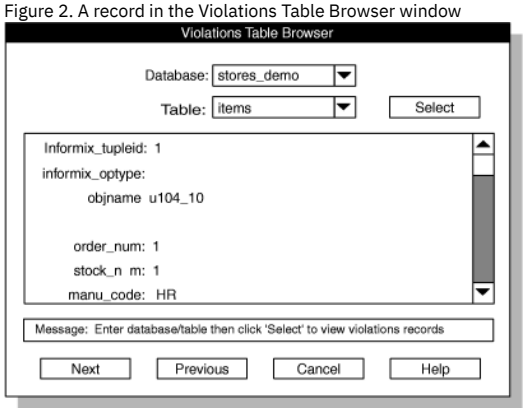
The *IBM® Informix® Guide to SQL: Syntax* discusses in detail the information found in the violations table.

To view the violations table:

1. Choose Browsers > Violations from the HPL main window.
   The Violations Table Browser window appears, as the following figure shows.
   Figure 1. The Violations Table Browser window



2. Type the name of the database and table that you want to review the violations or click the down arrows to make your choices from selection lists.
3. Click Select.
   The following figure shows the first record of a violations table.
   Figure 2. A record in the Violations Table Browser window



4. Click Next and Previous to move forward and backward through the violations table.
5. Click Cancel to return to the HPL main window.

View the status of a load job or unload job

When a load or unload job is complete, **onpload** writes a record of the load or unload job into a log file.

- Viewing the log file
  The default name for a log file is /tmp/*jobname*.log, where *jobname* is the name that you chose for the job. You can specify a different name for the log file in the Load Job window or Unload Job window.
- Sample log file

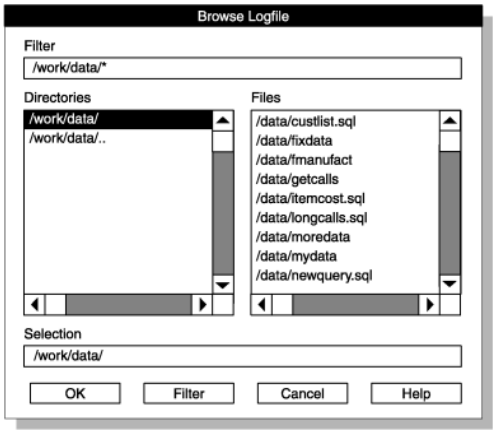
Related reference:
Run the unload job
The HPL log-file and pop-up messages

Viewing the log file

The default name for a log file is /tmp/*jobname*.log, where *jobname* is the name that you chose for the job. You can specify a different name for the log file in the Load Job window or Unload Job window.

To view the log file:

1. Choose Browsers > Logfile from the HPL main window.
   The Browse Logfile window appears, as the following figure shows. When the window appears, the Filter text box and the Selection text box show the directory from which **ipload** was started.
   Figure 1. The Browse Logfile window



2. In the Filter text box, type the full path name of the directory that contains the log. You can use wildcards to select only certain files from that directory.
3. Click Filter.
   The Files list box shows a list of the files that match the path that you entered in the Filter text box.
4. In the Files list box, click the name of the file that you want to examine. The full path name of the selected file appears in the Selection text box.
5. Click OK.
   A Browse window appears that displays the contents of the selected file.
6. Review the log with the scroll bar to move through the log.
7. Click Cancel to return to the HPL main window.

Alternatively, if you know the full path name of the log file, you can simply type the path name in the Selection text box and click OK.

Sample log file

The following example shows a sample log file entry:

```
Sat Mar 11 13:52:42 1995

Session ID 1

Unload Database -> stores_demo
Query Name  -> f_manufact
Device Array-> fmanufact
Query Mapping-> f_manufact
Query-> select * from manufact
Convert Reject  -> /work/data/f_manu_unl

Database Unload Completed -- Unloaded 9 Records  Detected 0 Errors

Sat Mar 11 13:52:50 1995
```

You can review the log file to determine load status and to see where any errors occurred. The log file is a simple ASCII file. You can print it if necessary.

Manage the High-Performance Loader

- Manage modes, errors, and performance
  When you manage the High-Performance Loader (HPL), you manage modes, violations (errors), and performance. You also need to avoid the limitation that occurs when using an Excalibur Text DataBlade module index
- HPL modes
  The High-Performance Loader (HPL) offers two load modes, deluxe mode and express mode, and one unload mode. The express mode is faster, and the deluxe mode is more flexible.
- HPL load and unload errors
  When you load records from a data file, some of the records might not meet the criteria that you established for the database table.
- HPL performance
  You can improve the HPL performance by preparing an environment that is optimized for the particular load or unload job that you are performing.
- Limitation when using the Excalibur Text DataBlade Module indexes
  You cannot create a load job with multiple devices to insert rows into a table that has an Excalibur Text DataBlade module index. A multiple-device load fails because the Excalibur Text DataBlade module does not handle concurrency correctly.

Related concepts:
HPL loading modes

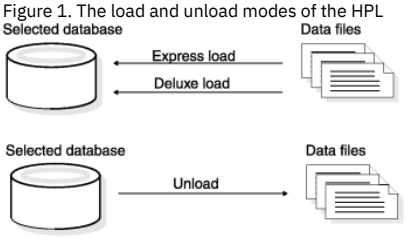
Manage modes, errors, and performance

When you manage the High-Performance Loader (HPL), you manage modes, violations (errors), and performance. You also need to avoid the limitation that occurs when using an Excalibur Text DataBlade module index

HPL modes

The High-Performance Loader (HPL) offers two load modes, deluxe mode and express mode, and one unload mode. The express mode is faster, and the deluxe mode is more flexible.

The following figure shows the load and unload modes of the HPL.
Figure 1. The load and unload modes of the HPL



- The HPL deluxe mode
- The HPL express mode
  The express mode for loading is faster than the deluxe mode, but has some limitations
- How the express and deluxe modes work

Related concepts:
Express-mode limitations

The HPL deluxe mode

The deluxe mode has the following features:

- Performs row-by-row referential and constraint checking as the data is loaded
- Allows loading of data while other users are working (no table locking) if the table already has an associated violations table before the load and if either of the following conditions apply:
  - The index is set to FILTERING WITHOUT ERROR mode
  - The table does not have unique indexes.
- Allows users to access and update the table during a load
  Loaded data is immediately visible to the user.

- Logs data, but also offers a no-logging option
- Updates indexes
- Evaluates triggers
- Sets constraints to FILTERING WITHOUT ERROR
- Sets the isolation mode as if for an insert cursor
- Simulates an INSERT statement, except that the HPL allows the load to handle parallel data streams
- Allows loading of data without replication

The deluxe mode has the following limitations:

- Does not support loads without conversion
- Does not support loads that have conversion but do not generate a violations table
- Does not support the **-fv** option

When you use the deluxe mode without replication to load data into the table that was used to define the replicates, the data in that table is not replicated.

Related concepts:
The HPL express mode
How the express and deluxe modes work
The onpload utility deluxe-mode process
Comparison between an express-mode and a deluxe-mode load operation

The HPL express mode

The express mode for loading is faster than the deluxe mode, but has some limitations

The express mode has the following features:

- Locks tables for exclusive use by the load utility
- Disables referential and constraint checking on the table
- Requires a level-0 backup
- Sets all objects (such as indexes or constraints) to disabled before loading
- Re-enables all objects after loading, if possible, and flags objects that cannot be re-enabled in the violations and diagnostic tables
- Supports loading of raw tables
- Supports the loading of logged databases and ANSI mode databases
  However:
  - If you load ANSI mode databases in express mode, you cannot use DB-Access to select data from the database tables.
  - If you load logged databases in express mode, you must perform a level-0 backup before you can write to the target database.

The express mode does not support:

- Tables that contain smart large objects (BLOB or CLOB data types)
- Tables that contain simple large objects (TEXT and BYTE) or extended data types
- Loading of data on a heterogeneous data replication (HDR) replicated table from a database that has transactions
- A table that has a fragmentation strategy and has a WITH ROW IDs clause to enable by using row IDs with fragmentation
- Tables with primary key constraints when child table records refer to the load table
- Rows that are larger than the system page size
- The static-hash access method
- Triggers on the loaded data, because express mode cannot invoke the triggers

Important: If your load job has any of these conditions, you must use deluxe mode to load your data.
Related concepts:
The HPL deluxe mode
How the express and deluxe modes work
The onload utility express-mode load process
Express-mode limitations
Comparison between an express-mode and a deluxe-mode load operation

How the express and deluxe modes work

A deluxe-mode load simulates an INSERT statement, except that the High-Performance Loader (HPL) allows the load to handle parallel data streams.

The sequence of events when you run an express-mode load is as follows:

1. The **onload** utility locks the table with a shared lock.
   Other users can read data in existing rows.
2. The **onload** utility creates new extents and fills them with the new rows. However, **onload** does not update the database structures that track extents.
   The new extents are not visible to the user.
3. At the end of the express load, **onload** updates the internal structures of the database.
4. The **onload** utility sets the table to read-only.
   This setting occurs because in express mode **onload** does not log data, and therefore, the table is in an unrecoverable state.
5. The **onload** utility unlocks the table and enables the constraints. The new rows become visible to the user for read only.
6. The user performs a level-0 backup.
7. Your database server sets the table to read/write.

If the load fails, **onload** discards the extents and clears the internal information that says the table is unrecoverable.

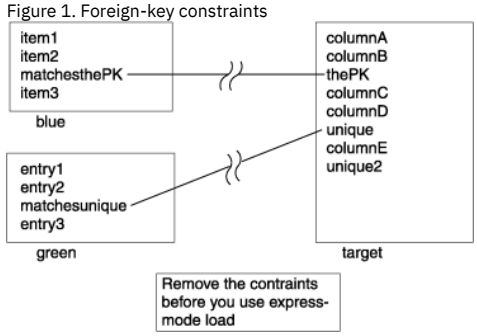
- Foreign-key constraints
  Express® mode cannot disable primary constraints or unique constraints that are referenced as foreign keys that are active on other tables. If you want to load data into such a table, you must first use SET CONSTRAINTS DISABLED statements to disable the foreign-key constraints in the referencing table or tables. After the load is finished, re-enable the foreign-key constraints.
- Comparison between an express-mode and a deluxe-mode load operation

Related concepts:
The HPL deluxe mode
The HPL express mode
Make a level-0 backup
Related information:
INSERT statement
The INSERT statement

Foreign-key constraints

Express® mode cannot disable primary constraints or unique constraints that are referenced as foreign keys that are active on other tables. If you want to load data into such a table, you must first use SET CONSTRAINTS DISABLED statements to disable the foreign-key constraints in the referencing table or tables. After the load is finished, re-enable the foreign-key constraints.

The following figure shows an example of foreign-key constraints. The table **target** has a primary key (**thePK**) and a unique key (**unique**) that table **blue** and table **green** reference. Before you perform an express-mode load into the table **target**, you must disable the foreign-key constraints in both table **blue** and table **green**.
Figure 1. Foreign-key constraints



Related concepts:
Comparison between an express-mode and a deluxe-mode load operation

Comparison between an express-mode and a deluxe-mode load operation

The following table contrasts the operation of an express-mode and a deluxe-mode operation.

Express® mode action	Performed by	Deluxe mode action	Performed by
Set objects to disabled	**onpload**	Set constraints to filtering	**onpload**
Load records from the data file into the table (including rows that would cause violations if constraints were on)	**onpload**	Load records from data file into the table. Write records that violate constraints into the violations table and not into the target table	**onpload**
Enable objects. Detect violators and copy them into the **_vio** table.	**onpload**	Set constraints to non-filtering	**onpload**
Complete a level-0 backup to make the database writable	user		
Resolve violators	user	Resolve violators	user

Related concepts:
Foreign-key constraints
The HPL deluxe mode
The HPL express mode

HPL load and unload errors

When you load records from a data file, some of the records might not meet the criteria that you established for the database table.

For example, the data file might contain:

- Null values where the table specifies NOT NULL
- Values in an incorrect format (for example, alphabetic characters in a numeric field)
- Records that do not have the expected number of fields

The way that the HPL treats these errors depends on the mode (deluxe or express) and the type of job (load or unload).

The HPL separates errors into the following two classes:

- Rejected records from the input file
  These records include:
  - Records that the filter rejected
  - Records that cannot be converted
- Constraint violations

- Rejected records from the input file
  Input-file records that the High-Performance Loader (HPL) rejects because they could not be converted include records in an incorrect format, records that do not have the expected number of fields, and records whose fields contain null values for columns that do not allow null values.
- Constraint violations
- View error records
  Choose from the Browsers menu in the HPL main window to view the error records that **onpload** generates.

Rejected records from the input file

Input-file records that the High-Performance Loader (HPL) rejects because they could not be converted include records in an incorrect format, records that do not have the expected number of fields, and records whose fields contain null values for columns that do not allow null values.

The **onpload** utility writes these records into a file with the suffix .rej. The **onpload** utility writes records that are rejected because they do not match the filter criteria into a file with the suffix .flt.

Constraint violations

When the **onpload** utility starts a deluxe-mode load, it runs the following SQL statement:

```
SET CONSTRAINTS ON mytable FILTERING
```

This statement has two results:

- The utility adds two tables, **mytable_vio** and **mytable_dia**, to the database that contains **mytable**.
- All of the constraints that are associated with the table are set to filtering. Filtering mode causes any record that does not meet the constraint requirements to be added to the **mytable_vio** table instead of generating an error.

The use of filtering mode for constraints is covered in detail in the *IBM® Informix® Guide to SQL: Syntax*.

View error records

Choose from the Browsers menu in the HPL main window to view the error records that **onpload** generates.

Related concepts:
Browsing options

HPL performance

You can improve the HPL performance by preparing an environment that is optimized for the particular load or unload job that you are performing.

You must consider the following aspects of your load and unload jobs:

- Configuration-parameter values
- Mode (express or deluxe)
- Devices for the device array
- Usage models

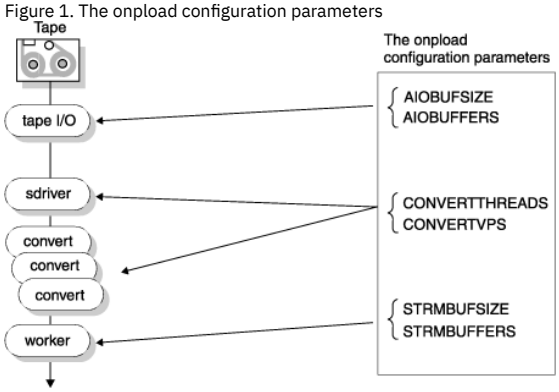
- The onpload configuration parameters
  The **onpload** configuration parameters control the number of threads that **onpload** starts and the number and size of the buffers that are used to transfer data.
- Express-mode limitations
- The onstat options for onpload
  The **onstat** utility has two options that you can use to observe the behavior of database server threads during express-mode loads.
- Devices for the device array
  On a data-load job, each device runs independently of other devices. Thus mixing fast and slow devices does not adversely affect the speed of the load.
- HPL usage tasks
- Settings for a no-conversion load or unload job
  A no-conversion load or unload is not highly CPU intensive because no conversion is involved. In this case, the load or unload is expected to be limited by the speed of the tape devices. No-conversion loads and unloads are always completed in express mode.
- An express-mode load with delimited ASCII
  In an express-mode load with delimited ASCII, the load or unload job is limited by the available CPU. The conversion to or from IBM® Informix® types to ASCII is the most expensive portion of these operations
- HPL performance hints
  In general, the performance of the High-Performance Loader (HPL) depends on the underlying hardware resources: CPU, memory, disks, tapes, controllers, and so on. Any of these resources could be a bottleneck, depending on the speed of the resource, the load on the resource, and the particular nature of the load or unload.

Related reference:
Configure the ipload utility

The onpload configuration parameters

The **onpload** configuration parameters control the number of threads that **onpload** starts and the number and size of the buffers that are used to transfer data.

The following figure shows which part of the **onpload** process is affected by each configuration parameter.
Figure 1. The onpload configuration parameters



The AIOBUFSIZE and AIOBUFFERS parameters control the number and size of the buffers that **onpload** uses for reading from the input device. CONVERTTHREADS and CONVERTVPS control the amount of CPU resources to apply to data conversion. STRMBUFFSIZE and STRMBUFFERS control the number and size of the buffers used to transport data between **onpload** and the database server.

The **onpload** configuration parameters are stored in the following files:

- $INFORMIXDIR/etc/$PLCONFIG (UNIX)
- %INFORMIXDIR%\etc\%PLCONFIG (Windows)

Related reference:
HPL usage tasks
HPL performance hints

High-Performance Loader configuration file

Express-mode limitations

You cannot use express mode in certain situations. For example, express mode does not support the loading of simple large objects (Simple LOs) or Ext Type Data and smart-large-object data (Ext Type data types), ensuring constraints, or invoking triggers.

In addition, you cannot use DB-Access to select data from ANSI database tables if the data was loaded in express mode. If you attempt to select data from such tables, the database server rejects the SELECT statements because the tables were not archived. Only read-only access is allowed to the tables until a level-0 archive is performed.

Related concepts:
The HPL express mode
HPL modes

The onstat options for onpload

The **onstat** utility has two options that you can use to observe the behavior of database server threads during express-mode loads.

Use the following command to display light-append information:

```
onstat -g lap
```

The **onstat -j** option provides an interactive mode that lets you gather special information about an **onpload** job.

Related reference:
The onstat -j option

Devices for the device array

On a data-load job, each device runs independently of other devices. Thus mixing fast and slow devices does not adversely affect the speed of the load.

In most unload jobs, all devices receive equal amounts of data. Thus the speed of all devices is limited by the speed of the slowest device. If you have several fast devices and one or two slow devices, it might be advantageous to remove the slow devices.

When CPU resources are plentiful during an HPL job, the device controllers are a potential bottleneck. If you have configured extra converter threads and extra converter VPs, CPU use should be close to 100 percent. If CPU use is not close to 100 percent, the cause might be one of the following situations:

- The device controller is managing too many devices.
- The devices themselves are slow.

Related reference:
Define device arrays

HPL usage tasks

Three tasks that you might need to perform for the High-Performance Loader (HPL) are:

- Reorganizing computer configuration
- Altering the schema of a table
- Assessing information for loading or unloading external data

- Reorganize computer configuration
  If you are not changing the table schema, use a No-Conversion Job to unload and load when you need to reorganize the configuration of your computer or change to a different computer. The no-conversion mode is the fastest means of performing an unload job or a load job, because rows are unloaded in IBM® Informix® internal format with no conversion and reloaded in the same fashion.
- Alter the schema of a table
  When you need to alter a table (add, drop, or change the data type of columns), use the Fixed Internal format. In Fixed Internal format, rows are unloaded in IBM Informix internal format on a column-by-column basis. Thus you can drop, add, or modify columns and still minimize conversion overhead.
- Assess information for loading or unloading external data
  When you load or unload data from an external source, you must assess the type of data and the amount of conversion that is required so that you can choose appropriate configuration parameters.

Related concepts:
The onpload configuration parameters

Reorganize computer configuration

If you are not changing the table schema, use a No-Conversion Job to unload and load when you need to reorganize the configuration of your computer or change to a different computer. The no-conversion mode is the fastest means of performing an unload job or a load job, because rows are unloaded in IBM® Informix® internal format

with no conversion and reloaded in the same fashion.

Restriction: You cannot use a No-Conversion Job when the source and target computers use different internal byte representations. For information about the byte-order type, see The Machines window.
For information about preparing for a no-conversion unload/load with **ipload**, see Using the No Conversion Job option. To set no-conversion mode when you are using the **onpload** utility at the command line, use the **-fn** option. For more information, see The onpload utility.

Alter the schema of a table

When you need to alter a table (add, drop, or change the data type of columns), use the Fixed Internal format. In Fixed Internal format, rows are unloaded in IBM® Informix® internal format on a column-by-column basis. Thus you can drop, add, or modify columns and still minimize conversion overhead.

Related reference:
Format type group

Assess information for loading or unloading external data

When you load or unload data from an external source, you must assess the type of data and the amount of conversion that is required so that you can choose appropriate configuration parameters.

The following sections show possible configuration parameters for two different types of load jobs. Suppose your hardware has the following configuration:

- Eight CPU symmetric multiprocessors, each about 40 MIPS
- Several (say 16) 300-MB disks (for the database)
- Four 1.2-MB tape devices (to load to and unload from)
- 512-MB memory

You also have this information about your system:

- The database server is running with seven CPU virtual processors.
- The data that you want to load has a row of about 150 bytes with a mix of **INT**, **DATE**, **DECIMAL**, **CHAR**, and **VARCHAR** data types. (This is similar to the LINEITEM table in the TPC-D database).

Settings for a no-conversion load or unload job

A no-conversion load or unload is not highly CPU intensive because no conversion is involved. In this case, the load or unload is expected to be limited by the speed of the tape devices. No-conversion loads and unloads are always completed in express mode.

The following table lists sample values of configuration parameters for a raw load.

Configuration parameter	Suggested value	Comment
CONVERTVPS	4	This process is not CPU intensive.
CONVERTTHREADS	1	This process is not CPU intensive.
STRMBUFFSIZE	128	Choose some multiple of the AIOBUFSIZE, up to about 8*AIOBUFSIZE.
STRMBUFFERS	5	Should be CONVERTTHREADS + 4.
AIOBUFSIZE	32	Choose a buffer size to match the best block size for the tape drive. Large buffers increase performance if sufficient memory is available.
AIOBUFFERS	5	CONVERTTHREADS + 4 or 2 * CONVERTTHREADS, whichever is larger

- Run no-conversion load jobs with tables with hidden columns
  If you run no-conversion jobs, the physical image of the table in the load file must match the physical image of the target table and all columns must exist in the same order and be of the same type.

Related tasks:
Using the No Conversion Job option

Run no-conversion load jobs with tables with hidden columns

If you run no-conversion jobs, the physical image of the table in the load file must match the physical image of the target table and all columns must exist in the same order and be of the same type.

By default a no-conversion unload job does not contain the hidden columns, because the job resulted from unloading a "SELECT *" query and this type of SELECT query does not return any hidden columns. Under these circumstances, the target table where the load will occur must not contain any hidden columns. If it does contain hidden columns, the image of the file will not be a match.

If your application requires that the unload file contains the hidden columns (for example, if the target table has hidden columns and it is not possible to alter the table to remove the columns while the load occurs) , you must use a SELECT statement that explicitly indicates the columns in the select list.

The target table should not contain hidden columns, because these columns retain internal information for internal activities of the engine. Loading hidden columns can create unexpected results.

Hidden columns tables include but are not restricted to fragmented tables with rowid columns, tables with VERCOLS, and tables with CRCOLS. Also fragmented tables with rowid columns do not support no-conversion jobs.

An express-mode load with delimited ASCII

In an express-mode load with delimited ASCII, the load or unload job is limited by the available CPU. The conversion to or from IBM® Informix® types to ASCII is the most expensive portion of these operations

Because this conversion is expensive, the following configuration might be more appropriate. The following table lists sample values of configuration parameters for an express-mode load with delimited ASCII.

Configuration parameter	Sample value	Comment
CONVERTVPS	8	One convert VP per CPU
CONVERTTHREADS	2	Four devices * 2 thread/device = 8 threads
STRMBUFFSIZE	32	Should be some multiple of AIOBUFSIZE. For this CPU-intensive case, 1 or 2 * AIOBUFSIZE is sufficient.
STRMBUFFERS	4	CONVERTTHREADS + 4
AIOBUFSIZE	32	Choose a buffer size to match the best block size for the tape drive. Large buffers increase performance if sufficient memory is available.
AIOBUFFERS	5	CONVERTTHREADS + 4 or 2 * CONVERTTHREADS, whichever is larger

HPL performance hints

In general, the performance of the High-Performance Loader (HPL) depends on the underlying hardware resources: CPU, memory, disks, tapes, controllers, and so on. Any of these resources could be a bottleneck, depending on the speed of the resource, the load on the resource, and the particular nature of the load or unload.

For example, load and unload jobs that perform no conversions consume minimal CPU resources. These jobs are thus likely to be limited by device or controller bandwidth. Alternatively, ASCII loads and unloads are CPU intensive because of the overhead of conversion to and from ASCII. This section discusses some topics that you should consider when you try to improve performance.

- Choose an efficient format
  When you load data to or unload data from a source that is not IBM® Informix®, you can use fixed or delimited format in an appropriate code set such as ASCII or EBCDIC.
- Ensure enough converter threads and VPs
  Loads and unloads other than raw and fast-format ones are likely to be CPU intensive due to conversion overhead. In such cases, conversion speed is likely to determine the load or unload speed. It is thus important to use sufficient conversion resources (that is, enough converter threads and VPs).
- Ensure enough memory
  To maximize available memory and scan resources, the High-Performance Loader (HPL) automatically sets the PDQPRIORITY environment variable to `100`, if it is not already set. If the PDQPRIORITY environment variable is set, HPL uses that value.
- Ensure enough buffers of adequate size
  You set the size and number of buffers in the plconfig configuration file with the **onpload** utility.
- Increase the commit interval

Related concepts:
The onpload configuration parameters

Choose an efficient format

When you load data to or unload data from a source that is not IBM® Informix®, you can use fixed or delimited format in an appropriate code set such as ASCII or EBCDIC.

In general, ASCII loads and unloads are the fastest. If you are using **onpload** for a machine or schema reorganization, choose the no-conversion format. Delimited and fixed ASCII formats are comparable in behavior except when VARCHAR data is present. If the schema contains VARCHAR data and the length of the VARCHAR data varies greatly, you might want to choose delimited format.

Related concepts:
Ensure enough converter threads and VPs
Ensure enough memory
Related reference:
Ensure enough buffers of adequate size
Increase the commit interval

Ensure enough converter threads and VPs

Loads and unloads other than raw and fast-format ones are likely to be CPU intensive due to conversion overhead. In such cases, conversion speed is likely to determine the load or unload speed. It is thus important to use sufficient conversion resources (that is, enough converter threads and VPs).

The number of converter threads that is required for a device depends on the relative speeds of the device and the CPU as well as the data types in the table being loaded or unloaded. CHAR and VARCHAR formats are the cheapest to convert. INT, DATE, SMFLOAT, and FLOAT are more expensive. DECIMAL and MONEY are among the most expensive formats to convert.

The plconfig file specifies the number of converter threads per device. You can override this value on the **onpload** command line with the **-M** option.

The number of converter VPs should be based on the conversion intensity of the load or unload and the number of physical CPUs on the computer. If the load or unload is expected to be highly intensive, you might want to specify the number of convert VPs to be the number of physical CPUs (or one fewer) to take advantage of all of the available CPUs. You can set the number of converter VPs in the **onpload** configuration file.

The database server and **onpload** client VPs might both be competing for the same physical CPU resources. To reduce contention, run only the number of VPs that are necessary on both the database server and **onpload** sides. However, if the number of database server VPs is already specified, you might have a choice only in the number of **onpload** VPs. In this case, the suggestions in the previous paragraph apply.

Too few converter threads and VPs can make conversion a bottleneck. However, too many converter threads can waste time in scheduling threads in and out of the VPs. In general, more than ten converter threads per VP is too many.

Related concepts:
Choose an efficient format
Ensure enough memory
Related reference:
Ensure enough buffers of adequate size
Increase the commit interval

Ensure enough memory

To maximize available memory and scan resources, the High-Performance Loader (HPL) automatically sets the PDQPRIORITY environment variable to `100`, if it is not already set. If the PDQPRIORITY environment variable is set, HPL uses that value.

If the PDQPRIORITY environment variable is set to `0` or if the PDQPRIORITY environment variable is disabled on the database server, then HPL cannot unload multiple devices.

Related concepts:
Choose an efficient format
Ensure enough converter threads and VPs
Related reference:
Ensure enough buffers of adequate size
Increase the commit interval

Ensure enough buffers of adequate size

You set the size and number of buffers in the plconfig configuration file with the **onpload** utility.

For adequate performance, you should provide at least two (preferably three) AIO and stream buffers per converter thread. The size of AIO buffers should be at least as large as the device block size, and the size of the stream buffers should be large (32 or 64 KB), as the following table shows.

Configuration Parameter	Size	Comment
STRMBUFFSIZE	64	Should be some multiple of AIOBUFSIZE for best performance
STRMBUFFERS	2 * CONVERTTHREADS	
AIOBUFSIZE	64	
AIOBUFFERS	2 * CONVERTTHREADS	

Related concepts:
Choose an efficient format
Ensure enough converter threads and VPs
Ensure enough memory
Related reference:
Increase the commit interval

Increase the commit interval

In the deluxe modes, the commit interval specifies the number of records that should be loaded before the transaction is committed (see Figure 1). Low values (frequent commits) degrade performance and high values improve performance. If you increase the commit interval, you might need to increase the size of your logical-log buffers.

While larger commit intervals can speed up loads, larger commit intervals require larger logical-log space and increase the checkpoint time. These side effects impact other system users during **onpload** operations.

Related concepts:
Choose an efficient format
Ensure enough converter threads and VPs
Ensure enough memory
Related reference:

Ensure enough buffers of adequate size

Limitation when using the Excalibur Text DataBlade Module indexes

You cannot create a load job with multiple devices to insert rows into a table that has an Excalibur Text DataBlade module index. A multiple-device load fails because the Excalibur Text DataBlade module does not handle concurrency correctly.

To avoid this limitation:

- Create a load with a single device.
- Complete the load by using multiple devices and then create the Excalibur Text DataBlade module index on the table.

The onpload utility

This section describes how to use the **onpload** utility.

- Overview of the onpload utility
  After you create the onpload database with the **ipload** interface, the **onpload** utility allows you to perform loads and unloads directly from the command line. However, you cannot load or unload binary large objects (BLOBs) or character large objects (CLOBs) to or from multiple files.
- The onpload file name size limitations on UNIX
  On UNIX, you cannot have a file name size greater than 256 characters.
- Start the onpload utility
  You can start **onpload** from the command line or from **ipload**.
- Using the onpload utility
- The onpload utility syntax
- Set the onpload run mode with the -f option
  The **-f** option lets you set the type of source data and the type of mode. Possible modes are express load, deluxe load, deluxe load without replication, or unload. If used along with the **-j** option, you can override only a few values from the **onpload** database.
- Modify the size of onpload database parameters
  The options that are described in this section let you enter size information that overrides existing parameters in the onpload database.
- Override the onpload database values
  The options that are described in this topic let you enter size information that overrides existing parameters in the onpload database. You can override only these parameters. You cannot override other options, such as run mode.
- Load data into collection data type columns

Related concepts:
The onpload utility
Unload jobs

Overview of the onpload utility

After you create the onpload database with the **ipload** interface, the **onpload** utility allows you to perform loads and unloads directly from the command line. However, you cannot load or unload binary large objects (BLOBs) or character large objects (CLOBs) to or from multiple files.

The **onpload** command uniquely specifies a row in the **session** table in the **onpload** database. Each row in the **session** table specifies all the components and options that are associated with a job.

The onpload file name size limitations on UNIX

On UNIX, you cannot have a file name size greater than 256 characters.

Suppose you have a 128-character database name and a 128-character table name and automatically generate the project (auto job generation), the generated device file names have the following format:

```
directory_specified/databasename_tablename.unl
```

In this situation, the device name would exceed 256 characters and the job would not run. To work around this problem, modify the device file names (assuming the specification file name is device.hpl):

1. Describe the device with the following command:

   ```
   onpladm describe device devicename -F device.hpl
   ```

2. Modify the device.hpl specification file to shorten the names of .unl files.
3. Modify the device attributes with the following command:

   ```
   onpladm modify device devicename -F device.hpl
   ```

Alternatively, do not use the auto generation option of **onpladm** when you have both database and table names that would produce a generated name exceeding the 256 UNIX file name limit. Instead, manually provide a device file name.

Related reference:
The onpladm utility

Start the onpload utility

You can start **onpload** from the command line or from **ipload**.

When you click Run in the Load Job window (Figure 2) or in the Unload Job window (Figure 2), **ipload** uses information from the **onpload** database to start **onpload**.

Typically, you use **ipload** to start a job when you plan to perform that particular load or unload once. For a job that needs to be run periodically, such as a weekly report, you might choose to run the job from the command line.

The information that you give to **onpload** as command-line arguments takes precedence over any information that is in the **onpload** database. The information from the command-line arguments is effective only for a single load. The command-line arguments do not affect the values in the **onpload** database or in the plconfig configuration file.

Using the onpload utility

In most cases, use the Load Job window (Figure 2) or Unload Job window (Figure 2) to prepare the load or unload job.

After you prepare the job, the Command Line text box on the Load Job Select window (Figure 1) or the Unload Job Select window (Figure 1) shows you the command line that **ipload** prepared for the job. You can copy that command line and use it to run the job at a later time. You can also use the command-line options shown in this section to modify the basic command line that **ipload** prepared.
Tip: Enter **onpload** with no options at the command line to display a command-line listing of all **onpload** options and their functions.
When you use the **onpload** utility, you cannot load or unload smart large objects (BLOB or CLOB data types) to or from multiple files.

The **onpload** and **onpladm** utilities include support for long object names up to 128 characters, but the **ipload** utility does not.

The following sections give additional information about the syntax and individual options of the **onpload** utility.

The onpload utility syntax

```
>>-onpload------------------------------------------------------>

>--+- -V-------------------------------------------------------------+-><
   '-+- -j--jobname--+-------------+--+------------------+------------+-'
     |               '- -d--source-'  '- -p--projectname-'            |
     '- -m--map-- -d--source--+------------------+--| Other Options |-'
                              '- -p--projectname-'

Other Options

|--| Setting the run mode |--| Modifying parameter size |------->

>--| Overriding database values |--+-----+----------------------|
                                    '- -Z-'
```

For more information about the options shown in the diagram, see Set the onpload run mode with the -f option, Modify the size of onpload database parameters, and Override the onpload database values.

The following table contains an explanation of the elements in the diagram.

Element	Purpose	Key considerations
-V	Displays the current version number and the software serial number.	This option is available only from the command line.
-d *source*	Sets the path name of the file, tape, or pipe (UNIX only) or the name of the device array to use for the load or unload session	If the **-f** option is not set to **a**, **d**, or **p**, **onpload** assumes that the data source is a file. To use **ipload**, see Interpret the onpload -d and -f options together.
-j *jobname*	Names a load or unload job from the **onpload** database	When using the **-j** option, you can override only a few values from the onpload database. See Override the onpload database values To set by using **ipload**, see Components of the unload job and Components of the load job.
-m *map*	Names a map from the **onpload** database	To use **ipload**, see Figure 1.
-p *projectname*	Identifies the project where the format and map are stored	To use **ipload**, see Project organizationIf you use this feature, you must use it for both the load and unload commands. Otherwise, the unloaded data might not match the loaded data.
-Z	Enables writing to or reading from a tape until the end of the device. The **onpload** utility prompts you for additional tapes until the load or unload is completed	If the **-Z** option is not set, the **onpload** uses the tape size provided on the command line. If the **-Z** option is set, it supersedes the tape size information provided. This option is equivalent to checking the Write/read to/from tape until end of device check box on the Load Job Select or Unload-Job Select windows.

The **pload** command line assumes the default project unless otherwise specified with the **-p** option.

For example, you might use the Load Job window to prepare the following command:

```
onpload -j bigload -p zz
```

If you receive a tape that you know contains data with bugs, you might choose to modify the command to allow errors and to save the log in a special place, as follows:

```
onpload -j bigload -p zz -fl -e 1000 -l /mylogs/buggytape.log
```

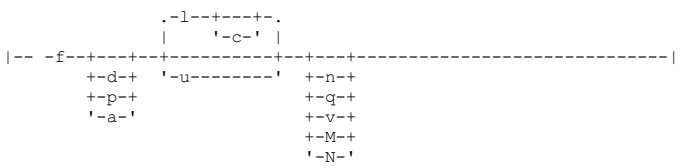
For information about the **-fl** option, see Set the onpload run mode with the -f option.

Set the onpload run mode with the -f option

The **-f** option lets you set the type of source data and the type of mode. Possible modes are express load, deluxe load, deluxe load without replication, or unload. If used along with the **-j** option, you can override only a few values from the **onpload** database.

See Override the onpload database values.

Setting the Run Mode



Element	Purpose	Key considerations
M	Displays the program module or line number in messages.	This flag is available only from the command line. This flag is used for debugging.
N	Allows deluxe mode load without replication	This flag works only if the **-j *job*** argument, which specifies a session table job to run, is left blank.
a	Treats data source as a device-array.	The definition of the device array is extracted from the onpload database. To use **ipload**, see Device arrays.
c	Sets mode to deluxe mode.	If this flag is not set, **onpload** uses express mode. To use **ipload**, see HPL modes.
d	Treats data source as a device (tape or file).	To set this option by using **ipload**, see Device arrays.
l	Loads data into database.	This is the default flag, as opposed to **u**, which unloads data from the database. To use **ipload**, see Components of the load job.
n	Specifies that **onpload** does not need to perform data conversion.	The target table for the load must have the same schema as the table from which the data is extracted. If **onpload** generated the input data file as a data file in IBM® Informix® format, you do not need to perform data conversion when you reload data. To use **ipload**, see Using the No Conversion Job option.
p	Treats data source as a program to run and reads interface to the program by way of a pipe (on UNIX only).	To use **ipload**, see Device arrays.
q	Tells **onpload** not to generate status messages while a job is running.	None.
u	Unloads data from database.	If this flag is not set, **onpload** loads data into the database. To use **ipload**, see Components of the unload job.
v	Tells **onpload** not to generate violations records.	This flag is available only from the command line, and is not supported with deluxe mode load.

- Type the onpload -f flags
  When you combine **-f** flags into one group, do not put spaces between the flags. For example, use **-f acq**.
- Interpret the onpload -d and -f options together

Type the onpload -f flags

When you combine **-f** flags into one group, do not put spaces between the flags. For example, use **-f acq**.

If you prefer, you can use multiple occurrences of the **-f** option instead of combining all of the possible **-f** flags into one group. For example, the following two command lines are equivalent:

```
onpload -m mymap -d mydev -flnc
```

```
onpload -m mymap -d mydev -fl -fn -fc
```

Interpret the onpload -d and -f options together

The argument of the **-d** option gives the name of the data source. You can specify the device type of the data source with flags of the **-f** option, as follows:

- If the command line does not specify a device type, **onpload** treats the data source as the path name of a cooked file on disk. Because no device type is specified, the following **onpload** command treats **filename** as the name of a file:

  ```
  onpload -m mapname -d filename
  ```

- The **-fd** option in the following command causes **onpload** to treat /dev/rmt/rst11b as the name of a tape device:

  ```
  onpload -m mapname -d /dev/rmt/rst11b -fd
  ```

  The tape device name must be Berkeley Software Distribution (BSD) compliant.

- The **-fa** option in the following command causes **onpload** to treat **tapearray3** as the name of a device array. The device array is described in the onpload database.

  ```
  onpload -m mapname -d tapearray3 -fa
  ```

- In an UNIX environment, the **-fp** option in the following command causes **onpload** to treat **apipename** as the name of a pipe. When **onpload** starts executing, it causes the pipe process to start executing.

  ```
  onpload -m mapname -d apipename -fp
  ```

The same semantics apply for an unload job. If you use the **u** flag of the **-f** option to indicate an unload job, the interpretation of the data-source name is as described previously. For example, the following command specifies that **onpload** should unload data to the device /dev/rmt/rst11:

```
onpload -m mapname -d /dev/rmt/rst11 -fdu
```

Modify the size of onpload database parameters

The options that are described in this section let you enter size information that overrides existing parameters in the onpload database.

```
Modifying Parameter Size

|--+--------------------+--------------------------------------|
   +- -A--tapehead------+
   +- -B--blocksize-----+
   +- -G--swapbytes-----+
   +- -I--commit_int----+
   +- -a--iobufsize-----+
   +- -b -bufsize-------+
   +- -e--maxerrors-----+
   +- -i--prog_interval-+
   +- -n--numrecs-------+
   +- -s--startrec------+
   '- -t--numtapes------'
```

Element	Purpose	Key considerations
-A *tapehead*	Tells **onpload** to skip the specified number of bytes on the tape before it starts reading data records.	This option is available only from the command line. For specific details on this option, see The session table in the onpload database.
-B *blocksize*	Sets the tape I/O block size (bytes).	If the data source is a device array, this setting is ignored. To use **ipload**, see Figure 2.
-G *swapbytes*	Sets the number of bytes in a swap group.	This option is available only from the command line. This option globally reverses the byte order in the input data stream. Each group of bytes is swapped with the group of bytes that follows it.
-I *commit_int*	Sets the number of records to process before doing a commit.	This option applies only to deluxe mode. To use **ipload**, see Figure 1.
-a *iobufsize*	Sets the size (kilobytes) of the asynchronous I/O buffers, the memory buffers used to transfer data to and from tapes and files.	This option is available only from the command line This value overrides the value (AIOBUFSIZE) set in the HPL configuration file (plconfig). For specific details on this option, see The AIOBUFSIZE configuration parameter.
-b *bufsize*	Sets the size (kilobytes) of the server stream buffer, the memory buffer used to write records to the database.	This option is available only from the command line. Larger buffers result in more efficient data exchange with the database. This value overrides the value (STRMBUFFSIZE) set in the HPL configuration file (plconfig). For specific details on this option, see The STRMBUFFSIZE configuration parameter.
-e *maxerrors*	Sets the error threshold that causes the load or unload session to shut down.	If no number is specified, the default is to process all records. To use **ipload**, see Figure 1.

Element	Purpose	Key considerations
-i *prog_interval*	Sets the number of records to process before making an entry in the log file specified by the **-l** option.	This option is available only from the command line. If no log file is specified, progress messages are sent to **stdout**. If the **-i** option is omitted, the default number is `1000`. To set, see The onpload -i option.
-n *numrecs*	Sets the number of records to load.	If no number is specified, all records are processed. The **-n** option does not affect unload operations because pload cannot maintain row ordering. To use **ipload**, see Figure 1.
-s *startrec*	Sets the starting record to load.	This option is used to skip records. For example, setting **-s** to 10 starts the loading at the 10th record, so that if the file contains 20 records, 11 records are loaded. Setting **-s** to 0 or 1 the load starts with the first record. If you do not set this option, the load starts with the first record. The **-s** option does not affect unload operations because pload cannot maintain row ordering. To use **ipload**, see Figure 1.
-t *numtapes*	Specifies the number of tapes to load.	If you do not set this option, the default value is `1`. To use **ipload**, see Figure 1.

- The onpload -i option
  You can specify the number of records to process before **onpload** reports the progress in an entry in the log file with the **-i** option.

The onpload -i option

You can specify the number of records to process before **onpload** reports the progress in an entry in the log file with the **-i** option.

The **onpload** utility calculates the progress message count in the **pload** log file as equal to the number of rows processed, but rounded down to the nearest multiple of the value of *progress_interval*. For example, if the number of rows processed is 910 and the value of *progress_interval* is 100, then the progress message count is 900.

The **onpload** utility updates the row count only once for each stream buffer of data that it processes. Thus, reducing the row count on the **-i** option does not necessarily increase the number of progress messages in the log file. For example, if the stream buffer holds 910 rows of data, setting *row_count* to `10`, `100`, and `900` has the same effect: **onpload** writes one progress message.

Override the onpload database values

The options that are described in this topic let you enter size information that overrides existing parameters in the onpload database. You can override only these parameters. You cannot override other options, such as run mode.

```
Overriding the onpload database values

|--+------------------+-----------------------------------------|
   +- -C--caseconvert-+
   +- -D--override_db-+
   +- -F--filter------+
   +- -L--trace_level-+
   +- -M--converters--+
   +- -R--rejectfile--+
   +- -S--servername--+
   +- -T--target_db---+
   '- -l--logfile-----'
```

Element	Purpose
-C *caseconvert*	Sets the case-conversion option that converts all character information
-D *override_db*	Overrides the database specified in the map used for the load
-F *filter*	Identifies the filter that **onpload** uses for screening load records
-L *trace_level*	Sets the amount of information logged during the load
-M *converters*	Sets the maximum number of conversion threads per device
-R *rejectfile*	Identifies the file destination for rejected records
-S *servername*	Sets the onpload database server
-T *target_db*	Sets the target database serve
-l *logfile*	Specifies the name of a file to which **onpload** sends messages

Load data into collection data type columns

The **onpload** utility might need to create temporary smart large objects while loading data into collection data type columns. To create temporary smart large objects, **onpload** obtains sbspace information from the SBSPACETEMP configuration parameter, or if that parameter is not set, from the SBSPACENAME configuration parameter. For more information about temporary sbspaces, see the *IBM® Informix® Administrator's Guide*.

If **onpload** does not find a temporary sbspace, the load job fails with the following error message in the **onpload** log file:

```
Fatal error in server row processing - SQL error -9810 ISAM error -12053
```

If you see the error message shown previously in **onpload** log file, configure a temporary sbspace by using the SBSPACETEMP configuration parameter or a default sbspace by using the SBSPACENAME configuration parameter and then restart the load job.

The onpladm utility

This section describes how to use the **onpladm** utility.

- Overview of the onpladm utility
  The **onpladm** command-line interface is equivalent to the **ipload** utility. You can use the **onpladm** utility from the command line to create, modify, and delete High-Performance Loader (HPL) objects. The HPL objects include projects, jobs, maps, formats, queries, filters, device arrays, and machines.
- Define onpladm utility jobs
  You can create, modify, describe, list, run, and delete jobs with the **onpladm** utility.
- Define device arrays
  You can create, modify, describe, list, and delete device arrays with the **onpladm** utility.
- Define maps
  You can create, modify, describe, list, and delete maps with the **onpladm** utility.
- Define formats
  You can create, modify, describe, list, and delete formats with the **onpladm** utility.
- Define queries
  You can create, modify, describe, list, and delete queries with the **onpladm** utility.
- Define filters
  You can create, modify, describe, list, and delete filters with the **onpladm** utility.
- Define projects
  You can create a project, run all jobs in a project, list all projects, and delete a project with the **onpladm** utility.
- Define machine types
  You can create, modify, describe, list, and delete machine types with the **onpladm** utility.
- Define database operations

Related concepts:
The onpladm utility
The onpload file name size limitations on UNIX
Related reference:
Examples of loading and unloading jobs using the ipload utility
Start the ipload utility
The ipload utility GUI or the onpladm command-line interface

Overview of the onpladm utility

The **onpladm** command-line interface is equivalent to the **ipload** utility. You can use the **onpladm** utility from the command line to create, modify, and delete High-Performance Loader (HPL) objects. The HPL objects include projects, jobs, maps, formats, queries, filters, device arrays, and machines.

You can use the **onpladm** utility on both UNIX and Windows computers.

While the **onpload** and **onpladm** utilities include support for object names that contain up to 128 characters, the **ipload** utility does not. If you use long database, table or column names and create jobs by using **onpladm**, you cannot run these jobs with the **ipload** utility. For **ipload**, database, table and column names cannot exceed 18 characters.

- The onpladm utility features
  You can create and maintain the onpload database with the **onpladm** utility and the **ipload** utility. The onpload database stores information about the High-Performance Loader (HPL) objects of load and unload jobs.
- Specification-file conventions
  Use specification files to create, modify, and describe the HPL objects. When you enter the **onpladm** command to describe an object, the contents of the specification file appear.
- Error handling

Related information:
Cannot create shared-memory pool: errno UNIX_error_num

The onpladm utility features

You can create and maintain the onpload database with the **onpladm** utility and the **ipload** utility. The onpload database stores information about the High-Performance Loader (HPL) objects of load and unload jobs.

The first time that you issue a **create** command, the **onpladm** utility creates the onpload database, if it does not yet exist.

The **onpladm** utility can perform the functions that the following table describes.

Object	Operation
Project	Create, delete, describe, list, run
Job	Create, delete, describe, list, modify, run
Map	Create, delete, describe, list, modify
Format	Create, delete, describe, list, modify
Query	Create, delete, describe, list, modify
Device array	Create, delete, describe, list, modify
Computer	Create, delete, describe, list, modify
Target server	Configure and list default settings
Filter	Create, delete, describe, list, modify

The **onpladm** utility also allows you to:

- Create jobs and other objects with a single, minimum-input command.
- Specify object characteristics in specification files.
- Create, load, or unload jobs for all tables in a database.
- Create jobs for an entire database and group the jobs into a project with a single command.
- Run individual jobs or projects independently.

Important: The most effective way to use **onpladm** is to create and modify HPL objects is to:

1. Create the default objects by using default job creation options.
2. Use `describe object` syntax to describe the objects.
3. Manually modify these objects with correct values for different configuration parameters.
4. Use `modify object` syntax to correctly create these objects.

Specification-file conventions

Use specification files to create, modify, and describe the HPL objects. When you enter the **onpladm** command to describe an object, the contents of the specification file appear.

When you create a job or map with a quick command, the **onpladm** utility uses default attributes to create the job or map. If you create a job or map with a specification file, you can specify attribute values.

The following diagram illustrates the syntax to create or modify an object by using a specification file.

```
Creating or modifying an object with a specification file

>>-onpladm--+-create-+--object--+-------------------+----------->
            '-modify-'          '- -F -specfilename-'

>--+-----------------+-----------------------------------------><
   '- -S -servername-'
```

Element	Purpose	Key considerations
-F *specfilename*	Sets the specification file	The default value is the standard output.
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Use the following conventions when you create specification files:

- Begin object definitions with BEGIN OBJECT and end them with END OBJECT.
- If object definitions contain variable items, begin each variable item with BEGIN SEQUENCE and end each item with END SEQUENCE.
  For example, you might use the following specification file to create a device array that consists of a file and a pipe:

```
BEGIN OBJECT DEVICEARRAY mydevice
# Optional Attributes
BEGIN SEQUENCE
TYPE FILE
FILE /work/data.unl
TAPEBLOCKSIZE 0
TAPEDEVICESIZE 0
PIPECOMMAND
END SEQUENCE
BEGIN SEQUENCE
TYPE PIPE
FILE
TAPEBLOCKSIZE 0
TAPEDEVICESIZE 0
PIPECOMMAND /work/bin/datacreate.sh
END SEQUENCE
END OBJECT
```

  For more information about attributes and their possible values, see the description of each specification file.

- Precede comments in specification files with a pound sign (#).
- List attributes in the exact order in which the specification-file format displays them.

- Use the following syntax to refer to the attributes of an object or the attributes of elements of an object:
  Attribute_name Attribute_value

  Important: Do not use this for *BEGIN* and *END* statements and comment statements.
  You must always provide an attribute name. You must provide both the attribute name and the attribute value to describe a required attribute, but you only have to provide the attribute name if the attribute is optional.

  Attributes and their values depend on their object type. For more information about attributes and object types, see the corresponding specification files.

- Enclose attribute values that contain spaces in double quotation marks.
- Precede double quotation marks in attribute values with a double quotation mark.
  For example, to enter a MATCH condition for "CA" in a filter object, include the following line:`MATCH =""CA""`

For more information about file conventions, see individual specification-file formats.

Related reference:
Create maps

Error handling

The **onpladm** commands have two return values:

0
: If the command is successful, a success message appears.

-1
: If the command fails, an error message appears.

Define onpladm utility jobs

You can create, modify, describe, list, run, and delete jobs with the **onpladm** utility.

- Create onpladm jobs
- Modify a job by using a detailed specification file
- Describe a job
- List all jobs in a project
- Run a job
- Delete a job

Related reference:
Components of the load job

Create onpladm jobs

You can create two types of jobs:

- Conversion jobs
  Use a conversion job to process data that requires conversion before loading and that is not in internal format. Conversion jobs have associated maps and format objects.

- No-conversion jobs (fast jobs)
  Use a no-conversion job to process data that does not require conversion before loading and that uses the IBM® Informix® internal format. No maps or formats are associated with a no-conversion job. Consequently, a no-conversion job is faster than a conversion job. No-conversion jobs are always completed in express mode.

- Create conversion jobs
- Create no-conversion jobs
  A no-conversion job is faster than a conversion job because **onpload** uses the internal format of the target database and does not create maps or formats. No-conversion jobs are always run in express mode.

Create conversion jobs

When you use command-line parameters to create a conversion load or unload job, all maps, formats, and query objects automatically have the same name as the job name. If you delete a job, the **map.format** or the query objects will remain intact unless you specifically delete the format or query objects.

When you create a conversion job with detailed specification files, the High-Performance Loader (HPL) objects can have different names from the job name.

- Create conversion jobs by using a quick command
  When you create a conversion job by using a quick command, the **onpladm** utility creates all High-Performance Loader (HPL) objects associated with the job. The HPL objects that it creates have the same name as the job.
- Create conversion jobs by using detailed specification files
  You can also specify job details in specification files that you reference from the command line. When you create a job by using a specification file, you must create all associated High-Performance Loader (HPL) objects; the **onpladm** utility does not create these objects for you.

Create conversion jobs by using a quick command

When you create a conversion job by using a quick command, the **onpladm** utility creates all High-Performance Loader (HPL) objects associated with the job. The HPL objects that it creates have the same name as the job.

The following diagram illustrates the syntax to create a conversion job from the command line.

```
Creating a conversion job

>>-onpladm create job -job--+---------------+-- -d--device ------>
                           '- -p--project -'

>-- -D--database-- -t--table--+-------------------------+------>
                              '-| Setting the run mode |-'

>--+-------------------------+--+-------------+------------------>
   +- -n -------------------+  '- -T--target-'
   '-| Setting the format |-'

>--+-------------+--+-----------------+--+----------------+----><
   '- -S--server-'  '- -M--devicesize-'  '- -B--blocksize-'
```

Element	Purpose	Key considerations
-B *blocksize*	Sets the tape I/O block size (bytes)	No default value
-d *device*	Sets the name of device, such as a file, device array, tape, or pipe	No default value
-D *database*	Name of the target database that contains the information to be loaded or unloaded	No default value.
job	Names a load or unload job from the **onpload** database	None
-M *devicesize*	Tape device size in kilobytes	The device size must be greater than zero.
-n	Sets no-conversion express job	None
-p *project*	Identifies the project where the format and map are stored	The default is the project created when the onpload database is built.
-S *server*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.
-t *table*	Name of the table to be loaded or unloaded	None
-T *target*	Name of the target server to which the data will download	The default is the value of the INFORMIXSERVER environment variable.

The following diagram illustrates the syntax to set the run mode with the **-f** option.

```
Setting the Run Mode

              .-l--+---+-.
              |    '-c-' |
|-- -f--+---+--+----------+--+---+--+--+---+-----------------------|
        +-d-+  '-u--------'  '-n-'  '-N-'
        +-p-+
        '-a-'
```

Element	Purpose	Key considerations
a	Treats data source as a device-array	The definition of the device array is extracted from the **onpload** database.
c	Sets mode to deluxe mode	If this flag is not set, **onpladm** uses express mode.
d	Treats data source as a tape device	None
l	Loads data into database	This is the default flag, as opposed to **u**, which unloads data from the database.
n	Specifies that **onpladm** does not need to perform data conversion	The target table for the load must have the same schema as the table from which the data is extracted. If **onpladm** generated the input data file as a data file in IBM® Informix® format, you do not need to perform data conversion when you reload data.
N	Allows deluxe mode load without replication	This flag works only if the *job* argument, which specifies a session table job to run, is left blank.
p	Treats data source as a program to run and reads interface to the program by way of a pipe (on UNIX only)	None
u	Unloads data from database	If this flag is not set, **onpladm** loads data into the database.

The following diagram illustrates the syntax to set the format type with the **-z** option.

```
Setting the Format

|-- -z--+----+-----------------------------------------------|
        +-D--+
        +-FI-+
        +-FA-+
        +-FB-+
        +-C--+
        '-CB-'
```

Element	Purpose	Key considerations
D	Sets the format to delimited	This is the default value. See Delimited records.
FI	Sets the format to fixed internal	See Fixed-length records.
FA	Sets the format to fixed ASCII	See Fixed-length records.
FB	Sets the format to fixed binary format.	See Fixed-length records.
C	Sets the format to COBOL	See COBOL records.
CB	Sets the format to COBOL (byte)	See COBOL records.

Create conversion jobs by using detailed specification files

You can also specify job details in specification files that you reference from the command line. When you create a job by using a specification file, you must create all associated High-Performance Loader (HPL) objects; the **onpladm** utility does not create these objects for you.

- Create a conversion-load job
- Create a conversion unload job

Related reference:
Describe a job

Create a conversion-load job

Use the syntax shown in Specification-file conventions to create a conversion-load job from a specification file.

Use the following syntax to create a specification file for a conversion-load job:

```
BEGIN OBJECT LOADJOB jobname
# Compulsory Attributes
PROJECT projectname
DEVICE device_array_name
MAP mapname

FILTER filtername
SERVER targetservername
DATABASE targetdatabasename
FLTFILE filtered_records_filename
REJECTFILE rejected_records_filename
LOGFILE job_progress_logfilename
RUNMODE runmode_type
GENERATEVIORECS violation_records_option
TAPES sourcetape_num
NUMRECORDS records_num
STARTRECORD start_record
MAXERRORS max_error_num

END OBJECT
```

The following table lists the attributes and their attribute values.

Attribute	Attribute value
device_array_name	Device-array name You must create the device array; **onpladm** does not create it for you.
filtername	Filter name
jobname	Job name
job_progress_logfilename	Path to the job-progress log file
mapname	Map name You must create the map before you use this option; **onpladm** does not create the map for you.
max_error_num	Maximum number of errors; if exceeded, load ends
projectname	Name of an existing project
records_num	Number of records to be processed in the data file

Attribute	Attribute value
rejected_records_filename	Rejected-records file name (rejected by database server)
filtered_records_filename	Filtered-records file name (rejected by filter)
runmode_type	Type of run mode: E Express® mode D Deluxe mode without replication DR Deluxe mode with replication
sourcetape_num	Number of tapes that contain source data
start_record	Number of the first record to begin a load
targetdatabasename	Name of the database that the records will be loaded or unloaded to; if set, this value overrides the database value in the load or unload map
targetservername	Name of the target database server
violation_records_option	Specify: **Y** to generate violations records or **N** not to generate violations records

Create a conversion unload job

Use the syntax shown in Specification-file conventions to create a conversion unload job from a specification file.

Use the following syntax to create a conversion unload job:

```
BEGIN OBJECT UNLOADJOB jobname
# Compulsory Attributes
PROJECT projectname
DEVICE device_array_name
MAP mapname

FILTER filtername
SERVER targetservername
DATABASE targetdatabasename
REJECTFILE rejected_records_filename
LOGFILE job_progress_logfilename
ISOLATIONLEVEL isolation_level
MAXERRORS max_error_num

END OBJECT
```

The following table lists the arguments and their attribute values.

Attribute	Attribute value
device_array_name	Device-array name You must create the device array; **onpladm** does not create it for you.
filtername	Filter name
isolation_level	Unload isolation level: DR Dirty Read CR Committed Read CS Cursor Stability RR Repeatable Read
jobname	Job name
job_progress_logfilename	Path to the job-progress log file
mapname	Map name You must create the map before you use this option; **onpladm** does not create the map for you.
max_error_num	Maximum number of errors; if exceeded, unload ends
projectname	Name of an existing project
rejected_records_filename	Rejected-records file name (rejected by database server)
targetdatabasename	Name of the database that the records will be loaded or unloaded to; if set, this value overrides the database value in the load or unload map
targetservername	Name of the target database server

Create no-conversion jobs

A no-conversion job is faster than a conversion job because **onpload** uses the internal format of the target database and does not create maps or formats. No-conversion jobs are always run in express mode.

- Create no-conversion jobs by using a quick command
  When you create a job by using a quick command, the **onpladm** utility creates all High-Performance Loader (HPL) objects associated with that job.
- Create no-conversion jobs by using detailed specification files
  You can also specify job details in specification files that you reference from the command line. When you create a job by using a specification file, you must create all associated High-Performance Loader (HPL) objects; the **onpladm** utility does not create these objects for you.

Create no-conversion jobs by using a quick command

When you create a job by using a quick command, the **onpladm** utility creates all High-Performance Loader (HPL) objects associated with that job.

The following diagram illustrates the syntax to create a no-conversion job.

```
Creating a no-conversion job

>>-onpladm create job -job--+----------------+-- -n-------------->
                           '- -p--project -'

>-- -d--device -- -D--database-- -t--table---------------------->

   .-------------------------------------------------------------.
   V                                                             |
>----+-------------------------------------------------------+-+-><
     '-+-------------+--+-------------+--| Setting the run mode |-'
       '- -S--server-'  '- -T--target-'
```

Element	Purpose	Key considerations
-d *device*	Sets the name of device, such as a file, device array, tape, or pipe	No default value.
-D *database*	Name of the target database that contains the information to be loaded or unloaded	No default value.
job	Names a load or unload job from the onpload database	None
-n	Sets no-conversion express job	None
-p *project*	Identifies the project where the format and map are stored	The default is the project created when the onpload database is built.
-S *server*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.
-t *table*	Name of the table to be loaded or unloaded	None
-T *target*	Name of the target server to which the data will download	The default is the value of the INFORMIXSERVER environment variable.

The following diagram illustrates the syntax to set the run mode with the **-f** option.

```
Setting the run mode

              .-l-.
|-- -f--+---+--+---+-----------------------------------------|
        +-d-+  '-u-'
        +-p-+
        '-a-'
```

Element	Purpose	Key considerations
a	Treats the data source as a device-array	The definition of the device array is extracted from the **onpload** database.
d	Treats the data source as a tape device	None
l	Loads data into the database	This is the default flag, as opposed to **u**, which unloads data from the database.
p	Treats the data source as a program to run, and reads the interface to the program by way of a pipe (UNIX only)	None
u	Unloads data from the database	If this flag is not set, **onpladm** loads data into the database.

Create no-conversion jobs by using detailed specification files

You can also specify job details in specification files that you reference from the command line. When you create a job by using a specification file, you must create all associated High-Performance Loader (HPL) objects; the **onpladm** utility does not create these objects for you.

Create a no-conversion load job

Use the syntax shown in Specification-file conventions to create a no-conversion load job with specification files.

Use the following syntax to create a specification file for a no-conversion load job:

```
BEGIN OBJECT FASTLOADJOB jobname
# Compulsory Attributes
PROJECT projectname
DEVICE device_array_name
DATABASE targetdatabasename
TABLE tablename

SERVER targetservername

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
device_array_name	Device-array name You must create the device array; **onpladm** does not create it for you.
jobname	Job name
projectname	Name of an existing project
tablename	Table name
targetdatabasename	Name of the database that the records will be loaded or unloaded to; if set, this value overrides the database value in the load map
targetservername	Name of the target database server

Create a no-conversion unload job

Use the syntax shown in Specification-file conventions to create a no-conversion unload job with specification files.

Use the following syntax to create a specification file for a no-conversion unload job:

```
BEGIN OBJECT FASTUNLOADJOB jobname
# Compulsory Attributes
PROJECT projectname
DEVICE device_array_name
DATABASE targetdatabasename
QUERY queryname

SERVER targetservername

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
device_array_name	Device-array name You must create the device array; **onpladm** does not create it for you.
jobname	Job name
projectname	Name of an existing project
queryname	The SQL SELECT statement, in quotation marks, that contains unload criteria
targetdatabasename	Name of the database that the records will be loaded or unloaded to; if set, this value overrides the database value in the load map
targetservername	Name of the target database server

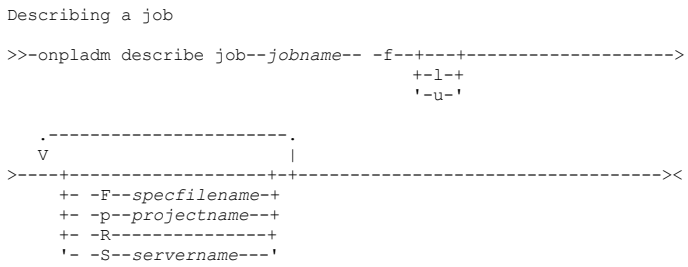
Modify a job by using a detailed specification file

Use the syntax shown in Specification-file conventions to modify a job by using a specification file.

For information about the job-specification file, see Create no-conversion jobs by using detailed specification files.

Describe a job

The following diagram illustrates the syntax to describe a job.

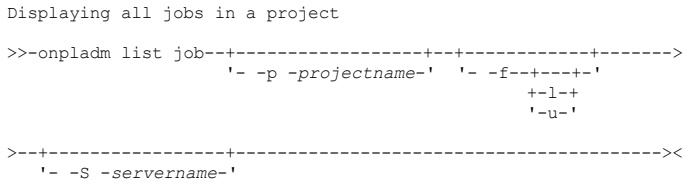


Element	Purpose	Key considerations
-F *specfilename*	Sets the specification file	The default value is the standard output.
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
jobname	Names a job from the onpload database	None
-p *projectname*	Identifies the project where the format and map are stored	None
-R	Deletes all associated objects if they are not referenced by other objects	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Create conversion jobs by using detailed specification files

List all jobs in a project

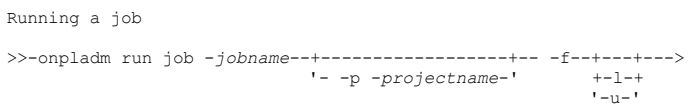
The following diagram illustrates the syntax to display all the jobs in a project.



Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Run a job

The following diagram illustrates the syntax to run a job and display the job progress or send it to a log file.



Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
jobname	Names a load or unload job from the onpload database	None
-l *logname*	Sets the path for a log file where job progress is recorded	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.
-Z	Enables writing to or reading from a tape until the end of the device. The **onpladm** utility prompts you for additional tapes until the load or unload is completed.	If you use this feature, you must use it for both load and unload jobs. Otherwise, the unloaded data might not match the loaded data. If the **-Z** option is not set, the **onpladm** uses the tape size specified with the **-M** option of the **create job** command. If the **-Z** option is set, it supersedes the tape size information provided.

- The onpladm utility when referential constraints are on tables

Related tasks:
Running onpladm on UNIX with the database server running on Windows

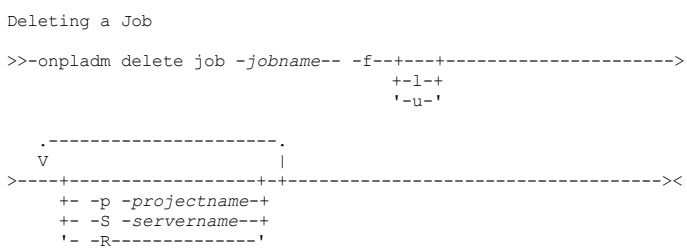
The onpladm utility when referential constraints are on tables

The **onpladm** utility unloads and loads an entire database by creating individual table unload and load jobs. If the load mode is **express**, the default load mode, the **onpladm** utility disables referential constraints during load jobs. If database tables have constraints on them, problems can occur. For example, if one table has a referential constraint to another table and that table is not yet loaded, a violation that prevents a table from loading can occur.

If tables have constraints, you can manually disable the constraints after the load job is complete.

Delete a job

The following diagram illustrates the syntax to delete a job.



Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
jobname	Names a load or unload job from the **onpload** database	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.
-R	Deletes all associated objects if they are not referenced by other objects	None

Define device arrays

You can create, modify, describe, list, and delete device arrays with the **onpladm** utility.

- Create a device array
  You can only create a device array by using a detailed specification file.
- Modify a device array
  You can only modify a device array with a specification file.
- Describe a device array
- List project device arrays
- Deleting a device array

Related concepts:
Device arrays
Devices for the device array

Create a device array

You can only create a device array by using a detailed specification file.

Use the syntax shown in Specification-file conventions to create a device array.

Use the following syntax to create a device array:

```
BEGIN OBJECT DEVICEARRAY device_array_name
# Compulsory Attributes
BEGIN SEQUENCE
TYPE device_type
FILE device_path
TAPEBLOCKSIZE tapeblock_size
TAPEDEVICESIZE tapedevice_size
PIPECOMMAND pipe_commandname
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
device_array_name	Device-array name You must create the device array; **onpladm** does not create it for you.
device_path	Path to the device; only valid if device type is file or tape
device_type	Type of device, in uppercase letters (for instance, `PIPE`, `FILE`, or `TAPE`)
tapeblock_size	Tape-block size
tapedevice_size	Tape-device size in megabytes
pipe_commandname	Pipe command name

Related reference:
Modify a device array
Describe a device array

Modify a device array

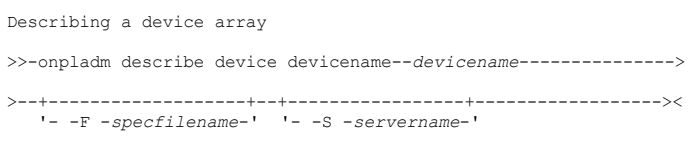
You can only modify a device array with a specification file.

Use the syntax shown in Specification-file conventions to modify a device array.

Related reference:
Create a device array

Describe a device array

The following diagram illustrates the syntax to describe a device array.



Element	Purpose	Key considerations
devicename	Sets the name of device, such as a file, device array, tape, or pipe	None
-F *specfilename*	Sets the specification file	The default value is the standard output.
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Create a device array

List project device arrays

To list all the device arrays in a project, use the following syntax.

```
Listing project device arrays

>>-onpladm list device--+------------------+-------------------->
                       '- -p -projectname-'

>--+-----------------+-----------------------------------------><
   '- -S -servername-'
```

Element	Purpose	Key considerations
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Deleting a device array

The following diagram illustrates the syntax to delete a device array.

```
Deleting project device arrays

>>-onpladm delete device -devicename--+------------------+------><
                                     '- -S -servername-'
```

Element	Purpose	Key considerations
devicename	Sets the name of device, such as a file, device array, tape, or pipe	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Define maps

You can create, modify, describe, list, and delete maps with the **onpladm** utility.

- Create maps
  You can create maps by using a quick command or a specification file.
- Delete a map
- Describe a map
- Modify a map by using a detailed specification file
- List all maps in a project

Related concepts:
Load and unload maps

Create maps

You can create maps by using a quick command or a specification file.

- Create a map by using a quick command
  When you create a map by using a quick command, the **onpladm** utility creates a format object with the same name as the map, plus the suffix **-fmt**. The generated format name (as for all **onpladm** objects) has a maximum length of 18 characters.

- Create a map with a detailed specification file
  When you create a map, the **onpladm** utility creates a format object with the same name as the map. When you create a map with a specification file, you must create all associated High-Performance Loader (HPL) objects; the **onpladm** utility does not create these objects for you.

Related reference:
Specification-file conventions

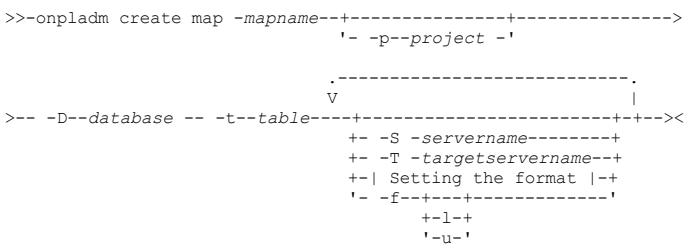
Create a map by using a quick command

When you create a map by using a quick command, the **onpladm** utility creates a format object with the same name as the map, plus the suffix **-fmt**. The generated format name (as for all **onpladm** objects) has a maximum length of 18 characters.

For example, if the map name is **mymap**, the format name is **mymap-fmt**. If the map name is **123456789123456789**, the format name is **12345678912345-fmt**.

The **create map** command also creates a query object for the unload map. The following diagram illustrates the syntax to create a map from the command line.

Creating a map



Element	Purpose	Key considerations
-D *database*	Name of the target database that contains the information to be loaded or unloaded	No default value
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
mapname	Sets the map	None
-p *project*	Identifies the project where the format and map are stored	The default is the project created when the onpload database is built.
-S *server*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.
-t *table*	Name of the table to be loaded or unloaded	None
-T *target*	Name of the target server to which the data will download	The default is the value of the INFORMIXSERVER environment variable.

The following diagram illustrates the syntax to set the format type with the **-z** option.

Setting the format

Element	Purpose	Key considerations
D	Sets the format to delimited	This is the default value. See Delimited records.
FI	Sets the format to fixed internal	See Fixed-length records.
FA	Sets the format to fixed ASCII	See Fixed-length records.
FB	Sets the format to fixed binary	See Fixed-length records.
C	Sets the format to COBOL	See COBOL records.
CB	Sets the format to COBOL (byte)	See COBOL records.

Create a map with a detailed specification file

When you create a map, the **onpladm** utility creates a format object with the same name as the map. When you create a map with a specification file, you must create all associated High-Performance Loader (HPL) objects; the **onpladm** utility does not create these objects for you.

Use the syntax shown in Specification-file conventions to create maps with specification files.

Use the following syntax to creating a load map with a specification file:

```
BEGIN OBJECT LOADMAP mapname

# Compulsory Attributes
PROJECT projectname
FORMAT formatname
DATABASE targetdatabasename
TABLE targettablename

BEGIN SEQUENCE
COLUMNNAME columnname
FIELDNAME fieldname
JUSTIFICATION justification
CASECONVERT caseconversion
DEFAULTVALUE defaultvalue
TRANSFERBYTES byte_transfer
COLUMNOFFSET column_offset
FIELDOFFSET field_offset
FIELDMINIMUM field_minimum
FIELDMAXIMUM field_maximum
FILLCHARACTER fillcharacter
PICTURE picture
FUNCTION record_function
STORAGECODING storage_format
BLOBCOLUMN blob_columnname
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
blob_columnname	The column that contains the name of the file where BYTE or TEXT data is stored See Simple LO data in a separate file.
byte_transfer	Number of bytes to transfer from record field to database column
caseconversion	Enter UPPER for all uppercase data, LOWER for all lowercase data, and PROPER for data with an initial capital letter
columnname	Name of the column to be mapped
column_offset	Amount of offset from the beginning of the column to the location on the column from which data transfer begins
defaultvalue	Value when no field is mapped to the column
fieldname	Field corresponding to the record format to be mapped
field_maximum	Largest acceptable numeric-column value
field_minimum	Smallest acceptable numeric-column value
field_offset	Amount of offset from the start of the field record to the location in the record from which data transfer begins
fillcharacter	Character used to pad contents of a field
formatname	Associated format name You must create the format; **onpladm** does not create it for you.
justification	Enter left, right, or CENTER to position text within a record
mapname	Map name
picture	Reformats and masks data from the field of a record before data is transferred to database
projectname	Name of existing project
record_function	User-defined function in a shared library that is called for every record that is processed See Custom-conversion functions.
storage_format	The format in which to store BYTE or TEXT data
targetdatabasename	Name of the database that the records will be loaded and unloaded to
targettablename	Target-table name

Use the following syntax to create an unload map with a specification file:

```
BEGIN OBJECT UNLOADMAP mapname

# Compulsory Attributes
PROJECT projectname
FORMAT formatname
DATABASE targetdatabasename
QUERY queryname

BEGIN SEQUENCE
```

```
COLUMNNAME columnname
FIELDNAME fieldname
JUSTIFICATION justification
CASECONVERT caseconversion
DEFAULTVALUE defaultvalue
TRANSFERBYTES byte_transfer
COLUMNOFFSET column_offset
FIELDOFFSET field_offset
FIELDMINIMUM field_minimum
FIELDMAXIMUM field_maximum
FILLCHARACTER fillcharacter
PICTURE picture
FUNCTION record_function
STORAGECODING storage_format
BLOBCOLUMN blob_columnname
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
blob_columnname	The column that contains the name of the file where BYTE or TEXT data is stored See Simple LO data in a separate file.
byte_transfer	Number of bytes to transfer from record field to database column
caseconversion	Enter UPPER for all uppercase data, LOWER for all lowercase data, and PROPER for data with an initial capital letter
columnname	Name of the column to be mapped
column_offset	Amount of offset from the beginning of the column to the location on the column from which data transfer begins
defaultvalue	Value when no field is mapped to the column
fieldname	Field corresponding to the record format to be mapped
field_maximum	Largest acceptable numeric-column value
field_minimum	Smallest acceptable numeric-column value
field_offset	Amount of offset from the start of the field record to the location in the record from which data transfer begins
fillcharacter	Character used to pad contents of a field
formatname	Associated format name
justification	Enter left, right, or CENTER to position text within a record
mapname	Map name
picture	Reformats and masks data from the field of a record before data is transferred to database
projectname	Name of existing project
queryname	Query name
record_function	User-defined function in dynamically linked library that is called for every record that is processed See Custom-conversion functions.
storage_format	The format in which to store BYTE or TEXT data
targetdatabasename	Name of the database that the records will be loaded and unloaded to

Related reference:
Describe a map
Modify a map by using a detailed specification file

Delete a map

The following diagram illustrates the syntax to delete a map.

```
Deleting a map

>>-onpladm delete map -mapname-- -f--+---+---------------------->
                                     +-l-+
                                     '-u-'

   .----------------------.
   V                      |
>----+------------------+-+----------------------------------><
     +- -p -projectname-+
     '- -S -servername--'
```

Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None

Element	Purpose	Key considerations
mapname	Sets the map name	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Describe a map

The following diagram illustrates the syntax to describe a map.

```
Describing a map

>>-onpladm describe map -mapname-- -f--+---+-------------------->
                                       +-l-+
                                       '-u-'

   .-----------------------.
   V                       |
>----+-------------------+-+---------------------------------><
     +- -p -projectname--+
     +- -F -specfilename-+
     '- -S -servername---'
```

Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
-F *specfilename*	Sets the specification file	The default value is the standard output.
mapname	Sets the map name	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Create a map with a detailed specification file

Modify a map by using a detailed specification file

Use the syntax shown in Specification-file conventions to modify an existing map with a specification file.

Related reference:
Create a map with a detailed specification file

List all maps in a project

The following diagram illustrates the syntax to list all the maps in a project.

```
Listing all maps in a project

                                    .-----------------------.
                                    V                       |
>>-onpladm list map--+------------+----+-------------------+-+--><
                     '- -f--+---+-'    +- -p -projectname-+
                            +-l-+      '- -S -servername--'
                            '-u-'
```

Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.
l	Specifies a load job	None
u	Specifies an unload job	None
-p *projectname*	Identifies the project where the format and map are stored	None

Element	Purpose	Key considerations
-S *servername*	Sets the onload database server	The default is the value of the INFORMIXSERVER environment variable.

Define formats

You can create, modify, describe, list, and delete formats with the **onpladm** utility.

- Create a format
  You can only create formats by using a detailed specification file.
- Modify a format by using a specification file
  You can only modify a format by using a specification file.
- Describe a format
- List all formats in a project
- Delete a format

Related concepts:
Formats of supported datafile records

Create a format

You can only create formats by using a detailed specification file.

Use the syntax shown in Specification-file conventions to create a format by using a detailed specification file.

Use the following syntax to create a fixed-format object from a specification file:

```
BEGIN OBJECT FIXEDFORMAT formatname

# Compulsory Attributes
PROJECT projectname
CHARACTERSET data_codeset
MACHINE machine_type
BEGIN SEQUENCE
FIELDNAME fieldname
DATATYPE datatype
BYTES field_bytes
DECIMALS decimal_places
OFFSET offset_bytes
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
data_codeset	Code set used to translate data in the data table For more information about data code sets, see the following file on your product CD: $INFORMIXDIR/gls/cm3/registry
datatype	Type of field data See Data types allowed in a fixed format.
decimal_places	Number of decimal places for Float and Double data types
field_bytes	Number of bytes that the field occupies in the record
fieldname	Field name in the format
formatname	Format name
machine_type	Type of computer that produced the data, such as a SPARCstation See List all existing machine types.
offset_bytes	Number of bytes of the field offset in the record
projectname	Name of existing project

Use the following syntax to create a COBOL-format object from a specification file:

```
BEGIN OBJECT COBOLFORMAT formatname

# Compulsory Attributes
PROJECT projectname
CHARACTERSET data_codeset
MACHINE machine_type
DRIVER driver_type
BEGIN SEQUENCE
FIELDNAME fieldname
PICTURE picture_description
USAGE usage_description
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
data_codeset	Code set used to translate data in the data table For more information about data code sets, see the following file on your product CD: $INFORMIXDIR/gls/cm3/registry
driver_type	Type of driver: COBOL (default) or COBOL_b
fieldname	Field name in the format
formatname	Format name
machine_type	Type of computer that produced the data, such as a SPARCstation
picture_description	Picture description that matches the record FD from the COBOL program See Picture strings.
projectname	Name of existing project
usage_description	Number of bytes that the field occupies in the record

Use the following syntax to create a delimited-format object:

```
BEGIN OBJECT DELIMITEDFORMAT formatname

# Compulsory Attributes
PROJECT projectname
CHARACTERSET data_codeset
RECORDSTART recordstart_delimit_character
RECORDEND recordend_delimit_character
FIELDSTART fieldstart_delimit_character
FIELDEND fieldend_delimit_character
FIELDSEPARATOR fieldseparator_delimit_character
BEGIN SEQUENCE
FIELDNAME format_fieldname
FIELDTYPE field_datatype
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
data_codeset	Code set used to translate data in the data table For more information about data code sets, see the following file on your product CD: $INFORMIXDIR/gls/cm3/registry
field_datatype	Type of field data See Data types allowed in a delimited format.
fieldend_delimit_character	Delimiting character that specifies the end of the field, in hexadecimal or decimal format Begin a hexadecimal character with `0x`.
fieldstart_delimit_character	Delimiting character that specifies the start of the field, in hexadecimal or decimal format Begin a hexadecimal character with `0x`.
fieldseparator_delimit_character	Delimiting character that specifies the field separator, in hexadecimal or decimal format Begin a hexadecimal character with `0x`.
format_fieldname	Format field name
formatname	Format name
recordend_delimit_character	Delimiting character that specifies the end of the record, in hexadecimal or decimal format Begin a hexadecimal character with `0x`.
recordstart_delimit_character	Delimiting character that specifies the start of the record, in hexadecimal or decimal format Begin a hexadecimal character with `0x`.
projectname	Name of existing project

Related reference:
Modify a format by using a specification file
Describe a format

Modify a format by using a specification file

You can only modify a format by using a specification file.

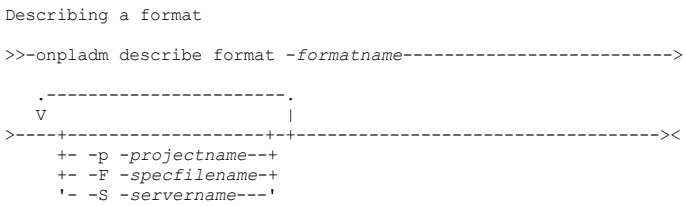
Use the syntax shown in Specification-file conventions to modify a format.

Related reference:
Create a format

Describe a format

The following diagram illustrates the syntax to describe a format to a file.

Describing a format



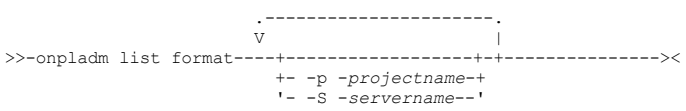
Element	Purpose	Key considerations
-F *specfilename*	Sets the specification file	The default value is the standard output.
formatname	Sets the format name	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Create a format

List all formats in a project

The following diagram illustrates the syntax to list all formats in a project to standard output.

Listing all formats in a project

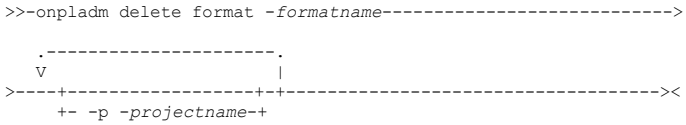


Element	Purpose	Key considerations
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Delete a format

The following diagram illustrates the syntax to delete a format.

Deleting all formats in a project



Element	Purpose	Key considerations
formatname	Sets the format name	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Define queries

You can create, modify, describe, list, and delete queries with the **onpladm** utility.

- Creating a query
  You can only create a query by using a detailed specification file.

- Modify a query
  You can only modify a query by using a detailed specification file.
- Describing a query
- List all queries in a project
- Delete a query

Creating a query

You can only create a query by using a detailed specification file.

Use the syntax shown in Specification-file conventions to create a query.

Use the following syntax to create a specification file for a query:

```
BEGIN OBJECT QUERY queryname
# Compulsory Attributes
PROJECT projectname
DATABASE targetdatabasename
SELECTSTATEMENT sql_statement

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
queryname	Query name
projectname	Name of an existing project
sql_statement	SQL SELECT statement, in quotation marks, that contains unload criteria
targetdatabasename	Name of the database that the records will be loaded or unloaded to

Related reference:
Modify a query
Describing a query

Modify a query

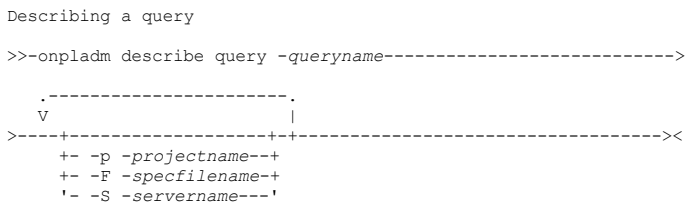
You can only modify a query by using a detailed specification file.

Use the syntax shown in Specification-file conventions to modify a query by using a specification file.

Related reference:
Creating a query

Describing a query

The following diagram illustrates the syntax to describe a query.



Element	Purpose	Key considerations
-F *specfilename*	Sets the specification file	The default value is the standard output.
queryname	Sets the query name	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Creating a query

List all queries in a project

The following diagram illustrates the syntax to list all queries in a project to standard output.

```
Listing all queries in a project

                      .----------------------.
                      V                      |
>>-onpladm list query----+------------------+-+----------------><
                         +- -p -projectname-+
                         '- -S -servername--'
```

Element	Purpose	Key considerations
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Delete a query

The following diagram illustrates the syntax to delete a query.

```
Deleting a query

                                   .----------------------.
                                   V                      |
>>-onpladm delete query -queryname----+------------------+-+--->><
                                      +- -p -projectname-+
                                      '- -S -servername--'
```

Element	Purpose	Key considerations
-p *projectname*	Identifies the project where the format and map are stored	None
queryname	Sets the name of the query	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Define filters

You can create, modify, describe, list, and delete filters with the **onpladm** utility.

- Create a filter
  You can only create a filter by using a specification file.
- Modify a filter
  You can only modify a filter by using a detailed specification file.
- Describe a filter
- List all filters in a project
- Delete a filter

Related reference:
Example of using filter

Create a filter

You can only create a filter by using a specification file.

Use the syntax shown in Specification-file conventions to create a filter by using a specification file.

Use the following syntax to create a filter by using a specification file:

```
BEGIN OBJECT FILTER filtername
# Compulsory Attributes
PROJECT projectname
FORMAT formatname
BEGIN SEQUENCE
FIELDNAME data_file_fieldname
STATUS record_status
MATCH match_criteria
END SEQUENCE

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
data_file_fieldname	Data-file field to be used in the match condition
formatname	Associated format name
filtername	Filter name
match_criteria	Match criteria, in quotation marks. See Match condition operators and characters.
projectname	Name of existing project
record_status	Type a `K` to keep records that meet a match condition or `D` to discard them.

Related reference:
Modify a filter
Describe a filter

Modify a filter

You can only modify a filter by using a detailed specification file.

Use the syntax shown in Specification-file conventions to modify a filter.

Related reference:
Create a filter

Describe a filter

The following diagram illustrates the syntax to describe a filter.

```
Describing a filter

>>-onpladm describe filter -filtername-------------------------->

   .-----------------------.
   V                       |
>----+---------------------+-+---------------------------------><
     +- -p -projectname--+
     +- -F -specfilename-+
     '- -S -servername---'
```

Element	Purpose	Key considerations
-F *specfilename*	Sets the specification file	The default value is the standard output.
filtername	Sets the filter name	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Create a filter

List all filters in a project

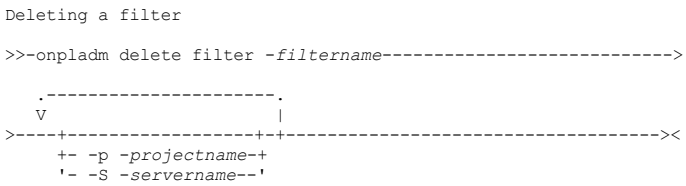
The following diagram illustrates the syntax to list all the filters in a project.

```
Listing all filters in a project

                        .-----------------------.
                        V                       |
>>-onpladm list filter----+---------------------+-+---------------><
                          +- -p -projectname-+
                          '- -S -servername--'
```

Element	Purpose	Key considerations
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Delete a filter

The following diagram illustrates the syntax to delete a filter.



Element	Purpose	Key considerations
filtername	Sets the name of the filter	None
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Define projects

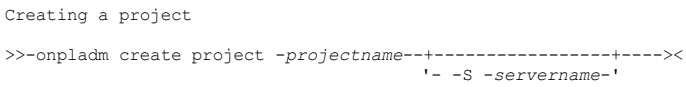
You can create a project, run all jobs in a project, list all projects, and delete a project with the **onpladm** utility.

- Create a project
- Run all jobs in a project
- List all projects
- Delete a project

Related concepts:
Project organization

Create a project

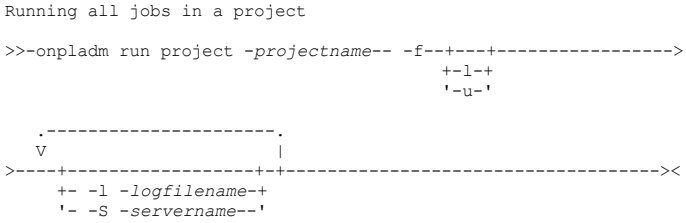
The following diagram illustrates the syntax to create a new, empty project to which you can add the HPL jobs.



Element	Purpose	Key considerations
-p *projectname*	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Run all jobs in a project

The following diagram illustrates the syntax to run all load or unload jobs in a project.



Element	Purpose	Key considerations
-f	Flags to specify the type of job	The default is load job.

Element	Purpose	Key considerations
l	Specifies a load job	None
u	Specifies an unload job	None
-l *logfilename*	Sets the path for a log file where job progress is recorded	None
projectname	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
The onpladm utility on Windows

List all projects

The following diagram illustrates the syntax to list all your projects.

```
Listing all projects

>>-onpladm list project--+-----------------+------------------><
                        '- -S -servername-'
```

Element	Purpose	Key considerations
-S *servername*	Sets the **onpload** database server	The default is the value of the INFORMIXSERVER environment variable.

Delete a project

The following diagram illustrates the syntax to delete a project, all of its corresponding jobs, and other HPL objects that it holds.

```
Deleting a project

>>-onpladm delete project -projectname--+-----------------+----><
                                        '- -S -servername-'
```

Element	Purpose	Key considerations
projectname	Identifies the project where the format and map are stored	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Define machine types

You can create, modify, describe, list, and delete machine types with the **onpladm** utility.

- Create a machine type
  You can only create a machine type by using a specification file.
- Modify a machine type
  You can only modify a machine type by using a detailed specification file.
- Describe a machine
- List all existing machine types
- Delete a machine type
  You can only delete a machine type by using a specification file.

Create a machine type

You can only create a machine type by using a specification file.

Use the syntax shown in Specification-file conventions to create a machine type.

Use the following syntax to create a machine object by using a specification file:

```
BEGIN OBJECT MACHINE machinename
# Compulsory Attributes
BYTEORDER machinetype_byteorder
SHORTSIZE shortinteger_bytes
```

```
INTEGERSIZE integer_bytes
LONGSIZE longinteger_bytes
FLOATSIZE float_bytes
DOUBLESIZE double_bytes

END OBJECT
```

The following table lists the attributes and their values.

Attribute	Attribute value
double_bytes	Double integer size
float_bytes	Float size
integer_bytes	Integer size
machinename	Machine name
longinteger_bytes	Long integer size
machinetype_byteorder	Computer byte-order type Enter `MSB` for most-significant bit or `LSB` for least-significant bit.
shortinteger_bytes	Short integer size

Related reference:
Modify a machine type
Describe a machine

Modify a machine type

You can only modify a machine type by using a detailed specification file.

Use the syntax shown in Specification-file conventions to modify a machine type.

Related reference:
Create a machine type

Describe a machine

The following diagram illustrates the syntax to describe a machine.

```
Describing a machine

>>-onpladm describe machine -machinename------------------------>

   .------------------------.
   V                        |
>----+--------------------+-+----------------------------------><
     +- -F -specfilename-+
     '- -S -servername---'
```

Element	Purpose	Key considerations
-F *specfilename*	Sets the specification file	The default value is the standard output.
machinename	Sets the machine name	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Related reference:
Create a machine type

List all existing machine types

The following diagram illustrates the syntax to list the machine types.

```
Listing all machines

>>-onpladm list machine--+-----------------+-------------------><
                         '- -S -servername-'
```

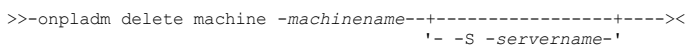
Element	Purpose	Key considerations
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Delete a machine type

You can only delete a machine type by using a specification file.

The following diagram illustrates the syntax to delete a machine type.

Deleting a Machine



Element	Purpose	Key considerations
machinename	Sets the machine type	None
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Define database operations

You can perform the following database operations with the **onpladm** utility:

- Create load and unload jobs for all tables in a database with a single command.
- Load or unload all tables in a database.

- Create a database project
- Configure target-server attributes
  To configure target-server attributes, you must set target-server attribute values and list target-server defaults.

Create a database project

When you create a database project with a single command, the **onpladm** utility:

- Creates all load or unload jobs for every table in a database
- Creates all the associated project HPL objects required for the jobs
- Groups the load or unload jobs and associated project HPL jobs into a single project that has the same name as the database

When you create a database project, you must specify the data-files source directory name or the tape-device path.

If you specify the data-files source directory, the files that the **onpladm** utility creates have the following format:

***PREFIX_DATABASE_TABLE*.unl**

PREFIX is an option that you specify on the command line, *DATABASE* is the name of the target database, and *TABLE* is the name of the target-table name.

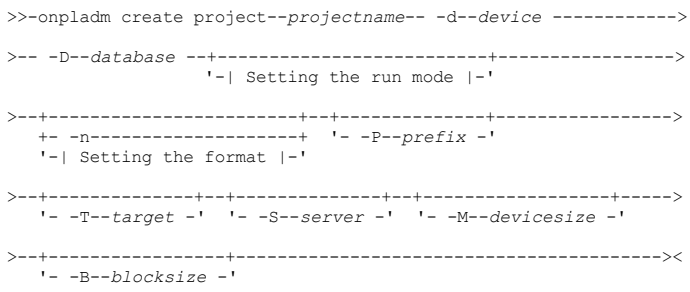
If you do not specify a prefix, the **onpladm** utility creates files of the following format:

***DATABASE_TABLE*.unl**

The **onpladm** utility truncates the format file name if it is longer than the maximum file name length of 18 characters.

The following diagram illustrates the syntax to create a project for all the tables in a database.

Creating a database project



Element	Purpose	Key considerations
-B *blocksize*	Sets the tape I/O block size (bytes)	No default value.
-d *device*	Sets the name of device, such as a file, device array, tape, or pipe	No default value.

Element	Purpose	Key considerations
-D *database*	Name of the target database that contains the information to be loaded or unloaded	No default value.
-M *devicesize*	Tape device size in kilobytes	The device size must be greater than zero.
-n	Sets no-conversion express job	None
-P *prefix*	Filename prefix for the files to be used as devices	None
projectname	Identifies the project where the format and map are stored	None
-S *server*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.
-T *target*	Name of the target server to which the data will download	The default is the value of the INFORMIXSERVER environment variable.

The following diagram illustrates the syntax to set the run mode with the **-f** option.

```
Setting the run mode

|-- -f--+---+--+---+--+---+-------------------------------------|
        '-d-'  '-c-'  '-N-'
```

Element	Purpose	Key considerations
c	Sets mode to deluxe mode.	If this flag is not set, **onpladm** uses express mode.
d	Treats data source as a tape device	None
N	Allows deluxe mode load without replication	None

The following diagram illustrates the syntax to set the format type with the **-z** option.

```
Setting the format

|-- -z--+----+-----------------------------------------------------|
        +-D--+
        +-FI-+
        +-FA-+
        +-FB-+
        +-C--+
        '-CB-'
```

Element	Purpose	Key considerations
D	Sets the format to delimited	This is the default value. See Delimited records.
FI	Sets the format to fixed internal	See Fixed-length records.
FA	Sets the format to fixed ASCII	See Fixed-length records.
FB	Sets the format to fixed binary	See Fixed-length records.
C	Sets the format to COBOL	See COBOL records.
CB	Sets the format to COBOL (byte)	See COBOL records.

Configure target-server attributes

To configure target-server attributes, you must set target-server attribute values and list target-server defaults.

- Set target-server attribute values
- List target-server defaults

Set target-server attribute values

The following diagram illustrates the syntax to set default values for target server attributes.

```
Setting target server attribute values

>>-onpladm configure defaults--+-----------------+-------------->
                               '- -s -servername-'

>-- -m -machinetype-- -c -data_codeset--+-----------------+---><
                                        '- -S -servername-'
```

Element	Purpose	Key considerations

Element	Purpose	Key considerations
- c *data_codeset*	Character set for data files	The character set of the database is determined by the DB_LOCALE environment variable. For information about locales and code sets, see the *IBM® Informix® GLS User's Guide*.
-m *machinetype*	Machine type	None
-s *servername*	Database server for which defaults are set	If a server is not specified, the default information within the onpload database that describes all database servers is modified.
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

List target-server defaults

The following diagram illustrates the syntax to list target-server default values.

```
Listing target server defaults

>>-onpladm list defaults--+-----------------+------------------->
                          '- -s -servername-'

>--+-----------------+-----------------------------------------><
   '- -S -servername-'
```

Element	Purpose	Key considerations
-s *servername*	Database server for which defaults are set	If a server is not specified, the default information within the onpload database that describes all database servers is modified.
-S *servername*	Sets the onpload database server	The default is the value of the INFORMIXSERVER environment variable.

Appendixes

This section contains additional reference information.

- The onpload database
  The tables in the onpload database hold information that the **onpload** utility uses. This section describes the tables in the onpload database that you create or modify with **ipload**.
- High-Performance Loader configuration file
  The default **$INFORMIXDIR/etc/plconfig.std** file on UNIX or **%\INFORMIXDIR%\etc\plconfig.std** on Windows is the *high-performance loader configuration file*.
- Picture strings
  The HPL uses two types of picture strings: COBOL picture strings and other picture strings.
- Match condition operators and characters
  This section describes the operators that are available when you match text and it provides an example of each operator.
- Custom-conversion functions
  Custom-conversion functions allow you to add additional data conversion capability to the High-Performance Loader (HPL). This feature lets **onpload** call a custom-conversion function during the data-conversion process.
- The onstat -j option
  The **-j** option of the **onstat** utility provides special information about the status of an **onpload** job. The **-j** option provides an interactive mode that is analogous to **onstat -i**.
- The HPL log-file and pop-up messages
  This section provides explanatory notes and corrective actions for unnumbered messages that print in the High-Performance Loader (HPL) log file. The section also includes information specific to messages that are returned to standard output or appear in a pop-up dialog box (depending on the way you started **onpload**).
- Custom drivers
  If your operating system supports dynamic linking of libraries, you can use a custom driver to extend the functionality of the High-Performance Loader (HPL) to support different file types or access mechanisms.
- Run load and unload jobs on a Windows computer
- Conversion and reversion scripts for HPL database migration
  When you convert or revert to different versions of the database server, you can use conversion and reversion scripts to manually upgrade or revert your onpload database. You must use these scripts if you are required to upgrade between the same server versions.

The onpload database

The tables in the onpload database hold information that the **onpload** utility uses. This section describes the tables in the onpload database that you create or modify with **ipload**.

When you start **ipload**, **ipload** looks for a database named onpload on the database server that your INFORMIXSERVER environment variable specifies. If the onpload database is not present, **ipload** creates an onpload database as a non-ANSI database.

When **ipload** creates an onpload database, it populates some of the tables in the database with default values. You can use DB-Acess to examine the values in the tables. However, it is recommended that you always use **ipload** to change the onpload database.

- The defaults table in the onpload database
  The **defaults** table contains default values that the High-Performance Loader (HPL) uses. When **ipload** creates the onpload database, it inserts a single row into this table. This row specifies the default configuration assumptions for the database server, the type of computer, and the data code set.
- The delimiters table in the onpload database
  The **ipload** utility uses the values in the **delimiters** table to display the field-delimiter values that the Delimiter Options window shows. When **ipload** creates the onpload database, it inserts values into this table. The values in the **delimiters** table are for reference and do not change.
- The device table in the onpload database
  The **device** table defines the elements of a device array. Use the Device-Array Definition window to modify this table.
- The driver table in the onpload database
- The filteritem table in the onpload database
  The **filteritem** table defines the conditions to be applied to load data to filter out records. Each filter item is attached to a particular field of a record in a data file. Use the filter options to modify this table.
- The filters table in the onpload database
  The **filters** table assigns a unique number to each group of filter items that together form a filter. Each filter is associated with a project and a format definition. Use the Filter-Definition window to create or modify a filter.
- The formatitem table in the onpload database
  The **formatitem** table defines the data-file records. Each field of a data file is described by an entry in this table. Use the Records Format window to prepare the record formats.
- The formats table in the onpload database
  The **formats** table defines the basic information for a record format. Use the Records Format window to modify this table.
- The language table in the onpload database
  The **onpload** utility does not use the **language** table at this time.
- The machines table in the onpload database
  The **machines** table defines the binary type sizes and byte order for different computers. The High-Performance Loader (HPL) uses this information when you transfer binary data.
- The mapitem table in the onpload database
  The **mapitem** table defines the relationship between the columns of a database table and the record fields of a data file. The table stores pairs of column/record entries. The map options modify this table.
- The mapoption table in the onpload database
  The **mapoption** table defines conversion options for the mapping pairs that are defined in **mapitem** table. Use the Mapping Options window to modify this table.
- The maps table in the onpload database
  The **maps** table defines record-to-table mappings (for loads) and query-to-record mappings (for unloads). Use the map options to modify this table.
- The note table in the onpload database
  The **note** table holds comments that you can store about the components that are used for loads and unloads. You can store notes about all of the **onpload** components: projects, devices, formats, maps, queries, filters, and load and unload jobs.
- The project table in the onpload database
  The **project** table lists the projects in this onpload database. Use the Project window to modify this table.
- The query table in the onpload database
  The **query** table stores the queries that are used for unloading data from an IBM® Informix® database. Use the Query-Definition window to modify this table.
- The session table in the onpload database
  The **session** table controls the parameters that **onpload** uses to start a load or unload job.

Related concepts:
Load and unload maps
Mapping options
The Unload Job windows
The Load Job windows
Related tasks:
Changing the load options
Related reference:
Start the ipload utility

The defaults table in the onpload database

The **defaults** table contains default values that the High-Performance Loader (HPL) uses. When **ipload** creates the onpload database, it inserts a single row into this table. This row specifies the default configuration assumptions for the database server, the type of computer, and the data code set.

Column	Type	Description
node	CHAR(18)	The name of a database server
machine	CHAR(18)	Specifies the default machine type (foreign key to the **machines** table)
datatype	CHAR(18)	The code set of the data file
dbgls	CHAR(18)	Reserved Used previously for the code set of the target database

You can specify a set of defaults for each database server. If this table does not contain an entry for a database server, the database uses the defaults that the record named **default** specifies.

Use the Defaults window to modify this table.

Related concepts:
Modify the onpload default values

The delimiters table in the onpload database

The **ipload** utility uses the values in the **delimiters** table to display the field-delimiter values that the Delimiter Options window shows. When **ipload** creates the onpload database, it inserts values into this table. The values in the **delimiters** table are for reference and do not change.

Column	Type	Description
hex	CHAR(2)	Hexadecimal representation of the delimiter
octal	CHAR(4)	Octal representation of the delimiter
ascii	CHAR(15)	ASCII characters (printable) that form the delimiter
control	CHAR(10)	Control character sequence that generates the delimiter

Related tasks:
Modifying delimited-format options

The device table in the onpload database

The **device** table defines the elements of a device array. Use the Device-Array Definition window to modify this table.

Column	Type	Description
name	CHAR(18)	Name of the device array described in this row (primary key)
seq	INTEGER	Device number within the device array (primary key)
type	CHAR(5)	Device type (pipe (UNIX only), file, or tape)
file	CHAR(128)	File or device to be accessed by this array element
blocksize	INTEGER	I/O blocksize (tape devices only)
devicesize	INTEGER	Capacity of device (tape devices only)
pipecommand	CHAR(128)	The pipe command to invoke when **onpload** starts to access to the device element (UNIX only)
lockflag	CHAR(1)	Flag for locking mechanism that **ipload** uses
header	TEXT	The tape header for a device that DDR uses

Related concepts:
The Device-Array Definition window

The driver table in the onpload database

The **onpload** utility uses different set routines, called drivers, to handle different file formats. For example, the delimited driver handles delimited file formats. The routines in a driver process data unloaded from or loaded into the data file. The **onpload** utility includes drivers for widely used data-file formats. You can prepare additional, custom drivers for other formats and bind them into the **onpload** shared library. The set of available drivers is stored in the **driver** table.

Column	Type	Description
drivername	CHAR(18)	Name of driver (primary key)
drivertype	CHAR(1)	Data-file format: Fixed, Delimited, COBOL

You can use the procedure that Custom drivers describes to build custom drivers. A custom driver takes data from a file and constructs input for **onpload** that is in a format that **onpload** recognizes (Fixed, Delimited, COBOL).

Related concepts:
The Drivers window

The filteritem table in the onpload database

The **filteritem** table defines the conditions to be applied to load data to filter out records. Each filter item is attached to a particular field of a record in a data file. Use the filter options to modify this table.

Column	Type	Description
formid	INTEGER	Filter identifier (foreign key to the **filters** table)
seq	INTEGER	Specifies the order in which the filter items (the match expression) are applied
fname	CHAR(128)	The name of the field that this filter affects
option	CHAR(7)	Specifies the disposition of a record (discard or keep) when the match criterion is true

Column	Type	Description
match	CHAR(60)	Match expression that is applied to data field

Related tasks:
Creating a filter

The filters table in the onpload database

The **filters** table assigns a unique number to each group of filter items that together form a filter. Each filter is associated with a project and a format definition. Use the Filter-Definition window to create or modify a filter.

Column	Type	Description
formid	SERIAL	Filter identifier (primary key)
projectid	INTEGER	Project with which this filter is associated (foreign key to the **project** table)
formatid	INTEGER	Format identifier of the format definition to which this filter applies (foreign key to the **formats** table)
name	CHAR(128)	The name of the filter
lockflag	CHAR(1)	Flag for locking mechanism used by **ipload**

Related tasks:
Creating a filter

The formatitem table in the onpload database

The **formatitem** table defines the data-file records. Each field of a data file is described by an entry in this table. Use the Records Format window to prepare the record formats.

Table 1 lists the possible values for the **ftype** column. For more information, see Figure 2.

Column	Type	Description
formid	INTEGER	Record format identifier (foreign key to the **formats** table)
seq	INTEGER	Item sequence number for internal organization
fname	CHAR(128)	Name of record field
ftype	INTEGER	A number that indicates the type of data in the field
bytes	INTEGER	Number of bytes in field
decimals	INTEGER	Number of decimal values to format when converting to ASCII
offset	INTEGER	Offset in record image where field starts
qual	INTEGER	IBM® Informix® DATETIME/INTERVAL qualifier
picture	CHAR(15)	COBOL picture definition

Table 1. Possible values for the ftype column

Value for ftype	Type of data
1	Character (fixed and delimited)
2	Date
3	Short integer
4	Integer
5	Long Integer
6	Floating-point vale
7	Double floating-point value
8	Unsigned short integer
9	Unsigned integer
10	Unsigned long integer
11	UNIX date
18	Packed Decimal
19	Zoned decimal
20	Comp-1
21	Comp-2
22	Comp-3

Value for ftype	Type of data
23	Comp-4
24	Comp-5
25	Comp-6
26	Comp-X
27	Comp-N
28	Character (COBOL)
34	Blob Length
35	Blob File
36	Blob HexASCII
37	Blob Text
39	INT8
40	SERIAL8
41	BOOLEAN
42	Extended Type String
43	Extended Type HexASCII
44	Extended Type Binary
45	Extended Type StringLength
46	Extended Type BinaryLength
49	BIGINT
50	BIGSERIAL
51	INT64 (64-bit integer)

Related concepts:
Formats of supported datafile records

The formats table in the onpload database

The **formats** table defines the basic information for a record format. Use the Records Format window to modify this table.

Column	Type	Description
formid	SERIAL	Unique format identifier (primary key)
projectid	INTEGER	Project to which the format is assigned (foreign key to the **project** table)
name	CHAR(128)	Name of format
type	CHAR(10)	Data-file format: Fixed, Delimited, COBOL
driver	CHAR(18)	Driver to use to access data records
machine	CHAR(18)	Machine name that defines binary-data parameters (foreign key to the **machinename** column of the **machines** table)
datatype	CHAR(18)	Character code set to use for conversion of data records
recordlength	INTEGER	Length in bytes of a fixed-format record
recordstrt	CHAR(15)	Record-start sequence for delimited format
recordstrty	CHAR(10)	Type of the record-start sequence:Hex, Octal, ASCII, or Decimal
recordend	CHAR(15)	Record-end sequence for delimited format
recordendt	CHAR(10)	Type of the record-end sequence:Hex, Octal, ASCII, or Decimal
fieldsep	CHAR(15)	Field-separator sequence for delimited format
fieldsept	CHAR(10)	Type of the field-separator sequence:Hex, Octal, ASCII, or Decimal
fieldstrt	CHAR(15)	Field-start sequence for delimited format
fieldstrty	CHAR(10)	Type of the field-start sequence: Hex, Octal, ASCII, or Decimal
fieldend	CHAR(15)	Field-end sequence for delimited format
fieldendt	CHAR(10)	Record-end sequence separator type:Hex, Octal, ASCII, or Decimal
lockflag	CHAR(1)	Flag for locking mechanism that **ipload** uses

Related concepts:
Formats of supported datafile records
Related tasks:
Creating a fixed format
Related reference:
Format options

The language table in the onpload database

The **onpload** utility does not use the **language** table at this time.

The machines table in the onpload database

The **machines** table defines the binary type sizes and byte order for different computers. The High-Performance Loader (HPL) uses this information when you transfer binary data.

When **ipload** creates the onpload database, it inserts definitions for several different types of computers into this table. To transfer binary data to or from a computer that is not described in this table, you must create a machine definition by using the Machines window.

Column	Type	Description
machinename	CHAR(18)	Computer name or type (primary key)
byteorder	CHAR(3)	Binary byte ordering: LSB or MSB
shortsize	INTEGER	Size of a short integer
intsize	INTEGER	Size of an integer
longsize	INTEGER	Size of a long integer
floatsize	INTEGER	Size of a float value
doublesize	INTEGER	Size of a double value

Related concepts:
Modify the machine description
The Machines window

The mapitem table in the onpload database

The **mapitem** table defines the relationship between the columns of a database table and the record fields of a data file. The table stores pairs of column/record entries. The map options modify this table.

Column	Type	Description
formid	INTEGER	Specifies the map to which this record belongs (foreign key to the **maps** table)
seq	INTEGER	Unique identifier for the database-column/data file-record pair
colname	VARCHAR(128,0)	Name of database column
fname	CHAR(128)	Name of field in a data-file record

Related concepts:
Load and unload maps

The mapoption table in the onpload database

The **mapoption** table defines conversion options for the mapping pairs that are defined in **mapitem** table. Use the Mapping Options window to modify this table.

Column	Type	Description
formid	INTEGER	Specifies the map to which this record belongs (foreign key to the **maps** table)
seq	INTEGER	The database-column and/or data file-record pair to which this option applies (foreign key to the **mapitem** table)
bytes	INTEGER	Maximum number of bytes to transfer from a field of a data file
minvalue	FLOAT	Minimum value allowed in field
maxvalue	FLOAT	Maximum value allowed in field
ccase	CHAR(18)	Case conversion option: None, Lower, Upper, Proper Noun
justify	CHAR(18)	String justification to perform: None, Left, Right, Center
fill	CHAR(1)	Fill character for string padding
picture	CHAR(55)	Picture mask to apply to target data
coloffset	INTEGER	Offset in column at which to start data transfer
recoffset	INTEGER	Offset in record field from which to start data extract

Column	Type	Description
function	CHAR(55)	Custom function to call
looktable	CHAR(18)	Not in use
matchcol	CHAR(18)	Not in use
coldefault	CHAR(18)	Default value to set on column: ASCII HEX or ASCII binary
inputcode	CHAR(18)	Format in which the BYTE or TEXT data is stored in the data file: ASCII HEX or ASCII binary
storecode	CHAR(18)	Format in which to store the BYTE or TEXT data
blobcolumn	CHAR(18)	The column that contains the name of the file where the BYTE or TEXT data is stored

If the values of **inputcode** and **storecode** are different, **onpload** converts the contents of the BYTE or TEXT data.

Related tasks:
Defining the mapping options

The maps table in the onpload database

The **maps** table defines record-to-table mappings (for loads) and query-to-record mappings (for unloads). Use the map options to modify this table.

Column	Type	Description
projectid	INTEGER	Project to which this map is assigned (foreign key to the **project** table)
formid	SERIAL	Unique identifier for map (primary key)
name	CHAR(128)	Name of map
type	CHAR(6)	Specifies whether the map is a load or unload map; possible values include: • Record (load map) • Query (unload map)
dbname	CHAR(30)	Name of load or unload database
qtable	CHAR(18)	Name of table to be loaded; used only for loads
query	CHAR(128)	Name of query; used only for unloads
formatid	INTEGER	Identifier of the format that this map uses (foreign key to the **format** table)
lockflag	CHAR(1)	Flag for locking mechanism that **ipload** uses

Related concepts:
Load and unload maps

The note table in the onpload database

The **note** table holds comments that you can store about the components that are used for loads and unloads. You can store notes about all of the **onpload** components: projects, devices, formats, maps, queries, filters, and load and unload jobs.

Column	Type	Description
type	CHAR(18)	Specifies the type of component to which this note is attached
formid	INTEGER	Corresponds to the **formid** of the component specified in the **type** column (The two columns together uniquely identify the component to which the note is attached.)
projectid	INTEGER	ID of project to which this note belongs (foreign key to the **project** table)
createdate	DATE	Date that the note was created
modifydate	DATE	Date that the note was last modified
note	TEXT	Text of the note

Related concepts:
The Notes button

The project table in the onpload database

The **project** table lists the projects in this onpload database. Use the Project window to modify this table.

Column	Type	Description

Column	Type	Description
name	CHAR(128)	Name of object
projectid	SERIAL	Uniquely identifies the project (primary key)
dcreate	DATE	Date that the project was created

Related concepts:
Select or create a project with the Projects window
Project organization

The query table in the onpload database

The **query** table stores the queries that are used for unloading data from an IBM® Informix® database. Use the Query-Definition window to modify this table.

Column	Type	Description
formid	SERIAL	Unique number that identifies this query (primary key)
projectid	INTEGER	Number of the project that includes this query (foreign key to the **projects** table)
name	CHAR(128)	Name of the query
database	CHAR(30)	Name of database being queried
arrayname	CHAR(128)	Not in use
lockflag	CHAR(1)	Flag for locking mechanism that **ipload** uses
sqlselect	TEXT	SQL statement of the query

Related concepts:
HPL queries
Related tasks:
Creating a query

The session table in the onpload database

The **session** table controls the parameters that **onpload** uses to start a load or unload job.

Table 1. Columns in the session table

Column	Type	Description
sessiontype	CHAR(1)	Describes the type of load or unload session: • U = Job is driven by the user interface. • N = Job expects a socket interface and is removed when the job is finished. • S = Job is run from the command line.
automate	CHAR(1)	Flag for automatically creating maps and formats at run time: • Y = Create automatically. • blank = Do not create.
lockflag	CHAR(1)	Flag for locking mechanism that **ipload** uses
sessionid	SERIAL	Session identifier (primary key)
name	CHAR(130)	Name of the load or unload job. This name appears in the command line displayed in the Load Job Select or Unload Job Select window.
status	CHAR(1)	Job status: • R = Running • C = Connecting • S = Starting • blank = Job is complete.
server	CHAR(40)	Override default server to load and unload
map	CHAR(18)	Name of the map that controls the load (foreign key to the **name** column of the **maps** table; the **maps** table specifies the format and, for unload jobs, the query)
infile	CHAR(160)	Name of the device array (foreign key to the **name** column of the **device** table)
hostname	CHAR(40)	Name of the computer on which the **onpload** utility is running
dbname	CHAR(30)	Name of database to be loaded or unloaded
filter	CHAR(128)	Filter for screening import data (foreign key to the **name** column of the **filters** table)
recordfilter	CHAR(384)	File in which to store filtered records

Column	Type	Description
suspensefile	CHAR(384)	File in which to store records that do not pass conversion
rejectfile	CHAR(384)	File in which to place records that the database server rejected
logfile	CHAR(384)	File in which to place session status messages
projectid	INTEGER	Project for maps and formats (foreign key to the **project** table)
headersize	INTEGER	Size in bytes of header information to strip from input
quiet	INTEGER	If true, suppresses status message output
tracelevel	INTEGER	Higher values result in more status messages
sourcetrace	INTEGER	If true, source and module line numbers are placed in status message outputs
multicthread	INTEGER	Sets the maximum number of conversion threads that you can invoke on a device
blocksize	INTEGER	I/O block size for accessing device
filetype	INTEGER	Specifies the type of file: tape, array, pipe (UNIX only)
number_records	INTEGER	Specifies the number of records to load
start_record	INTEGER	Specifies the number of the record at which to start loading
maxerrors	INTEGER	Maximum number of errors to allow before aborting the load or unload
swapbytes	INTEGER	Specifies the number of bytes to swap (If *swapbytes* is 4, the first 4 bytes are swapped with the next 4 bytes. If blank, bytes are not swapped.)
runmode	INTEGER	Contains a value that qualifies **onpload** to perform a load or unload operation. You can determine the run mode by selecting the **loadmode** and **runmode** fields from the session table. For load operations, the **runmode** is a combination of the following values: • 129 = Deluxe mode + conversion • 130 = Express® mode + conversion • 385 = Deluxe mode + no violations + conversion • 386 = Express mode + conversion + no violations • 4225 = Deluxe mode + no replication + conversion • 4481 = Deluxe mode + no replication + no violations + conversion For example, you could have this command for a load operation: `onpladm create job j4 -flN -D tst -t tab1 -d data.unl` For this command for a load job, the **runmode** value is: `0x00001081 = Deluxe+Conversion+NoReplication` For unload operations, the **runmode** is a combination of the following values: • 2 = No conversion • 129 = Conversion (128) + dirty read isolation level (1) • 130 = Conversion (128) + committed read isolation level (2) • 131 = Conversion (128) + cursor stability isolation level (3) • 132 = conversion (128) + repeatable read isolation level (4)
loadmode	INTEGER	Type of job: • 1 = Load • 2 = Unload
caseconvert	INTEGER	Case conversion type. Convert to: • U or u = uppercase • L or l = lowercase • P or p = proper names
commitinterval	INTEGER	Commit interval for committing a load transaction. The value is specified in the Load Options window, Figure 1. The commit interval applies only to deluxe mode.
socketport	INTEGER	Set by **onpload** to specify the port number of the connection
numtapes	INTEGER	Number of tapes to load

Tip: Deluxe-mode loads do not support the “no conversion” option and the “with conversion and do not generate violations table” option.

Related concepts:

The Load Job windows

High-Performance Loader configuration file

The default **$INFORMIXDIR/etc/plconfig.std** file on UNIX or **%\INFORMIXDIR%\etc\plconfig.std** on Windows is the *high-performance loader configuration file.*

The file is similar to the onconfig file in the **etc** directory in **$INFORMIXDIR**. The **plconfig.std** file sets various **onpload** buffer and system configuration parameters. You can modify the parameters to maximize resource utilization.

The PLCONFIG environment variable specifies an alternative name for the HPL configuration file. This file must be in the **etc** directory in **$INFORMIXDIR**. If you do not set the PLCONFIG environment variable, the default name of the file is **plconfig.std**.

- HPL configuration parameter descriptions
- HPL configuration parameter file conventions
  Each parameter in the **plconfig.std** file in the **etc** directory in **$INFORMIXDIR** is on a separate line. The file can also contain blank lines and comment lines that start with a # symbol.
- The AIOBUFFERS configuration parameter
  The AIOBUFFERS configuration parameter sets the number of buffers used to transport data from converter threads to the AIO handler.
- The AIOBUFSIZE configuration parameter
  The AIOBUFSIZE configuration parameter sets the size of the AIO memory buffers that transfer data to and from tapes and files. The High-Performance Loader (HPL) uses the AIO buffers to pass data between the converters and the I/O drivers.
- The CONVERTTHREADS configuration parameter
  The CONVERTTHREADS configuration parameter sets the number of convert threads for each file I/O device. The convert threads run on the convert VPs.
- The CONVERTVPS configuration parameter
  The CONVERTVPS parameter limits the maximum number of VPs used for convert threads. This parameter limits the number of VPs that the **onpload** client uses so that **onpload** does not monopolize system resources.
- The HPLAPIVERSION configuration parameter
  The HPLAPIVERSION configuration parameter specifies whether to use custom conversion or driver functions with three or four arguments. Using four arguments allows different lengths for data in the input and output buffers.
- The HPL_DYNAMIC_LIB_PATH configuration parameter
- The STRMBUFFERS configuration parameter
  The STRMBUFFERS parameter sets the number of server-stream buffers per device. The **onpload** utility sends data to the database server through a *server stream*. The server stream is a set of shared-memory buffers. The memory for the server-stream buffer is allocated from the memory allocated for the database server.
- The STRMBUFFSIZE configuration parameter
  The STRMBUFFSIZE configuration parameter sets the size of a server-stream buffer. Larger buffers are more efficient because moving buffers around requires less overhead.

Related concepts:
The onpload configuration parameters
Related reference:
The PLCONFIG environment variable

HPL configuration parameter descriptions

The description of each configuration parameter has one or more of the following fields (depending on their relevance):

Default value
: The value that appears in the **plconfig.std** file unless you explicitly change it

Units
: The units in which the parameter is expressed

Range of values
: The possible values for this parameter

See
: Cross-reference to further discussion

HPL configuration parameter file conventions

Each parameter in the **plconfig.std** file in the **etc** directory in **$INFORMIXDIR** is on a separate line. The file can also contain blank lines and comment lines that start with a # symbol.

The syntax of a parameter line is as follows:

```
PARAMETER_NAME parameter_value  # optional comment
```

Parameters and their values are case-sensitive. The parameter names are always all uppercase letters. If the parameter-value entry is described with uppercase letters, you must use uppercase. You must put white space (tabs or spaces or both) between the parameter name, parameter value, and optional comment. Do not use any tabs or spaces within a parameter value.

The AIOBUFFERS configuration parameter

The AIOBUFFERS configuration parameter sets the number of buffers used to transport data from converter threads to the AIO handler.

You must set the AIOBUFFERS parameter on Windows computers to a minimum of 8.

Default value
: Maximum of (4,CONVERTTHREADS)

Recommended value
: Maximum of (4, 2*CONVERTTHREADS)

Range of values
: Integer value > 4

See

Assess information for loading or unloading external data

The AIOBUFSIZE configuration parameter

The AIOBUFSIZE configuration parameter sets the size of the AIO memory buffers that transfer data to and from tapes and files. The High-Performance Loader (HPL) uses the AIO buffers to pass data between the converters and the I/O drivers.

The AIOBUFSIZE parameter is not the same as the tape-block size that you can set in the device arrays (see Figure 2). The tape-block size lets you control the size of the block that the device controller sends to the tape drive, while AIOBUFSIZE lets you control the size of internal buffers that pass data. If your computer has memory available, you can improve performance by increasing the AIOBUFSIZE parameter.

Default value

64

Units

Kilobytes

Range of values

Minimum: 0.5 KB (512 bytes)

Maximum: depends on operating system resources

See

Assess information for loading or unloading external data

The CONVERTTHREADS configuration parameter

The CONVERTTHREADS configuration parameter sets the number of convert threads for each file I/O device. The convert threads run on the convert VPs.

If you are doing a convert-intensive job, increasing CONVERTTHREADS can improve performance on multiple-CPU computers. For convert-intensive jobs, set CONVERTTHREADS to `2` or `3` as a starting point for performance tuning. Except for computers with many CPUs, the useful maximum number of CONVERTTHREADS is almost always less than `10`.

The total number of convert threads that **onpload** uses is as follows:

```
CONVERTTHREADS * numdevices
```

where *numdevices* is the number of devices in the current device array.
Having more than one converter per thread, in general, allows the conversion phase to run faster given that CPU resources are available. Conversion can be a CPU-intensive phase if complex conversions are being performed.

Default value

1

Range of values

Minimum: 1

Maximum: depends on computer configuration

See

Assess information for loading or unloading external data

The CONVERTVPS configuration parameter

The CONVERTVPS parameter limits the maximum number of VPs used for convert threads. This parameter limits the number of VPs that the **onpload** client uses so that **onpload** does not monopolize system resources.

Setting CONVERTVPS too large can cause performance degradation. Do not set more converter VPs than there are physical CPUs. If the number of CONVERTVPS exceeds the number of physical CPUs, system resources are consumed with no performance benefit.

On single-CPU computers, increasing this parameter has a negative effect on performance.

Default value

Single-processor computer: 1

Multiprocessor computer: 50 percent of physical CPUs

Range of values

From 1 to the number of physical CPUs

See

Assess information for loading or unloading external data

The HPLAPIVERSION configuration parameter

The HPLAPIVERSION configuration parameter specifies whether to use custom conversion or driver functions with three or four arguments. Using four arguments allows different lengths for data in the input and output buffers.

Default value

0

Range of values

0 or 1

0 = The custom conversion or driver function receives three arguments:

- The buffer into which the output should be placed
- The maximum length of the output buffer
- The value of the input field

1 = The custom conversion or driver function receives four arguments:

- The buffer into which the output should be placed
- The maximum length of the output buffer
- The value of the input field
- The length of the input value

See

The onpload conversion process

The HPL_DYNAMIC_LIB_PATH configuration parameter

The **ipldd11a**.***SOLIBSUFFIX*** shared-library file can contain custom-code files. You can add custom drivers or custom conversion functions to this file to extend the functionality of the High-Performance Loader (HPL). For more information about custom drivers, see Custom drivers. For more information about custom conversion functions, see Custom-conversion functions.

Previous versions of the database server required the **ipldd11a**.***SOLIBSUFFIX*** file to be installed in the **/usr/lib** directory on Solaris. Although you can set the value of HPL_DYNAMIC_LIB_PATH to **/usr/lib**, doing so creates a security risk.

Default value

$INFORMIXDIR/lib/ipldd11a.so

Range of values

Any valid directory, plus **ipldd11a**.***SOLIBSUFFIX***. (***SOLIBSUFFIX*** is the shared-library suffix for your operating system.)
For security reasons, you should keep all shared libraries used by the database server in directories under $INFORMIXDIR.

See

Rebuilding the shared-library file

If you use customized files with the High-Performance Loader, set the HPL_DYNAMIC_LIB_PATH configuration parameter in the plconfig file to the location of the custom-code shared library.

The STRMBUFFERS configuration parameter

The STRMBUFFERS parameter sets the number of server-stream buffers per device. The **onpload** utility sends data to the database server through a *server stream*. The server stream is a set of shared-memory buffers. The memory for the server-stream buffer is allocated from the memory allocated for the database server.

Each device has a separate server stream with STRMBUFFERS buffers. Thus the total number of stream buffers is as follows:

```
STRMBUFFERS * numdevices
```

where *numdevices* is the number of devices in the current array.

Default value

Maximum of (4,2*CONVERTTHREADS)

Recommended value

Maximum of (4,2*CONVERTTHREADS)

Range of values

Integer > 4

See

Assess information for loading or unloading external data

The STRMBUFFSIZE configuration parameter

The STRMBUFFSIZE configuration parameter sets the size of a server-stream buffer. Larger buffers are more efficient because moving buffers around requires less overhead.

Default value

64

Units

Kilobytes

Range of values

Minimum: 2 * operating system page size
Maximum: depends on operating system resources

See

Assess information for loading or unloading external data

Picture strings

The HPL uses two types of picture strings: COBOL picture strings and other picture strings.

COBOL picture strings describe a data field in a file that a COBOL program generates. For a discussion of COBOL picture strings, see COBOL records. The other picture-string type reformats and masks character data. This appendix discusses the non-COBOL picture strings.

Picture strings allow you to insert constants, strip unwanted characters, and organize the position of character data. Picture strings have three basic types: alphanumeric, numeric, and date. Each type is handled uniquely. The picture-string type is determined by the control characters that you use to specify the picture.

You specify the picture string in the **Picture** text box in the Mapping Options window. For information about the Mapping Options window, see Mapping options.

- Alphanumeric pictures
  Alphanumeric pictures control formatting of alphanumeric strings. An alphanumeric picture allows you to mix constant characters in the picture specification with the data being processed. You can also mask out unwanted character types.
- Numeric pictures
  Numeric pictures allow you to decode and reformat integer and decimal numeric values. A value is interpreted as a numeric value only if its picture string contains numeric picture-control characters.
- Date pictures
  When you load data, the date-format picture specifies how the High-Performance Loader (HPL) formats the input data before it writes the data into a database. When you extract data from a database, the date-format picture specifies how the HPL reformats the date before it writes the date to the output.

Alphanumeric pictures

Alphanumeric pictures control formatting of alphanumeric strings. An alphanumeric picture allows you to mix constant characters in the picture specification with the data being processed. You can also mask out unwanted character types.

When the HPL processes an alphanumeric picture, the picture string is scanned until a picture-control character is found. All noncontrol characters in the picture string are placed directly into the output string.

When a control character is found in the picture string, the input data is scanned until a character that matches the type of the picture-replacement character is found. This character is placed in the output string, and the process is repeated.

The alphanumeric picture-control characters are `X`, `a`, `A`, `9`, and `\`. A picture string that includes any of the preceding characters is, by definition, an alphanumeric picture string. All other characters in an alphanumeric picture string are treated as literals and inserted directly into the resulting output string.

The following list describes the behavior of the alphanumeric picture-control characters.

Character
Definition

X
Replaces the control character with any character from input data

A
Replaces the control character with an alphanumeric character from input

a
Replaces the control character with an alphabetic character from input

9
Replaces the control character with a numeric character from input Fills the string with leading 0 characters so that the length of the input string matches the length of the picture specification.

\
Causes the character that follows the backslash to be placed in the output. That is, the character that follows a backslash is not a control character.

The following table lists some examples of alphanumeric pictures.

Picture	Input data	Output data
XX-AJXXXX	12P45-q	12-PJ45-q
AA-\AJAAAA	12P45-q	12-AJP45q
aaaaaaaa	12P45-q	Pq
aa99999	123abc	ab00000

Numeric pictures

Numeric pictures allow you to decode and reformat integer and decimal numeric values. A value is interpreted as a numeric value only if its picture string contains numeric picture-control characters.

The input data is first scanned for the number of digits to the left and right of the decimal point (if any), and for a negative sign that can either precede or follow the data. The picture string is then used to reformat the value. The numeric picture-control characters are `9`, `S`, `V`, and `Z`.

The following list describes the behavior of the numeric picture-control characters.

Character
Definition

9
Replaces the control character with a numeric character

S
: Replaces the control character with a minus sign if the input value is negative

V
: Inserts a decimal point

Z
: Replaces the control character with a numeric character or a leading zero

The following table lists some examples of numeric pictures.

Picture	Input data	Output data	Comment
9999999	123	0000123	Simple reformat
S999.99	123-	-123.00	Sign controlled on output
99V99	123	01.23	Implicit decimal point
99.99	103.455	103.45	Strip decimals

Date pictures

When you load data, the date-format picture specifies how the High-Performance Loader (HPL) formats the input data before it writes the data into a database. When you extract data from a database, the date-format picture specifies how the HPL reformats the date before it writes the date to the output.

The date control characters are `M`, `D`, and `Y`. The following list provides definitions of these control characters.

Character
: **Definition**

D
: Day value

H
: Hour value

M
: Month value or minute value

S
: Second value

Y
: Year value

You can use IBM® Informix® DATETIME strings, such as YYYY/MM/DD HH:MM:SS.

The following table shows some examples of date picture strings.

Picture	DBDATE value	Input	Output
MM/DD/YY	YMD2/	12/20/91	91/12/20
MM/DD/YY	DMY2/	12/20/91	20/12/91
MMDDYY	DMY2/	122091	20/12/91
MM DD YYYY	DMY4/	12/20/1991	20/12/1991
MM/DD/YY	DMY2.	12/20/91	20.12.91
M/D/YY	DMY2/	02/01/91	2/1/91

Match condition operators and characters

This section describes the operators that are available when you match text and it provides an example of each operator.

- Operator descriptions and examples

Operator descriptions and examples

Operator	Description
= *value*	Matches if the character string in, or the value of, the data-record field equals the specified text or value. If you specify a character string, the characters must be delimited by quotes. For example, if you are matching on a field named City, the match condition = `"Dallas"` selects all records whose City field contains the entry Dallas.
value	Equals (=) is the default operator. Thus, this case is equivalent to = *value*, except that the characters do not have to be delimited by quotes. For example, if you are matching on a field named City, the match condition `Dallas` selects all records whose City field contains the entry Dallas.
> *value*	Matches if the data record field is greater than the specified value. For example, if you are matching on a field named Income, the match condition > `50000` selects all records whose Income field contains an entry greater than 50,000. Character strings must be delimited by quotes (> `"Jones"`).
< *value*	Matches if the data record field is less than the specified value. For example, if you are matching on a field named Income, the match condition < `50000` selects all records whose Income field contains an entry less than 50,000. Character strings must be delimited by quotes (< `"Jones"`).

Operator	Description
>= *value*	Matches if the data-record field is equal to or greater than the specified value. For example, if you are matching on a field named Income, the match condition >= `50000` selects all records whose Income field contains an entry of 50,000 or greater. Character strings must be delimited by quotes (>= `"Jones"`).
<= *value*	Matches if the data-record field is less than or equal to the specified value. For example, if you are matching on a field named Income, the match condition <= `50000` selects all records whose Income field contains an entry of 50,000 or less. Character strings must be delimited by quotes (<= `"Jones"`).
<> *value*	Matches if the data-record field is not equal to the specified value. Character strings must be delimited by quotes. For example, if you are matching on a field named State, the match condition <>`"TX"` selects all records whose State field contains an entry other than TX.
between *value1* and *value2*	Matches if the data-record field is between the range specified in value 1 and value 2. For example, if you are matching on a field named Income, the match condition `between 50000 and 100000` selects all records whose Income field contains an entry 50,000 - 100,000. Character strings must be delimited by quotes.
and	Constructs a comparison of two or more items. Matches only if the data record fields match all of the comparisons. The comparisons can only be applied to one field. For example, if you are matching a field named Income, the match condition `> 5000` and `<>` 6000 selects all the records with income greater than 5000, but not a record of 6000.
or	Constructs a comparison of two or more items. Matches if the data record fields match any of the comparisons. For example, if you are matching on a field named City, the match condition = `"Dallas"` or = `"Fort Worth"` selects all records whose City field contains either the entry Dallas or the entry Fort Worth.
NULL	Matches when all characters are blank or when a character is binary zero (null). For example, you might want to discard any records that have all blanks for a name field.
* (asterisk)	Wildcard match of any number of characters in a string. For example, to match on a field that contains the city name and state, the match condition `Dall*` would select records with any of the following entries: • Dallas-Forth Worth • Dallas, TX • Dallas TX
?	Matches any single character in a string. For example, to match on a field that contains a last name, the match condition `Sm?th` would select records with any of the following entries: • Smith • Smyth

Custom-conversion functions

Custom-conversion functions allow you to add additional data conversion capability to the High-Performance Loader (HPL). This feature lets **onpload** call a custom-conversion function during the data-conversion process.

When you create a custom-conversion function, you associate it with a particular mapping of input field to output field. To associate a custom function with a field, enter the name of the function in the **Function** text box of the Mapping Options window. For information about mapping options, see Mapping options.

Although the mapping options associate the custom-conversion function with a particular field, the function can access all the input data fields and all the output data fields through a set of API functions provided with the **onpload** utility.

- Custom conversion example
- The onpload conversion process
  The **onpload** conversion process is identical for both import or export operations.

Custom conversion example

As an example, you might implement custom-conversion functions to do the following, expressed in pseudocode:

```
IF input field 1 satisfies condition A
THEN
DO calculation X on input field 7
OUTPUT data to output column 7
ELSE
DO calculation Y on input field 6
OUTPUT data to output column 5
```

The custom-conversion function feature is available only on computers with operating systems that support dynamic linking.

The onpload conversion process

The **onpload** conversion process is identical for both import or export operations.

The **onpload** utility:

- Extracts the source data from their native format.

- Examines the map.
- Applies the conversions called out in the map.
  Conversion order is implied by the ordering of the source-field names that are specified in the map.

- Calls any custom-conversion function that is specified for a source field.
  The parameters that **onpload** passes to the conversion function depends on the value of the HPLAPIVERSION parameter in the plconfig file:
  - If HPLAPIVERSION is set to `0` or it is not present in the plconfig file, then **onpload** passes the buffer into which the output should be placed, the maximum length of the output buffer, and the value of the input field.
  - If HPLAPIVERSION is set to `1`, then **onpload** passes the buffer into which the output should be placed, the maximum length of the output buffer, the value of the input field, and the length of the input value.

  For more information, see The HPLAPIVERSION configuration parameter.

- If there is a custom-conversion function, applies the value that the custom-conversion function places in the function output buffer to the destination field that is associated with the source field in the map.
- Sends the results to the output generators.

The custom-conversion function API uses ASCII strings as the canonical data type. The API functions present data as ASCII strings and expect data from the custom-conversion functions to be presented as ASCII strings. The API functions convert source data of different types to ASCII strings, and also convert ASCII string data from custom-conversion functions to destination data types.

- Integrating custom conversion functions
  Custom-conversion functions are loaded into the **onpload** executable command through a shared library.
- API functions

Integrating custom conversion functions

Custom-conversion functions are loaded into the **onpload** executable command through a shared library.

To integrate your custom-conversion functions into the **onpload** executable command:

1. Prepare the custom-conversion function table.
   The **onpload** utility uses the entries in a function table to translate custom-function string names that are specified in the load or unload map. You must supply the function table and the custom-conversion functions.

   To code the function table, use the following template for the file plcstcnv.c. You can copy this template from the $INFORMIXDIR/incl/hpl directory. Add as many entries into the **functiontable** array as needed.

   The **onpload** utility searches the **functiontable** array for the string name of the custom-conversion function that the map specifies. The function pointer that is associated with the string name is retrieved and used as the custom-conversion function. In the following template for the file plcstcnv.c, **ycf1** and **ycf2** are the strings that **ipload** uses to find the custom functions **your_conversion_func1** and **your_conversion_func2**. To add custom function string names to the **onpload** database, see Mapping options.

   ```
   /*
    * plcstcnv.c
    */
   #include "pldriver.h"

   extern int your_conversion_func1();
   extern int your_conversion_func2();

   struct functable functiontable[] =
   {
   {"ycf1", your_conversion_func1},
   {"ycf2", your_conversion_func2},
   {0, 0}
   };
   /* end of plcstcnv.c */
   ```

2. Prepare your conversion functions. Use the template in the following example to code your conversion functions:

   ```
   /*
    * your_custom_conversion.c
    */

   /*
    * The argument list must be adhered to.
    */
   int your_conversion_func1(outbuffer, buflen, value)
   char *outbuffer; /* where to put your output */
   intbuflen;/* max size of buffer in bytes*/
   char *value; /* input value */
   {
   /* your processing here */
   }

   int your_conversion_func2(outbuffer, buflen, value)
   char *outbuffer; /* where to put your output */
   intbuflen;/* max size of buffer */
   char *value; /* input value */
   {
   /* your processing here */
   }
   /* end of your_custom_conversion.c */
   ```

3. Rebuild the **onpload** shared-library file ipldd11a.*SOLIBSUFFIX*, (where *SOLIBSUFFIX* is the shared-library suffix for your platform). Follow the instructions in Rebuilding the shared-library file.

The **onpload** utility uses the same library for both custom-conversion functions and custom drivers. When you rebuild the library, if there are custom drivers, you must link the custom-driver code as well as the custom-conversion functions.

4. Install the shared library in the appropriate path for your platform. For example, on Solaris the shared library should be installed in $INFORMIXDIR/lib or any configurable path that is specified by the HPL_DYNAMIC_LIB_PATH configuration parameter.

API functions

Depending on the value of the HPLAPIVERSION parameter in the plconfig file, **onpload** expects your custom-conversion function to have one of the following prototypes.

If the HPLAPIVERSION parameter is set to `0` or HPLAPIVERSION is not present in the plconfig file, use the following prototype:

```
/*
 * input::  char* outbuffer:  where to put your output.
 *  intbuflen: size you have for your output.
 *  char* value:  the input value to work on.
 * return:: 0 to indicate ok.
 *  non-zero  to discard entire record.
 */

int your_func(outbuffer, buflen, value)
char *outbuffer;
intbuflen;
char *value;
{
/* your processing here */
}
```

If the HPLAPIVERSION parameter is set to `1`, use the following prototype:

```
/*
 * input::  char* outbuffer:  where to put your output.
 *  intbuflen: size you have for your output.
 *  char* value:  the input value to work on.
 *  intvallen: size of the input value.
 * return:: 0 to indicate ok.
 *  non-zero  to discard entire record.
 */

int your_func(outbuffer, buflen, value, vallen)
char *outbuffer;
intbuflen;
char *value;
intvallen;
{
/* your processing here */
}
```

To discard an entire record, return a nonzero value. Otherwise, return a zero value.

The following functions support your access to data in the source and destination buffers.

- The DBXget_source_value(fldname,buffer,buflen) routine
  This routine retrieves the source value that is associated with **fldname** and copies the value to the specified buffer.
- The DBXget_dest_value(fldname,buffer,buflen) routine
  This routine retrieves the destination value that is associated with **fldname** and copies the value to the specified buffer.
- The DBXput_dest_value(fldname,buffer) routine
  If a previous conversion has not set the destination value, this routine sets the destination value that is passed to the buffer. The **ipload** utility automatically clips the data value if it is too long.
- The DBXget_dest_length(fldname) routine
  This routine returns the maximum length of the data buffer that is associated with **fldname**.

The DBXget_source_value(fldname,buffer,buflen) routine

This routine retrieves the source value that is associated with **fldname** and copies the value to the specified buffer.

Arguments	I/O	Description
char *fldname	Input	Name of source field, as defined in **ipload** map
char *buffer	Input	Address where **fldname** value is placed
int buflen	Input	Buffer size in bytes

The DBXget_dest_value(fldname,buffer,buflen) routine

This routine retrieves the destination value that is associated with **fldname** and copies the value to the specified buffer.

Arguments	I/O	Description

Arguments	I/O	Description
char *fldname	Input	Name of destination field, as defined in **ipload** map
char *buffer	Input	Address where **fldname** value is placed
int buflen	Input	Buffer size in bytes

The DBXput_dest_value(fldname,buffer) routine

If a previous conversion has not set the destination value, this routine sets the destination value that is passed to the buffer. The **ipload** utility automatically clips the data value if it is too long.

Arguments	I/O	Description
char *fldname	Input	Name of destination field, as defined in **ipload** map
char *buffer	Input	Address where **fldname** value is placed

The DBXget_dest_length(fldname) routine

This routine returns the maximum length of the data buffer that is associated with **fldname**.

Arguments	I/O	Description
char *fldname	Input	Name of destination field, as defined in **ipload** map

The onstat -j option

The **-j** option of the **onstat** utility provides special information about the status of an **onpload** job. The **-j** option provides an interactive mode that is analogous to **onstat -i**.

For information about **onstat -i** and how to use the interactive mode, refer to the *IBM® Informix® Administrator's Reference*.

- Using the onstat -j option
  When **onpload** starts, it writes a series of messages to **stdout** or to a log file.

Related concepts:
Threads that the onpload utility uses
Related reference:
The onstat options for onpload

Using the onstat -j option

When **onpload** starts, it writes a series of messages to **stdout** or to a log file.

The following lines show a typical **onpload** log file:

```
Mon Jul 24 16:11:30 1995

SHMBASE0x4400000
CLIENTNUM  0x49010000
Session ID 1

Load Database-> cnv001
Load Table  -> cnv001a
Load File-> testrec.dat
Record Mapping  -> cnv001a

Database Load Completed -- Processed 50 Records
Records Inserted-> 50
Detected Errors--> 0
Engine Rejected--> 0

Mon Jul 24 16:11:37 1995
```

The two lines that start with SHMBASE and CLIENTNUM provide the information that you need to locate shared memory for an instance of **onpload**. The **oninit** process has similar values stored in the **$onconfig** file. When you use **onstat** to gather information about the **oninit** process, **onstat** uses information from **$INFORMIXDIR/etc/$onconfig** to locate shared memory. When you use **onstat** to gather information about **onpload**, you must give **onstat** the name of a file that contains SHMBASE and CLIENTNUM information.

Typically the file that contains the SHMBASE and CLIENTNUM information is the log file. For example, if the **onpload** log file is **/tmp/cnv001a.log**, you can enter the following command:

```
onstat -j /tmp/cnv001a.log
```

The previous command causes **onstat** to attach to **onpload** shared memory and to enter interactive mode. You can then enter ? or any other bogus request to see a usage message displayed. An example follows:

```
onstat> ?
Interactive Mode: One command per line, and - are optional.
-rzrepeat option every n seconds (default: 5) and
zero profile counts
  MT COMMANDS:
allPrint all MT information
athPrint all threads
waiPrint waiting threads
actPrint active threads
reaPrint ready threads
slePrint all sleeping threads
spiprint spin locks with long spins
schprint VP scheduler statistics
lmxPrint all locked mutexes
wmxPrint all mutexes with waiters
conPrint conditions with waiters
stk <tid>Dump the stack of a specified thread
gloPrint MT global information
mem <pool name|session id>print pool statistics.
segPrint memory segment statistics.
rbmprint block map for resident segment
nbmprint block map for non-resident segments
afr <pool name|session id> Print allocated poolfragments.
ffr <pool name|session id> Print free pool fragments.
ufr <pool name|session id> Print pool usage breakdown
iovPrint disk IO statistics by vp
iofPrint disk IO statistics by chunk/file
ioqPrint disk IO statistics by queue
iogPrint AIO global information
iobPrint big buffer usage by IO VP class
stsPrint max and current stack sizes
qstprint queue statistics
wstprint thread wait statistics
jalPrint all Pload information
jctPrint Pload control table
jpaPrint Pload program arguments
jtaPrint Pload thread array
jmqPrint Pload message queues, jms for summary only
onstat>
```

Most of the options are the same as those that you use to gather information about the database server, with the following exceptions:

```
jalPrint all Pload information
jctPrint Pload control table
jpaPrint Pload program arguments
jtaPrint Pload thread array
jmqPrint Pload message queues, jms for summary only
```

These options apply only to **onpload**. You can use **onstat -j** to check the status of a thread, locate the VP and its PID, and then attach a debugger to a particular thread. The options for **onstat** that do not apply to **onpload** are not available (for example, `-g ses`).

The HPL log-file and pop-up messages

This section provides explanatory notes and corrective actions for unnumbered messages that print in the High-Performance Loader (HPL) log file. The section also includes information specific to messages that are returned to standard output or appear in a pop-up dialog box (depending on the way you started **onpload**).

For more information about error messages and corrective actions, use the **finderr** (UNIX) or **Informix Error Messages** (Windows) utility.

Several HPL error messages refer to the errno.h file, which is located in the following directories:

- /usr/include/errno.h in UNIX
- errno.h, windsock.h, and winsock2.h in the include subdirectory for Microsoft Visual C++.

A few of the messages included here might require you to contact Technical Support. Such messages are rarely, if ever, seen at customer locations.

For information about how to view the log file and some guidance on how and when you might want to read it, see "Viewing Log Files" in chapter 14.

Related reference:
View the status of a load job or unload job

How HPL logfile messages are ordered

The HPL log-file messages appear in this section in alphabetical order, sorted with the following additional rules:

- The time stamp that precedes each message is ignored.
- Letter case in alphabetization is ignored.
- File, record, database server, and table names are ignored.

- Error numbers are ignored.
- Spaces are ignored.
- Quotation marks are ignored.
- The word *the* is ignored if it is the first word in the message.

A cause and suggested corrective action for a message or group of messages follows the message text.

A section that lists pop-up messages (or messages that are returned to standard error) appears after the log-file message sections. Messages in this section are arranged according to the same rules that apply to log-file messages.

HPL logfile message categories

Four general categories of messages can be defined, although some messages fall into more than one category:

- Routine information
- Assertion-failed messages
- Administrative action needed
- Fatal error detected

The assertion-failed messages reflect their traditional use by technical staff to assist in troubleshooting. The information that they report often falls into the category of *unexpected events* that might or might not develop into problems caught by other error codes. Moreover, the messages are terse and often technical. They might report on one or two isolated statistics and not provide an overall picture.

When technical staff investigate a problem, this information can suggest to them possible research paths. However, you might find that the information has little or no application when it is taken out of this context, or when processing proceeds normally.

The HPL log-file messages

- Blob conversion error occurred on record record_num
- Cannot access database table table_name: SQL error error_num
- Cannot allocate shared memory
- Cannot allocate TLI memory for operating_system structure
- Cannot bind socket connection: errno= operating-system_error_num
- Cannot bind TLI connection: t_errno= t_error_num
- Cannot configure driver driver_name
- Cannot connect to message server: Socket error = UNIX_error_num
- Cannot connect to message server: TLI erro r= t_error_num, TLI event = t_event_num, errno = error_num
- Cannot connect to server_name: SQL error error_num, ISAM error error_num
- Cannot connect worker to server data stream
- Cannot disable table_name object constraints: SQL error error_num, ISAM error error_num
- Cannot disable primary-key constraint. Child-table references exist
- Cannot express load to logged table on HDR server server_name
- Cannot filter indexes for table table_name: SQL error error_num, ISAM error error_num
- Cannot find the shared library path in the plconfig file. Using the shared library from the default location library_location
- Cannot find the user-defined function user_func_name in the shared library: error error_num
- Cannot get systable info for table table_name: SQL error error_num, ISAM error error_num
- Cannot initialize shared library handling
- Cannot load code-set conversion file from file_name to file_name
- Cannot load mapping definitions
- Cannot load the shared library library_location
  Shared library load failed with error message *error_message*.
- Cannot locate delimiter in data file
- Cannot open
- Cannot open simple large object file: file_name, simple large object not loaded
- Cannot open database database_name: SQL error error_num, ISAM error error_num
- Cannot open file file_name: error number operating-system_error_num
- Cannot open TCP connection for server_name: errno operating-system_error_num
- Cannot perform express mode load on table with pseudo rowid
- Cannot perform express-mode load with rowsize = row_length > page_size
- Cannot read file file_name: AIO error code operating-system_error_num
- Cannot re-enable all objects: num_violations violations detected
  Check for violations in violations table *table_name* and diagnostics table *table_name*.
- Cannot reorder query statement to align simple large objects or Ext Types
- Cannot reorder query statement to align blobs
- Cannot set mode of table_name objects from current_mode to final_mode mode: SQL error error_num, ISAM error error_num
- Cannot start violations table for table_name: SQL error error_num, ISAM error error_num
- Cannot stop violations table for table_name: SQL error error_num, ISAM error error_num
- Cannot unload to multiple devices when the given query cannot be executed in parallel
- Cannot write file file_name: AIO error code operating-system_error_num
- Code-set conversion overflow
- Conversion of onpload database failed due to error error_num
- Conversion of onpload database failed due to error error_num, run as user informix
- Custom conversion function function_name not found in shared library
- Discarded num_bytes null bytes from end of tape device device_name
- Environment variable variable_name expansion would overflow string

- Error accepting socket connection: errno = operating-system_error_num
- Error accessing file_name
- Error accessing format: SQL error error_num, ISAM error error_num
- Error accessing map map_name: SQL error error_num, ISAMerror error_num
- Error accessing sysmaster: SQL error error_num, ISAMerror error_num
- Error accessing table table_name: SQL error error_num, ISAM error error_num
- Error: AIO buffer size buffer_size is less than required minimum size size
- Error error_num closing current database
- Error operating-system_error_num closing file file_name
- Error error_num converting record field field_name to column column_name
- Error declaring cursor: could not get table info
- Error declaring cursor: SQL error error_num, ISAM error error_num
- Error describing unload query query_name: SQL error error_num, ISAM error error_num
- Error error_num initializing backend connection
- Error inserting into table table_name: SQL error error_num, ISAM error error_num
- Error listening for socket connection: t_errno = t_error_num errno = operating-system_error_num
- Error listening for TLI connection: t_errno = t_error_num errno = UNIX_error_num
- Error error_num on record record_num converting column column_name to record field field_name
- Error occurred on record %d reading pipe %s
- Error on close of server load session: SQL error error_num, ISAM error error_num
- Error opening cursor: SQL Error error_num
- Error preparing query: SQL error error_num
- Error preparing statement statement_name: SQL error error_num, ISAM error error_num
- Error preparing unload query query_name: SQL error error_num, ISAM error error_num
- Error error_num reading message queue
- Error operating-system_error_num reading TLI/socket connection
- Error error_num setting isolation level
- Error error_num writing message on message queue
- Error operating-system_error_num writing TLI/socket connection
- Error: Stream buffer size buffer_size is less than required minimum size size
- Exhausted all attempts to allocate shared-memory key.
- Fatal error: cannot execute pipe_name
- Fatal error: cannot load X resource
- Fatal error creating server load session: error error_num
- Fatal error getting stream buffer from server
- Fatal error in server row processing: SQL error error_num, ISAM error error_num
- File type device file file_name is not a regular (disk) file
- Got Interrupt: Shutting down
- Internal error: Cannot initialize AIO library
- Internal error: Cannot send message
- Internal error: error_num. Contact Tech Support
- Internal error: invalid message type error_num
- Internal error error_num reading queue
- Invalid count detected, might be due to abnormal BE shutdown
- Invalid code-set character: Cannot convert
- Invalid HEXASCII simple large object or extended type representation on record record_num
- Invalid HEXASCII simple large object representation in fieldname, record record_num
- Invalid project name project_name entered
- Invalid reject count detected, might be due to abnormal BE shutdown. Using last known reject count and proceeding
- Invalid session ID id_number
- Invalid tape header expecting -> tape_name
- Map map_name type is not a load map
- Method not supported by current driver
- MT cannot bind to vpid
- MT internal failure
- MT failure putting CPU online
- No insert permission on table table_name
- No mapping to simple large object field field_name
- onpload must run on the host host_name that contains the target database
- onpload terminated by signal
- Pipe type device file file_name is not a regular file
- Pload cannot reorder queries having expressions/aggregates and blobs/udts in the same select list
  Reorder the select list in the query in the following order: 1. non-blob non-udt columns 2. inrow udts in the case of fixed format 3. other blob/udt columns
- Query contains unmapped simple large object column column_name: Cannot proceed
- Query for unload is not a select query.
- Record is too long to process: recnum record_num, length record_length, bufsize buffer_size
- Server interface error; expected num_input but got num_received instead
- SQL error error_num executing statement statement_name
- Simple large object or extended type conversion error occurred on record record_num
- Start record record_num is greater than number of records total_num read from input file_name
- Successfully loaded the shared library library_location
- Table table_name will be read-only until level-0 archive
- Tables with BLOBS cannot be loaded in High Performance Mode
- Tables with BLOBS or extended types cannot be loaded in Express mode
- Tables with simple large objects or extended types cannot be processed with no conversion (-fn)
- Tape header is larger than I/O buffer: tape header_length, I/O buffer_size
- Tape type device file file_name is not a character-special or block-special file
- There is no mapping to column column_name, which cannot accept null values
- Unable to load locale categories for locale locale_name: error error_num
- Unload query select item for the query_item expression needs to be assigned a name

- Write/read to/from tape until end of device
- Write to device (tape or pipe) device_name failed; no space left on device. AIO error error_num

Blob conversion error occurred on record *record_num*

Explanation

The SQLBYTE simple large object data could not be converted to HEXASCII, or the SQLTEXT simple large object data has invalid character data (characters not in the code set).

User response

Remove the invalid characters from the input data.

Cannot access database table *table_name*: SQL error *error_num*

Explanation

The target database table cannot be accessed.

User response

For more information, use the **finderr** or **Informix Error Messages** utility.

Cannot allocate shared memory

Explanation

A memory allocation error occurred. Probably the system is out of virtual shared memory.

User response

Run **onpload** again when fewer users are on the system.

For UNIX, increase the amount of available shared memory with the UNIX kernel configuration.

For Windows, reduce the number of applications running concurrently.

Cannot allocate TLI memory for *operating_system* structure

Explanation

System memory cannot be allocated for communications. This situation should only happen if all system resources are consumed.

User response

Note the circumstances and contact Technical Support.

Cannot bind socket connection: errno= *operating-system_error_num*

Explanation

A TCP socket cannot be opened.

User response

See your errno.h file.

Cannot bind TLI connection: t_errno= *t_error_num*

Explanation

An error occurred when **onpload** attempted to open a TLI connection.

User response

Check that TLI services are installed on the operating system. See your **tiuser.h** file.

UNIX Only

Cannot configure driver *driver_name*

Explanation

You might be specifying a driver incorrectly. If the Driver Class specification is not **Fixed**, **Cobol**, or **Delimited**, either the **onpload** custom-driver shared library is not in the path name, or the custom-driver shared library is not installed correctly.

User response

For information about building a shared library, see "Rebuilding the shared-library file". Make sure that your driver is configured correctly for **Fixed**, **Cobol**, or **Delimited**.

UNIX Only

Cannot connect to message server: Socket error = *UNIX_error_num*

Explanation

This message is generated by **ipload** when it cannot connect to the **onpload** socket service.

User response

See **/usr/include/errno.h**.

UNIX Only

Cannot connect to message server: TLI erro r= *t_error_num*, TLI event = *t_event_num*, errno = *error_num*

Explanation

An error occurred when **onpload** attempted to open a TLI connection.

User response

Check that TLI services are installed on the operating system. See /usr/include/tiuser.h (*t_error_num*).

UNIX only

Cannot connect to *server_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The target database server cannot be opened.

User response

For more information, use the **finderr** or **Informix Error Messages** utility.

Cannot connect worker to server data stream

Explanation

A possible permissions problem exists for **onpload** or **oninit**.

User response

Note the circumstances and contact Technical Support.

Cannot disable *table_name* object constraints: SQL error *error_num*, ISAM error *error_num*

Explanation

The constraint objects are disabled during the load and re-enabled after the load. An error occurred when **onpload** attempted to disable the constraint objects.

User response

For more information, use the **finderr** or **Informix Error Messages** utility.

Cannot disable primary-key constraint. Child-table references exist

Explanation

You attempted to use express mode to load a table that has child-table records that refer to it. The express mode does not support this condition. (The **onpload** utility cannot disable the primary key constraint when child-table records refer to the load table.)

User response

Perform the load in deluxe mode or remove the constraint in question.

Cannot express load to logged table on HDR server *server_name*

Explanation

You attempted to use express mode to load an HDR replicated table. The express mode does not support this condition.

User response

Perform the load in deluxe mode.

Cannot filter indexes for table *table_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The index objects are set to filtering mode during the load and re-enabled after the load. An error occurred when **onpload** attempted to set the indexes objects to filtering mode.

User response

For more information, use the **finderr** or **Informix Error Messages** utility.

Cannot find the shared library path in the plconfig file. Using the shared library from the default location *library_location*

Explanation

The ipldd11a.so shared library path is not set in the plconfig file.

User response

If the default location is not correct, set the correct shared library path in the plconfig file by using the HPL_DYNAMIC_LIB_PATH configuration parameter.

Cannot find the user-defined function *user_func_name* in the shared library: error *error_num*

Explanation

The **onpload** process could not find the required user-defined function in the shared library.

User response

Restart the **onpload** load or unload with the correct shared library or function name in the **pload** job definition.

Cannot get systable info for table *table_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

Cannot access the **systable** table to get dictionary information for the indicated table.

User response

For more information, use the **finderr** or **Informix Error Messages** utility.

Cannot initialize shared library handling

Explanation

The **pload** utility cannot start because it failed to initialize the shared library-handling functionality.

User response

Ensure that the computer has the proper resources and that the shared library has been built properly.

Cannot load code-set conversion file from *file_name* to *file_name*

Explanation

The data type for the load file is different from the data type for the database server. The code set does not exist in the $INFORMIXDIR/gls/cv*x* or %INFORMIXDIR%\gls\cv*x* directory where *x* is the version number of the cv subdirectory.

User response

Check that the file exists. Check the file for permissions.

Cannot load mapping definitions

Explanation

A memory-allocation error or database-integrity error occurred when **onpload** accessed the onpload database.

User response

Use **oncheck** to check the **maps**, **mapitem**, **mapoption**, **formats**, and **formatitem** tables for consistency. If the tables are consistent, a referential integrity problem between the map and the format the map references might exist. If the problems persist, contact Technical Support.

Cannot load the shared library *library_location* Shared library load failed with error message *error_message*.

Explanation

The **onpload** utility could not load the shared library from the *library_location* path and failed with *error_message*.

User response

For more information, use the **finderr** or **Informix Messages** utility. Correct the problem and reload the shared library.

Cannot locate delimiter in data file

Explanation

No delimiter is found when **onpload** scans for an end-of-record delimiter in the load data.

User response

Check that the end-of-record delimiter specification is correct, or that you have the correct data file. Note differences in the end-of-line characters between UNIX and Windows.

Cannot open

Explanation

An internal error occurred when **onpload** attempted to open the load or unload file.

User response

Note the circumstances and contact Technical Support.

Cannot open simple large object file: *file_name*, simple large object not loaded

Explanation

The record references a file name that should contain a simple large object, but the file cannot be located.

User response

Check that the simple-large-object file exists.

Cannot open database *database_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The target database cannot be opened.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Cannot open file *file_name*: error number *operating-system_error_num*

Explanation

The file cannot be opened.

User response

See your errno.h file.

Cannot open TCP connection for *server_name*: errno *operating-system_error_num*

Explanation

A TCP socket cannot be opened.

User response

See your errno.h file.

Cannot perform express mode load on table with pseudo rowid

Explanation

The load table is fragmented by row ID. The express mode does not support this condition.

User response

Perform the load in deluxe mode.

Cannot perform express-mode load with rowsize = *row_length* > *page_size*

Explanation

The table-row size exceeds page size. The express mode does not support this condition.

User response

Perform the load in deluxe mode.

Cannot read file *file_name*: AIO error code *operating-system_error_num*

Explanation

The load file cannot be accessed. This error might result from operating-system limitations; the **onpload** utility cannot load successfully from a file (on disk) that is longer than 2 GB.

User response

See your errno.h file.

Cannot re-enable all objects: *num_violations* violations detected Check for violations in violations table *table_name* and diagnostics table *table_name*.

Explanation

Data loaded by **onpload** violates the object constraints specified for the table. The records that violate the object constraints have been placed in the **violations** table, and the reason code for each violation is listed in the **diagnostics** table.

User response

Review the information in the violations and **diagnostics** tables.

Cannot reorder query statement to align simple large objects or Ext Types

Explanation

The unload query does not contain a FROM clause.

User response

Rewrite the query so that it contains a FROM clause.

Cannot reorder query statement to align blobs

Explanation

The unload query does not contain a FROM clause.

User response

Rewrite the query so that it contains a FROM clause.

Cannot set mode of *table_name* objects from *current_mode* to *final_mode* mode: SQL error *error_num*, ISAM error *error_num*

Explanation

The object constraints are disabled during the load and re-enabled after the load. An error occurred when **onpload** attempted to reset object constraints back to their original state.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Cannot start violations table for *table_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

An error occurred when **onpload** attempted to set up the violations table for the load table.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Cannot stop violations table for *table_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

If a violations table exists on the load table, violations can be turned off during the load. An error occurred when **onpload** attempted to turn off violations detection.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Cannot unload to multiple devices when the given query cannot be executed in parallel

Explanation

The server determined that the query cannot be run in parallel.

User response

Remove non-parallel aspects of the query, such as non-parallel UDRs, or unload to a single device.

Cannot write file *file_name*: AIO error code *operating-system_error_num*

Explanation

The unload file cannot be accessed.

User response

See your errno.h file.

Code-set conversion overflow

Explanation

The code-set conversion caused the number of bytes in the BYTE and TEXT data to expand or contract when **onpload** unloaded the data into a fixed-format record. The **onpload** utility cannot update the BYTE and TEXT data tag in the record that specifies the length of the BYTE and TEXT data at this stage.

User response

To unload this data, use a delimited format.

Conversion of onpload database failed due to error *error_num*

Explanation

The **onpload** tried to convert the old database when **onpload** ran for the first time on the new database server. This conversion failed because of the error referenced in the error message.

User response

For more information, use the **finderr** or **Informix Messages** utility. Resolve this error before you rerun **onpload**.

Conversion of onpload database failed due to error *error_num*, run as user informix

Explanation

Database conversion fails because the current user running **onpload** does not have sufficient privileges to convert the onpload database.

User response

Run the **onpload** job as user **informix** once.

Custom conversion function *function_name* not found in shared library

Explanation

The custom function specified in a map option was not located in ipldd11a.so. The shared library extension is platform-specific; for example, the .so extension is specific for Solaris and is probably different on other platforms.

User response

For information about how to configure the custom function library, see "Custom-conversion functions".

UNIX Only

Discarded *num_bytes* null bytes from end of tape device *device_name*

Explanation

The tape data is not blocked in a multiple of the record size, so that the last block of data contained bytes that are discarded. This situation occurs on devices with stream cartridges that allow writing to the device only in whole blocks.

User response

If necessary, manually enter the discarded data.

Environment variable *variable_name* expansion would overflow string

Explanation

A mapping option specifies an environment variable as the default value, but expansion of the environment variable requires more space than allocated to the column.

User response

Use a shorter default value, or expand the length of the column.

Error accepting socket connection: errno = *operating-system_error_num*

Explanation

A TCP socket cannot be accessed.

User response

See your errno.h file.

Error accessing *file_name*

Explanation

An error occurred when **onpload** attempted to open the load or unload file.

User response

Check that the file exists. Check the file for permissions.

Error accessing format: SQL error *error_num*, ISAM error *error_num*

Explanation

An integrity problem exists in the onpload database. The format for the map does not exist, or a problem exists with the **format** or **formatitem** table.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error accessing map *map_name*: SQL error *error_num*, ISAMerror *error_num*

Explanation

The requested map for the load or unload does not exist, or a problem exists with the onpload database.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error accessing sysmaster: SQL error *error_num*, ISAMerror *error_num*

Explanation

An access error occurred on the sysmaster database on the target server where **onpload** attempted to perform the load or unload job.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error accessing table *table_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The target database table cannot be accessed.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error: AIO buffer size *buffer_size* is less than required minimum size *size*

Explanation

AIO buffer size is less than required size.

User response

Increase the specified buffer size in the plconfig file.

Error *error_num* closing current database

Explanation

A server error occurred when **onpload** closed the onpload or target database.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error *operating-system_error_num* closing file *file_name*

Explanation

An error occurred when **onpload** closed the load or unload file.

User response

See your errno.h file

Error *error_num* converting record field *field_name* to column *column_name*

Explanation

A conversion error occurred when **onpload** attempted to convert the record data to the database column type.

User response

For more information, use the **finderr** or **Informix Messages** utility. If the load map indicates that the data field is mapped to the correct column, check that the supplied data is valid.

Error declaring cursor: could not get table info

Explanation

Cannot access information about the load table.

User response

Check the validity of the table in the target database.

Error declaring cursor: SQL error *error_num*, ISAM error *error_num*

Explanation

The **onpload** utility is unable to use the autogenerated formats and maps to create entries in a table in the onpload database.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error describing unload query *query_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The unload query cannot be processed.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error *error_num* initializing backend connection

Explanation

An internal error occurred in onpload. It is possible that the server went down.

User response

Note the circumstances and contact Technical Support.

Error inserting into table *table_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The **onpload** utility is unable to use the autogenerated formats and maps to create entries in a table in the onpload database.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error listening for socket connection: t_errno = *t_error_num* errno = *operating-system_error_num*

Explanation

An error occurred listening on a Socket connection.

User response

See your errno.h file.

Error listening for TLI connection: t_errno = *t_error_num* errno = *UNIX_error_num*

Explanation

An error occurred listening on a TLI connection.

User response

See /usr/include/tiuser.h (*t_error_num*).

UNIX Only

Error *error_num* on record *record_num* converting column *column_name* to record field *field_name*

Explanation

A conversion error occurred when **onpload** attempted to convert the column data to the record field type.

User response

For more information, use the **finderr** or **Informix Messages** utility. Check the load map to verify that the column is mapped to the correct record field.

Error occurred on record *%d* reading pipe *%s*

Explanation

A conversion error occurred in a record.

User response

No action is required.

Error on close of server load session: SQL error *error_num*, ISAM error *error_num*

Explanation

An internal error occurred in **onpload**. Probably the server went down.

User response

Note the circumstances and contact Technical Support.

Error opening cursor: SQL Error *error_num*

Explanation

An error occurred when **onpload** attempted to set up an insert cursor on the load table.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error preparing query: SQL error *error_num*

Explanation

The unload query cannot be processed.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error preparing statement *statement_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

An internal error occurred when **onpload** attempted to access the onpload database.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error preparing unload query *query_name*: SQL error *error_num*, ISAM error *error_num*

Explanation

The unload query cannot be processed.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error *error_num* reading message queue

Explanation

This critical initialization error probably means that the operating kernel does not have enough shared memory or semaphores configured or that the allocated shared memory was removed.

User response

On UNIX, increase shared memory or semaphores. On Windows, repeat the operation.

If the condition persists, contact Technical Support.

Error *operating-system_error_num* reading *TLI/socket* connection

Explanation

An error occurred reading a socket or TLI connection, based on the type of connection specified in onconfig file.

User response

See your errno.h file.

Error *error_num* setting isolation level

Explanation

An access error occurred when **onpload** attempted to set the isolation level for an unload job.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Error *error_num* writing message on message queue

Explanation

This critical initialization error probably means that the operating system kernel does not have enough shared memory or semaphores configured, or that the allocated shared memory has been removed.

User response

On UNIX, increase shared memory or semaphores. On Windows, repeat the operation.

If the condition persists, contact Technical Support.

Error *operating-system_error_num* writing *TLI/socket* connection

Explanation

An error occurred writing a socket or TLI connection, based on the type of connection specified in onconfig file.

User response

See your errno.h file.

Error: Stream buffer size *buffer_size* is less than required minimum size *size*

Explanation

Stream buffer size is less than required size.

User response

Increase the specified buffer size in the plconfig file.

Exhausted all attempts to allocate shared-memory key.

Explanation

All the shared-memory keys in the key range tried by **onpload** are currently allocated.

User response

Wait until another **onpload** session finishes. If the problem persists, contact Technical Support.

Fatal error: cannot execute *pipe_name*

Explanation

An attempt to run the PIPE type device in the device array failed.

User response

Make sure that the PIPE entry in the device array is a valid, executable program. Pipes are only supported on UNIX.

Fatal error: cannot load X resource

Explanation

This is an internal error.

Fatal error creating server load session: error *error_num*

Explanation

Cannot start the load session with the server.

User response

Note the circumstances and contact Technical Support.

Fatal error getting stream buffer from server

Explanation

An internal error occurred in **onpload**. It is possible that the server went down.

User response

Note the circumstances and contact Technical Support.

Fatal error in server row processing: SQL error *error_num*, ISAM error *error_num*

Explanation

An internal communication problem exists between the server and onpload.

User response

Note the circumstances and contact Technical Support.

File type device file *file_name* is not a regular (disk) file

Explanation

The device array specifies that the file is a disk file, but it is not.

User response

Change the type of the file in the device-array definition, or make sure that the file is a disk file.

Got Interrupt: Shutting down

Explanation

An internal error occurred, or a user sent an interrupt to **onpload**.

User response

If a user did not generate this interrupt, contact Technical Support.

Internal error: Cannot initialize AIO library

Explanation

This critical initialization error probably means that the UNIX kernel does not have enough shared memory or semaphores configured.

User response

Increase shared memory or semaphores. If the condition persists, contact Technical Support.

Internal error: Cannot send message

Explanation

An internal error occurred in onpload. The most likely cause is a lack of shared memory.

User response

Note the circumstances and contact Technical Support.

Internal error: *error_num*. Contact Tech Support

Explanation

A critical internal error occurred.

User response

Note the circumstances and contact Technical Support.

Internal error: invalid message type *error_num*

Explanation

A critical internal error occurred.

User response

Note the circumstances and contact Technical Support.

Internal error *error_num* reading queue

Explanation

This critical initialization error probably means that the operating system kernel does not have enough shared memory or semaphores configured.

User response

On UNIX, increase shared memory or semaphores. On Windows, repeat the operation.

If the condition persists, contact Technical Support.

Invalid count detected, might be due to abnormal BE shutdown

Explanation

The count of rows being loaded became corrupted. This message can appear if a deluxe load job ends with an error.

User response

Contact IBM® Technical Support.

Invalid code-set character: Cannot convert

Explanation

The data being loaded or unloaded has invalid character data.

User response

Make sure that you specified the correct data type on the format definition.

Invalid HEXASCII simple large object or extended type representation on record *record_num*

Explanation

The simple large object or extended type field being loaded was classed as HEXASCII, but the data contains a non-HEXASCII character.

User response

Fix the data.

Invalid HEXASCII simple large object representation in *fieldname*, record *record_num*

Explanation

The simple large object data field being loaded was classed as HEXASCII, but the data contains a non-HEXASCII character.

User response

Fix the data.

Invalid project name *project_name* entered

Explanation

Incorrect project name was specified for onpload.

User response

Check the given project name and restart onpload.

Invalid reject count detected, might be due to abnormal BE shutdown. Using last known reject count and proceeding

Explanation

The count of rows being loaded became corrupted. This message can appear if a deluxe load job ends with an error.

User response

Contact IBM® Technical Support.

Invalid session ID *id_number*

Explanation

The command line specified an invalid session ID for the job to run. An entry for the entered session ID must exist in the **session** table of the onpload database in order to run the job.

User response

Make sure the session ID on the command line matches the correct session ID in the **session** table.

Invalid tape header expecting -> *tape_name*

Explanation

Incorrect tape was mounted.

User response

Mount the correct tape.

Map *map_name* type is not a load map

Explanation

Incorrect map was specified to **onpload**. You must use a load map for a load job and an unload map for an unload job.

User response

Verify that you are using the correct map type.

Method not supported by current driver

Explanation

An internal error occurred in **onpload**.

User response

Note the circumstances and contact Technical Support.

MT cannot bind to vpid

Explanation

This critical initialization error probably means that the operating system kernel does not have enough shared memory or semaphores configured.

User response

On UNIX, increase shared memory or semaphores. On Windows, repeat the operation.

If the condition persists, contact Technical Support.

MT internal failure

Explanation

This critical initialization error probably means that the operating system kernel does not have enough shared memory or semaphores configured.

User response

On UNIX, increase shared memory or semaphores. On Windows, repeat the operation.

If the condition persists, contact Technical Support.

MT failure putting CPU online

Explanation

This critical initialization error probably means that the UNIX kernel does not have enough shared memory or semaphores configured.

User response

Increase shared memory or semaphores. If the condition persists, contact Technical Support.

No insert permission on table *table_name*

Explanation

You cannot load the indicated table because the DBA has not granted permission for you to do so.

User response

Make sure that you have insert permissions on the table.

No mapping to simple large object field *field_name*

Explanation

The record format specifies a simple large object or extended type, but no column from the query is mapped to the record field.

User response

Map a column to the field, or remove the field from the record format.

onpload must run on the host *host_name* that contains the target database

Explanation

User tried to run **onpload** on a host computer other than the one that has the target database.

User response

Run **onpload** on the host specified in the error message.

onpload terminated by signal

Explanation

Either an internal error occurred or a user sent **onpload** a termination signal.

User response

If the signal is not SIGKILL, SIGTERM, or SIGQUIT, note the circumstances and contact Technical Support.

UNIX Only

Pipe type device file *file_name* is not a regular file

Explanation

The device array specifies that the file is a pipe (executable program) file, but it is not.

User response

Change the type of the file in the device-array definition, or make sure that the file is an executable disk file. Pipes are only supported on UNIX.

Pload cannot reorder queries having expressions/aggregates and blobs/udts in the same select list Reorder the select list in the query in the following order: 1. non-blob non-udt columns 2. inrow udts in the case of fixed format 3. other blob/udt columns

Explanation

The **onpload** utility requires that the special columns (simple large column and user-defined types) appear at the end of the select list. The **onpload** utility will reorder simple SELECT statements but is unable to reorder the select list because of expressions, aggregates, or both.

User response

Reorder select columns manually as explained in error message. Alternatively, you can remove aggregates and expressions from the select list by selecting the columns into a temporary table and then unloading them from that table.

Query contains unmapped simple large object column *column_name*: Cannot proceed

Explanation

The unload query is extracting a simple large object or extended type column that is not mapped to the record field.

User response

Modify the unload query so that it does not reference the simple large object or extended type column, or map it to a field in the record format.

Query for unload is not a select query.

Explanation

The unload query does not contain a SELECT statement.

User response

Modify the query so that it contains a SELECT statement.

Record is too long to process: recnum *record_num*, length *record_length*, bufsize *buffer_size*

Explanation

The record size exceeds the size of the **onpload** buffers (AIOBUFSIZE). This error can occur when a delimited record contains simple-large-objects or extended types, and a format specification for a field is missing, which causes a simple large object or an extended type to be treated as a regular field.

User response

Increase the size of AIOBUFSIZE for this record, or check that the format specification for the field matches the input file.

Server interface error; expected *num_input* but got *num_received* instead

Explanation

An onpload/server interface error occurred.

User response

Note the circumstances and contact Technical Support.

SQL error *error_num* executing statement *statement_name*

Explanation

An internal error occurred when **onpload** accessed the onpload database.

User response

For more information, use the **finderr** or **Informix Messages** utility.

Simple large object or extended type conversion error occurred on record *record_num*

Explanation

The SQLBYTE simple large object or extended type data could not be converted to HEXASCII, or the SQLTEXT simple large object or extended type has invalid character data (characters not in the code set).

User response

Remove the invalid characters from the input data.

Start record *record_num* is greater than number of records *total_num* read from input *file_name*

Explanation

A start record was specified for the load, but fewer records are in the input file than the indicated number of records to skip.

User response

Specify the start-record number again.

Successfully loaded the shared library *library_location*

Explanation

The shared library was loaded successfully.

User response

Check the path to verify that the correct library is being used. If the path is incorrect, edit the HPL_DYNAMIC_LIB_PATH configuration parameter in the plconfig file to supply the correct path.

Table *table_name* will be read-only until level-0 archive

Explanation

After an express-mode load, a level-0 archive is needed to make the table available for update.

User response

Perform a level-0 archive.

Tables with BLOBS cannot be loaded in High Performance Mode

Explanation

You attempted to use express mode to load a table that contains BLOB data types. The express mode does not support this condition.

User response

Perform the load in deluxe mode.

Tables with BLOBS or extended types cannot be loaded in Express® mode

Explanation

You attempted to use express mode to load a table that contains BLOB or extended data types. The express mode does not support this condition.

User response

Perform the load in deluxe mode.

Tables with simple large objects or extended types cannot be processed with no conversion (-fn)

Explanation

You attempted a no-conversion load on a table with simple large object or extended type columns. This action is not allowed.

User response

Remove the no-conversion specification, and run the job again.

Tape header is larger than I/O buffer: tape *header_length*, I/O *buffer_size*

Explanation

A tape header size is too large to fit into a memory buffer.

User response

Increase AIOBUFSIZE in PLCONFIG to at least the value specified for tape I/O.

Tape type device file *file_name* is not a character-special or block-special file

Explanation

The device array specifies that the file is a tape device, but it is not.

User response

Change the type of the file in the device-array definition, or make sure that the file is a tape device.

There is no mapping to column *column_name*, which cannot accept null values

Explanation

The specified column has a NOT NULL constraint, but in the definition of the load map, no field is mapped to the column.

User response

Correct the load map or drop the NOT NULL constraint.

Unable to load locale categories for locale *locale_name*: error *error_num*

Explanation

The GLS locale specified in CLIENT_LOCALE or DB_LOCALE cannot be loaded, or if these variables are not set, the GLS file cannot be loaded.

User response

Check the $INFORMIXDIR/gls or %INFORMIXDIR%\gls directory to ensure that the locale files are present.

Unload query select item for the *query_item* expression needs to be assigned a name

Explanation

A SELECT statement contains a column name that might not be unique.

User response

Modify the SELECT statement to contain a name for each column expression. For example:

```
SELECT Max(I) Mcol FROM table x
```

Write/read to/from tape until end of device

Explanation

The **onpload** command-line option **-Z** enabled write to and read from the tape until the end of the device.

User response

No action is required.

Write to device (tape or pipe) *device_name* failed; no space left on device. AIO error *error_num*

Explanation

The write to tape is failing on a tape device due to lack of space.

User response

Increase the space on the tape device or replace the device and restart the load or unload job.

HPL logfile pop-up messages

Cannot attach to server shared memory

Explanation

If the server is on, a permissions problem exists.

User response

On UNIX, check that the following permissions and ownership of onpload are set:

```
-rwsr-sr-x 1 informix informix
```

On Windows, check the permissions of the user running **onpload**.

Cannot create shared-memory message queue: error *error_num*

Explanation

A critical initialization error occurred. Probably the UNIX kernel does not have enough shared memory or semaphores configured.

User response

Increase shared memory or semaphores. If the condition persists, contact Technical Support.

UNIX Only

Cannot create shared-memory pool: errno *UNIX_error_num*

Explanation

The operating system shared-memory system cannot be accessed.

User response

See your errno.h file.

UNIX Only

Related concepts:
Overview of the onpladm utility

Cannot initialize multithreaded library

Explanation

A critical initialization error occurred. Probably the UNIX kernel does not have enough shared memory or semaphores configured.

User response

Increase shared memory or semaphores. If the condition persists, contact Technical Support.

UNIX Only

Cannot initialize shared memory: errno *operating-system_error_num*

Explanation

The operating system shared-memory system cannot be accessed.

User response

See your errno.h file.

Cannot load X resource

Explanation

The **ipload** utility attempted to display a full-color splash screen image but another process was already using the resources needed for color display.

User response

Use the **ipload -n** option, which does not display a splash screen.

UNIX Only

Cannot open. Enter (r)etry, (c)ontinue, (q)uit job when ready

Explanation

An internal error occurred when **onpload** attempted to open the load or unload file.

User response

Press r to try to access the load or unload file again. Press c to skip the file indicated and continue to process the rest of the files. Press q to stop the job.

Cannot open log file *log_file_name*.

Explanation

The log file for the job cannot be opened.

User response

See your errno.h file.

Cannot start I/O. Enter (r)etry, (c)ontinue, (q)uit job when ready

Explanation

An internal error occurred when **onpload** attempted to open the load or unload file.

User response

Press r to try to access the load or unload file again. Press c to skip the file indicated and continue to process the rest of the files. Press q to stop the job.

Fatal error: shared memory will conflict with server

Explanation

The shared-memory segment allocated to onpload is located below the shared memory segment of the server, and the size needed to run the job would cause the onpload shared memory to overlap the shared memory of the server.

User response

Reduce the size and number of buffers allocated to onpload on $INFORMIXDIR/etc/plconfig or %INFORMIXDIR%\etc\plconfig, or increase the start address for the shared memory location of the server.

Incorrect database version. Make sure that it is upgraded properly

Explanation

Upgrade of onpload database failed.

User response

Look in **$INFORMIXDIR/etc/conpload.out** for information about conversion errors.

Press ‘r’ when ready, ‘c’ to shutdown device or ‘q’ to quit

Explanation

The tape device is full.

User response

Mount a new tape device and press r to continue the load or unload process. Press c to stop the load or unload process on the current drive. Press q to stop the load or unload process.

Set the shared library path as an absolute path in the plconfig file

Explanation

The full absolute path of the ipldd11a.so shared library is not set.

User response

Set the HPL_DYNAMIC_LIB_PATH configuration parameter to an absolute path in the plconfig file and restart the job.

Tables with blobs cannot be loaded in High-Performance Mode

Explanation

The express mode cannot load tables that contain BLOB data types.

User response

Use Deluxe mode.

Write error. Enter (r)etry, (c)ontinue, (q)uit job when ready

Explanation

An internal error occurred when **onpload** attempted to open the load or unload file.

User response

Press r to try to access the load or unload file again. Press c to skip the file indicated and continue to process the rest of the files. Press q to stop the job.

Custom drivers

If your operating system supports dynamic linking of libraries, you can use a custom driver to extend the functionality of the High-Performance Loader (HPL) to support different file types or access mechanisms.

For example, you could implement a custom interface to load data from a structured file, a high-speed communications link, or another program that generates data to be stored in the database.

The **onpload** utility accesses the custom code through the driver name that you assign to the record-format definition. When **onpload** references a record format, the driver that the record format specifies is examined. If the driver name does not match one of the standard drivers (Fixed, COBOL, Delimited), **onpload** looks into the custom-driver function table to find the custom driver.

The custom-driver code reads data into buffers during a load job and writes out buffers during an unload job. By following the coding procedure discussed in this section, you can use the parallel I/O facilities of the HPL to manipulate data buffers, or you can use a custom driver to replace the HPL I/O facilities with your own I/O functionality.

- Add a custom driver to the onpload utility
- Connect your code to onpload at run time
  When **onpload** determines that a custom driver is required to read or write data for a given record format, it calls the function **pl_get_user_method()**.
- An example of a custom driver
  The example in this topic shows how to code a custom driver that completely takes over the open, close, read, and write responsibility. The example illustrates the form of a driver and the necessary initialization, registration, and mechanism of the driver. The coding of user- specific functionality is not represented.
- Available driver methods
- Available API support functions
  This section describes the API support functions that you can use with custom drivers.

Add a custom driver to the onpload utility

To add a custom driver to **onpload**, you must perform the following tasks:

- Add the custom driver to the **onpload** database.
- Prepare the code for the custom driver.
  For instructions, see Preparing the custom-driver code.
- Build the shared-library file for the custom driver and custom-conversion functions and install the file in the appropriate directory.
  For instructions, see Rebuilding the shared-library file.

- Adding the driver name to the onpload database
  You must add the name of the custom driver to the onpload database so that you can select it when you prepare a load or unload job.
- Preparing the custom-driver code
  A driver is implemented as a set of functions, referred to as *methods*. The methods enable **onpload** to open, close, read, and write data files. You can create a custom driver that adds more complex functionality to data-file handling of **onpload**.
- Rebuilding the shared-library file
  If you use custom drivers or custom-conversion functions, you must rebuild the ipldd11a.SOLIBSUFFIX shared-library file with your custom-code files. (SOLIBSUFFIX is the shared-library suffix for your operating system.)

Adding the driver name to the onpload database

You must add the name of the custom driver to the onpload database so that you can select it when you prepare a load or unload job.

To add a driver name to the onpload database:

1. Choose a name for your custom driver. You can select your own name. This section uses the name *your_custom_driver*.
2. Add the name to the onpload database by choosing Configure > Driver For more information, see Adding a custom-driver name.
3. If you prepare multiple custom drivers, you must choose a name for each driver and add it to the onpload database.

Preparing the custom-driver code

A driver is implemented as a set of functions, referred to as *methods*. The methods enable **onpload** to open, close, read, and write data files. You can create a custom driver that adds more complex functionality to data-file handling of **onpload**.

A custom driver consists of one or more functions that replace the capability of an existing driver method. The custom driver needs to provide all of the methods for a driver, such as OPEN, READ, WRITE, and CLOSE.

To add to the capability of an existing driver method, the custom driver function calls the existing driver method from the custom function before or after any custom processing, as appropriate.

To replace an existing driver method, the custom function provides all processing that is necessary for that function. The custom driver function does not call the existing standard driver functions.

To prepare the custom-driver code, you must prepare the following two files. You can store the files in any convenient directory.

- The *your_custom_driver*.c file contains the functions that provide your user-specific driver functionality. You must provide a function for each driver.
- The plcstdrv.c file tells **onpload** where to find the custom-driver functions.

- Preparing the file that provides the driver functionality
- Preparing the plcstdrv.c file

Preparing the file that provides the driver functionality

To prepare the file that provides the driver functionality:

1. Create a file for the driver functions (for example, *your_custom_driver*.c).
2. Prepare the driver code.
   This section includes an example of driver files, An example of a custom driver. Use this example as a template for building your driver code.

   The driver methods and API functions that you can use are described in Available driver methods and Available API support functions.

Preparing the plcstdrv.c file

To prepare the plcstdrv.c file:

1. Use the following code as a template to create the plcstdrv.c file. You can copy a template for plcstdrv.c from the $INFORMIXDIR/incl/hpl directory.

```
/*******************************************************
 * Start of plcstdrv.c
 */

/* plcstdrv.h is in $INFORMIXDIR/incl/hpl */
#include "plcstdrv.h"

/* Your driver configuration function */
int your_driver_config_function();

(*pl_get_user_method(driver, method)) ()
char *driver;
intmethod;
{
/*
 * your_driver_name is the name of your driver
 */
if (strcmp(driver, "your_driver_name") == 0)
{
/*
 * If onpload is trying to configure the driver,
 * return the function that will handle the
 * initialization.
 */
if (method == PL_MTH_CONFIGURE)
return(your_driver_config_function);
}
/*
 * YYYY is the name of another driver
 * This is how additional custom drivers are configured
 */
if (strcmp(driver, "YYYY") == 0)
{
if (method == PL_MTH_CONFIGURE)
return(YYYY_driver_config_function);
}
}
/***************** end of plcstdrv.c ******************/
```

2. Replace *your_driver_name* with the name of the driver that you chose in step 1 of Adding the driver name to the onpload database.
3. Replace *your_driver_config_function* with the function name of the driver-configuration function that you coded in *your_custom_driver*.c.
4. To add multiple custom drivers, repeat the main **if** statement for each driver.

Rebuilding the shared-library file

If you use custom drivers or custom-conversion functions, you must rebuild the ipldd11a.SOLIBSUFFIX shared-library file with your custom-code files. (SOLIBSUFFIX is the shared-library suffix for your operating system.)

After you rebuild ipldd11a.SOLIBSUFFIX, you must install it in the $INFORMIXDIR/lib directory. To store the shared library in a different location, such as /usr/lib, set the HPL_DYNAMIC_LIB_PATH configuration parameter to the appropriate path in the plconfig file.
Important: For security reasons, you should keep all shared libraries used by the database server in directories under $INFORMIXDIR.
To build the shared-library file:

1. Use the following code as a template to prepare a makefile. You can copy a template for the makefile from the $INFORMIXDIR/incl/hpl directory.

```
#########################################################
# Makefile for building onpload shared library
#

LD = ld

# link flag for building dynamic library, may be different
# for your platform, see the man page for ld, the link
# editor.
LDSOFLAGS = -G

# where to find plcstdrv.h
INCL = $INFORMIXDIR/incl/hpl

# SOLIBSUFFIX is the shared library suffix for your
# platform. For Solaris it is "so"
SOLIBSUFFIX = so

PLCDLIBSO = ipldd11a.$(SOLIBSUFFIX)

.c.o:
$(CC) $(CFLAGS) -I$(INCL) -c $<

#
# plcstdrv.c contains the custom driver table, required
# plcstcnv.c contains the custom function table, required
# your_custom_driver.c contains your custom driver
#routines, optional
# your_custom_conversion.c contains your custom
#conversion functions, optional
#

SOOBJS = plcsrdrv.o plcsrcnv.o your_custom_driver.o
your_custom_function.o

solib: $(SOOBJS)
$(LD) -o $(PLCDLIBSO) $(LDSOFLAGS) $(SOOBJS)

################ end of makefile ######################
```

2. Include the following files in the makefile.

File	Description
plcstdrv.c	Prepares the custom driver tables.
plcstcnv.c	Prepares the custom-conversion function tables.
your_custom_driver.c	Contains your user-specific driver functions
your_custom_functions.c	Contains your user-specific conversion functions

Custom-conversion functions describes the plcstcnv.c and the *your_custom_functions*.c files.

3. Run make by using the makefile.
The makefile builds the shared-library file ipldd11a.SOLIBSUFFIX, where SOLIBSUFFIX is the shared-library suffix for your operating system.

   The HPL uses the same library for both custom-conversion functions and custom drivers, so when you rebuild the library, you must also link the custom-conversion code.

4. Install the shared library in the appropriate path for your operating system.
For example, on Solaris you must install the shared library in the $INFORMIXDIR/lib directory, or in the directory specified by the HPLDYNAMICLIB configuration variable in the plconfig file. You can move ipldd11a.SOLIBSUFFIX into the shared-library directory, or you can use a link. For administrative purposes, a link might be clearer.

5. Set the owner and group of the shared-library files to **informix**. Set the permission bits to 0755 (octal).

Tip: When the libraries are updated, the letter before the decimal (here, the letter a) changes to indicate that the library has changed. If you do not find ipldd11a, look for ipldd11b or ipldd11c.

Connect your code to onpload at run time

When **onpload** determines that a custom driver is required to read or write data for a given record format, it calls the function **pl_get_user_method()**.

The **pl_get_user_method()** function returns the function that the loader should call to perform initial driver configuration before any I/O activity is started. The function that you specify should be a function that you are supplying in your driver. This function should not do any other initialization. The example in the previous section illustrates the coding technique for this initial connection of your driver to onpload.

- Driver initialization
  The **onpload** utility calls the configure function that you returned in **pl_get_user_method()**, expecting this function to configure all driver methods that are to be customized. The configure function must call the **pl_inherit_methods()** function, specifying the class of driver that is appropriate for the data being processed.
- Register driver functions
  The **pl_set_method_function()** registers a passed function to the passed method ID.

Driver initialization

The **onpload** utility calls the configure function that you returned in **pl_get_user_method()**, expecting this function to configure all driver methods that are to be customized. The configure function must call the **pl_inherit_methods()** function, specifying the class of driver that is appropriate for the data being processed.

A driver class can be one of following classes.

Driver class	Description
Fixed	Fixed drivers process data on the assumption that the data is organized as constant length records of the same format. Fields in the record will consistently appear at the same offset in each record.
Delimited	Delimited drivers assume that the record and field boundaries are defined by markers (called delimiters) in the data.
COBOL	The COBOL driver treats data as constant length records in the same manner as the Fixed driver. The distinguishing factor of the COBOL driver is its support for conversion of the ANSI COBOL data types.

After the **pl_inherit_methods()** function is called, you can add additional functions that are called to support open, close, read, and write requirements of the driver. Call **pl_set_method_function()** to tie your driver functions into the **onpload** execution.

Register driver functions

The **pl_set_method_function()** registers a passed function to the passed method ID.

For a description of the method IDs (defined in **$INFORMIXDIR/incl/plcustom/pldriver.h**) applicable to your driver implementation, refer to the Available API support functions.

The following table shows the available methods.

Method	Description
PL_MTH_OPEN	The function called to open the file of interest for the load/unload
PL_MTH_CLOSE	The function called to close the file at the end of the load/unload
PL_MTH_RAWREAD	The function called to get the next block of data from the load file
PL_MTH_RAWWRITE	The function called to write a block of data that is passed to it

You do not need to register a function for any of the methods IDs (although presumably you register at least one, or there is no point in writing the driver).

Use the **pl_driver_inherit()** function to get the standard function for the passed method. For example, to find and run the function currently registered for reading data from the load input, you would code as follows:

```
int my_rawread_function(bufptr, bufsize, bytesread)
char *bufptr;
int  bufsize;
int *bytesread;
{
int (*funcptr)();
int rtn;

funcptr = pl_driver_inherit(PL_MTH_RAWREAD);
rtn = (*funcptr)(bufptr, bufsize, &bytesread);
if (rtn)
return(rtn);  /* error */
  /*
* Now you have a buffer of data in bufptr, of
* size bytesread.  So you can process data in
* this buffer here before it is converted
*/
return(rtn);
}
```

For performance reasons, system calls are discouraged. System calls cause the VP on which the driver thread is running to be blocked for the duration of the system call. This blockage prevents the VP from doing other work, causing **onpload** to run less efficiently.

An example of a custom driver

The example in this topic shows how to code a custom driver that completely takes over the open, close, read, and write responsibility. The example illustrates the form of a driver and the necessary initialization, registration, and mechanism of the driver. The coding of user- specific functionality is not represented.

The plcstdrv.c file

Assume that you chose **MYDRIVER** as the driver name and that you added this name to the onpload database with **ipload**. The plcstdrv.c file is as follows:

```
#include "plcstdrv.h"

int DrConfig();

(*pl_get_user_method(driver, method)) ()
char*driver;
int method;
{
if (strcmp(driver, "customdrv") == 0)
{
if (method == PL_MTH_CONFIGURE)
return (DrConfig);
}

return (0);
}
```

Custom-driver code for MYDRIVER

The following driver code supports **MYDRIVER**:

```
#include <plcstdrv.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/fcntl.h>

extern char *malloc();
externerrno;

static int DrOpen();
static int DrRead();
static int DrWrite();
static int DrClose();

#define CTRLDELIM 0x01/* fake delimiter (CTRL-A) */
#define REALDELIM 0x7c/* real delimiter (|)  */
#define ESCAPSIGN 0x5c/* escape sign(\)  */
#define ENDRECORD 0x0a/* end of record  (\n) */

int fd;

/*-------------------------------------------------------------------
 * DrConfig()
 *
 * Input  : (char *)driver name
 *  (void *)method table
 *
 * Return : PL_RTN_OK
 *
 * Schema : Fills in the driver table
 *-------------------------------------------------------------------*/

int
DrConfig(driver,methodtable)
char*driver;
void*methodtable;
{
pl_inherit_methods("Delimited", methodtable);
pl_set_method_function(methodtable, PL_MTH_OPEN, DrOpen);
pl_set_method_function(methodtable, PL_MTH_RAWREAD, DrRead);
pl_set_method_function(methodtable, PL_MTH_RAWWRITE, DrWrite);
pl_set_method_function(methodtable, PL_MTH_CLOSE, DrClose);
pl_lock_globals();

return PL_RTN_OK;
}

/*-------------------------------------------------------------------
 * DrOpen()
 *
 * Input  : (devicearray *) devdevice array structure
 *
 * Return : PL_RTN_FAILerror
 *  PL_RTN_OK  open succeeded
 *
 * Schema : Open the specific file for that driver thread
 *  Note that the custom driver thread is bound to its
 *  own CPU VP, therefore it is safe to have globals like fd
 *-------------------------------------------------------------------*/
```

```
static int
DrOpen(dev)
devicearray*dev;
{
fd = open(dev->filename, O_RDONLY);
if (fd < 0)
{
return PL_RTN_FAIL;
}

return PL_RTN_OK;
}



/*-----------------------------------------------------------------------
 * DrRead()
 *
 * Input  : (char *) bfoutput buffer to write record to
 *  (int)size size of output buffer
 *  (int *)  countnumber of bytes written to output buffer
 *
 * Return : PL_RTN_FAILerror
 *  PL_RTN_OK returning buffer
 *  PL_RTN_EOFreturning the last buffer, no more data
 *
 *
 * Schema : Reads from input and fill up data buffer provided by the
 *  caller. Here, the caller expect a record where the delimiter
 *  is |. Our custom driver changes all CTRL-A into | and
 *  escapes the already existing | from input.
 *-----------------------------------------------------------------------*/

static int
DrRead(bf, size, count)
char*bf;
int size;
int*count;
{
int rtn;/* return value  */
int n; /* bytes read in */
static char*bftemp = 0;/* temp buffer*/
char*p; /* pointer to temp buff  */
char*start; /* start of output buffer*/
static off_tcurrseek = 0;  /* current seek in input */
int escaped = 0;/* did we escape last character */

start = bf;

if (bftemp == 0)
{
if ( (bftemp = malloc(size)) == 0 )
{
return PL_RTN_FAIL;
}
}


/*
 * read data in
 */
errno = 0;
do
{
n = read(fd, bftemp, size);
} while (n == -1 && errno == 4);

rtn = (n < 0) ? PL_RTN_FAIL : (n == size) ? PL_RTN_OK : PL_RTN_EOF;

currseek += n;

p = bftemp;

/*
 * format output buffer
 */
while (size)
{
if (*p == REALDELIM)
{
*bf = ESCAPSIGN;
escaped = 1;
}
else if (*p == CTRLDELIM)
{
*bf = REALDELIM;
}
else
{
*bf = *p;
}

size--;
bf++;

if (escaped && size)
```

```
{
*bf++ = *p;
escaped = 0;
size--;
}

p++;

if ((int) (p - bftemp) == n)
break;
}

if (escaped)
{
p--;
rtn = PL_RTN_OK;
}

if ((int) (p - bftemp) != n)
{
currseek -= (off_t) (n - (p - bftemp));
lseek(fd, currseek, SEEK_SET);
}

if (rtn == PL_RTN_EOF)
{
*bf = 0;
}

*count = (int) (bf - start);

return rtn;
}


static int
DrWrite(bf, size)
char*bf;
int  size;
{
return PL_RTN_OK;
}


static int
DrClose(device)
devicearray *device;
{
close(fd);

return PL_RTN_OK;
}
```

Available driver methods

The following section describes the methods that you can use to build custom drivers. The methods defined in this section use the return values that are described in the following table.

Return value	Interpretation
PL_RTN_OK	Method ran successfully.
PL_RTN_FAIL	Method did not succeed. Stop processing.
PL_RTN_EOF	Method reached the end of the file.

- The PL_MTH_OPEN function
  Call this function to open the file of interest for the load or unload.
- The PL_MTH_CLOSE function
  Call this function to close the file at the end of the load or unload.

The PL_MTH_OPEN function

Call this function to open the file of interest for the load or unload.

Function information		Description of arguments
Input arguments:	devicearray *device;	Device/file path name to open
Output arguments:	None	
Return values:	PL_RTN_OK, PL_RTN_FAIL	

The PL_MTH_CLOSE function

Call this function to close the file at the end of the load or unload.

Function information		Description of arguments
Input arguments:	None	
Output arguments:	None	
Return values:	PL_RTN_OK, PL_RTN_FAIL	

Available API support functions

This section describes the API support functions that you can use with custom drivers.

The methods defined in this section use the return values that are described in the following table.

Return value	Interpretation
PL_RTN_OK	Function succeeded.
PL_RTN_FAIL n	Function failed.

- The pl_inherit_methods(driver, methodtable) function
  This function loads the passed method function table with the functions currently configured for the passed driver.
- The pl_set_method_function(methodtable, method, function) function
  This function inserts the passed function into the method chain.
- The pl_driver_inherit(method) function
  This function returns a pointer to the function that currently supports the passed method and advances the method chain so that the next request returns the next function in the method list.
- The pl_get_recordlength() function
  This function returns the length of the active record format. If the format is for a delimited record type, the value returned is `0`.
- The pl_set_informix_conversion(flag) function
  When this function is set to `1`, it disables data conversion during the load. The data must be formatted in IBM® Informix® format (as opposed to another type of format that would have to be converted before loading the data into the database). When this function is set to `0`, data conversion is enabled.
- The pl_lock_globals() function
  This function ensures global data integrity when you supply a custom driver that uses globally defined variables.
- The pl_reset_inherit_chain(method) function
  This function resets the function inheritance chain to the start of the list. You should need it only if you are implementing recursive processing.

The pl_inherit_methods(driver, methodtable) function

This function loads the passed method function table with the functions currently configured for the passed driver.

Function information		Description of arguments
Function type:	int	
Input arguments:	char *driver void *methodtable	Name of driver to inherit Pointer to method table
Return values:	PL_RTN_OK, PL_RTN_FAIL	

The **onpload** utility is supplied with three standard drivers described in Driver initialization.

The pl_set_method_function(methodtable, method, function) function

This function inserts the passed function into the method chain.

Function information		Description of arguments
Function type:	int	
Input arguments:	void *methodtable int method int (*funcptr)()	Method table passed to PL_MTH_CONFIGURE Method ID Function to insert into method chain
Return values:	PL_RTN_OK, PL_RTN_FAIL	

The pl_driver_inherit(method) function

This function returns a pointer to the function that currently supports the passed method and advances the method chain so that the next request returns the next function in the method list.

Function information		Description of arguments
Function type:	int	
Input argument:	int method	Method ID
Return values:	PL_RTN_OK, PL_RTN_FAIL	

When you inherit a driver and use the **pl_set_method_function** to override a method, you can assume all processing functionality for method.

However, if you want to add to the existing processing functionality, **pl_driver_inherit** returns the function that was installed in the function table before the call to **pl_set_method_function**.

Example

You are processing a file in which the context of the data determines the record fields present. You want to analyze the records and reformat the data into a delimited format that the **onpload** data converter recognizes.

In the driver setup, specify that your function should inherit the Delimited driver and add your custom function to the **PL_MTH_READREC** method.

When your function is called, get the inherited function with **pl_driver_inherit()**. Invoke this function, which returns a pointer to the start of a record and its length.

Apply your changes to the data in the buffer that is returned by the call to the inherited function.

The pl_get_recordlength() function

This function returns the length of the active record format. If the format is for a delimited record type, the value returned is `0`.

Function information		Description of arguments
Function type:	int	
Return values	length of record PL_RTN_FAIL	Function succeeded Function failed

The pl_set_informix_conversion(flag) function

When this function is set to `1`, it disables data conversion during the load. The data must be formatted in IBM® Informix® format (as opposed to another type of format that would have to be converted before loading the data into the database). When this function is set to `0`, data conversion is enabled.

Function information		Description of arguments
Function type:	int	
Input argument:	int flag	1 = disable, 0 = enable
Return values:	PL_RTN_OK, PL_RTN_FAIL	

The pl_lock_globals() function

This function ensures global data integrity when you supply a custom driver that uses globally defined variables.

Function information		Description of arguments
Function type:	int	
Return values:	PL_RTN_OK, PL_RTN_FAIL	

The pl_reset_inherit_chain(method) function

This function resets the function inheritance chain to the start of the list. You should need it only if you are implementing recursive processing.

Function information		Description of arguments
Function type:	void	
Input arguments:	int method	Method ID
Return values:	none	

Run load and unload jobs on a Windows computer

You can run load and unload jobs on Windows computers by using the following methods:

- Prepare and run jobs with the **onpladm** utility on a Windows computer.
- Prepare jobs with the **ipload** utility on a UNIX computer; then run them on a database server on a Windows computer.

- The onpladm utility on Windows
  You can run all **onpladm** utility commands from the Windows command line.
- Preparing jobs with the ipload utility on Windows computers
  You cannot prepare load and unload jobs for the High-Performance Loader (HPL) on Windows computers. However, you can use the **ipload** utility on UNIX to prepare the load and unload jobs and then use the **onpload** command on Windows to run those jobs.

Related concepts:
The ipload utility
Related reference:
Prepare to use the ipload utility

The onpladm utility on Windows

You can run all **onpladm** utility commands from the Windows command line.

If you use the **onpladm** utility to start a project on Windows, the utility starts each **onpload** job in a new command window because, by default, the INTERACTIVE_DESKTOP_OFF environment variable is set to `0`. If you want the jobs to start in the background, set the INTERACTIVE_DESKTOP_OFF environment variable to `1`.

The setting of the INTERACTIVE_DESKTOP_OFF environment variable influences the behavior of all stored procedures in an instance.

For general information about the **onpladm** utility, see The onpladm utility.

- Run the Run Job or Run Project commands
  To run the Run Job or Run Project commands with the **onpladm** utility, enter a user name and password in the **Host Information** tab of the **SETNET32** utility of the IBM® Informix® Client Software Development Kit.
- Running onpladm on UNIX with the database server running on Windows
  You can run the Run Job or Run Project commands on the **onpladm** utility on a UNIX computer and create High-Performance Loader (HPL) objects in a database server that runs on a Windows computer.

Related reference:
Run all jobs in a project

Run the Run Job or Run Project commands

To run the Run Job or Run Project commands with the **onpladm** utility, enter a user name and password in the **Host Information** tab of the **SETNET32** utility of the IBM® Informix® Client Software Development Kit.

For more information about the **SETNET32** utility, see the *Informix Client Products Installation Guide*.

If you do not enter the user name and user password, you receive the following error:

```
Cannot execute stored procedure start_onpload SQL ERROR -668
```

If you receive this error, the online log file contains the following message:

```
System() command "$INFORMIXDIR/bin/onpload -H hostname -S
servername -rl -fb" in SPL routine cannot be executed because user
"username" did not connect with a password.
```

To run other commands with the **onpladm** utility, you do not have to enter a user name and password.

Running onpladm on UNIX with the database server running on Windows

You can run the Run Job or Run Project commands on the **onpladm** utility on a UNIX computer and create High-Performance Loader (HPL) objects in a database server that runs on a Windows computer.

To run the Run Job or Run Project commands on UNIX with a database server on Windows:

1. Make the UNIX computer a trusted host on your Windows computer.
2. Verify that the UNIX computer and Windows computers can connect to each other.
3. Install a database server on the Windows computer. Verify that the database server is running.
4. Install a database server on the UNIX computer.
   Ensure that the database servers on the UNIX and Windows computers are the same version. You do not need to start the database server on the UNIX computer to run the **onpladm** utility.
5. Set the INFORMIXSERVER and INFORMIXSQLHOSTS environment variables on your UNIX workstation as follows:

```
INFORMXSERVER WINservername
INFORMIXSQLHOSTS full.sqlhosts.pathname
```

WINservername
Name of the database server on the Windows computer
full.sqlhosts.pathname
Complete path of the sqlhosts file on the UNIX computer (for instance, $INFORMIXDIR/etc/sqlhosts.wn)

Use the **-S** or **-T onpladm** utility option to override the INFORMIXSERVER environment variable setting.

6. Add the following line to the $INFORMIXSQLHOSTS file on the UNIX workstation:

```
WINservername ontlitcp WINname servicename
```

WINservername
Name of the database server on the Windows computer.
WINname
Name of the Windows computer.
servicename
Service that the Windows database server uses.

7. Use the **onpladm** utility commands to create, configure, delete, describe, list, and modify HPL objects on your Windows database server. For more information, see The onpladm utility.
8. Prepare load and unload jobs with the **onpladm** utility. For more information, see Create onpladm jobs.
9. Type the following line into the .netrc file in your home directory on a UNIX computer:

```
machine WIN_machinename login username password user_password
```

WIN_machinename
Name of the Windows computer.
username
Your user name.
user_password
Your user password.

Your user name and user password must be valid on the Windows computer.

Related reference:
Run a job

Preparing jobs with the ipload utility on Windows computers

You cannot prepare load and unload jobs for the High-Performance Loader (HPL) on Windows computers. However, you can use the **ipload** utility on UNIX to prepare the load and unload jobs and then use the **onpload** command on Windows to run those jobs.

To use the **ipload** utility on UNIX to prepare jobs for a database server on Windows:

1. Make the UNIX computer a trusted host on your Windows computer.
2. Ensure that the UNIX computer can connect to the Windows computer.
3. Install a database server on Windows and make sure that it is running.
4. Install a database server on UNIX. You need not start the database server to run the **ipload** utility.
5. Make sure that the database server installed on UNIX is the same version as the IBM® Informix® database server on Windows.
6. If the database server on UNIX is running, do not modify the environment variables. If the database server on UNIX is not running, set the environment variables (from your UNIX workstation), as the following example shows:

```
INFORMIXSERVER WINservername
INFORMIXSQLHOSTS full.sqlhosts.pathname
```

WINservername
The name of the database server on Windows
full.sqlhosts.pathname
The full path name of the sqlhosts file on your UNIX workstation (for example, $INFORMIXDIR/etc/sqlhosts.wn)

7. Add the following line to the $INFORMIXSQLHOSTS file on the UNIX workstation:

```
WINservername ontlitcp WINname servicename
```

WINservername
Name of the database server on the Windows computer.
WINname
Name of the Windows computer.
servicename
Service that the Windows database server uses.

8. On UNIX, run **ipload** as the DBA or as user **informix**.
If the database server on UNIX is not running (and you set the environment variables correctly), **ipload** immediately connects to the Windows database server. If the database server on UNIX is running, **ipload** initially connects to that database server. To connect to the database server on Windows, choose Configure Server.

   The Connect Server dialog shows two columns: Onpload Server and Target Server. Select the database server on Windows in both columns and click OK. The **ipload** utility connects to the database server on Windows.

9. Create, edit, view, and run HPL jobs on the database server on Windows. After you prepare the load and unload jobs by using **ipload**, you can also run the jobs by using the **onpload** command on your Windows computer.

Conversion and reversion scripts for HPL database migration

When you convert or revert to different versions of the database server, you can use conversion and reversion scripts to manually upgrade or revert your onpload database. You must use these scripts if you are required to upgrade between the same server versions.

Alternatively, you can use the **IFX_ONPLOAD_AUTO_UPGRADE** environment variable with the **ipload** or **onpladm** utilities to automatically upgrade the onpload database the first time that you run a High-Performance Loader (HPL) utility after you migrate to a new version of the database server. You cannot use the **IFX_ONPLOAD_AUTO_UPGRADE** environment variable with the **onpload** utility.

If you run an HPL utility before upgrading the onpload database, you receive the following error, which informs you that the onpload database must be converted:

```
Incorrect database version. Make sure that it is upgraded properly.
To upgrade please run the $INFORMIXDIR/etc/conv/conpload.sh script manually.
For automatic upgrade please set the IFX_ONPLOAD_AUTO_UPGRADE variable.
```

- Upgrade the High-Performance Loader onpload database
  When you upgrade to a new version of the database server, you must also upgrade the onpload database.
- Revert from the current onpload database

Upgrade the High-Performance Loader onpload database

When you upgrade to a new version of the database server, you must also upgrade the onpload database.

If you are upgrading from a version of the database server that is before Version 9.40, you must run a conversion script. For instructions for running this script, see your *IBM® Informix® Migration Guide*.

Starting with Version 9.40.xC3, the database server has a new version of the onpload database with longer column lengths. The onpload database now requires slightly more disk space than it did in previous versions. The following table shows the relationship between the database server version and the onpload database schema version.

Database server version	Database version of onpload
Pre 7.3	0 or not available
7.31	1
9.2, 9.3, 9.40.xC1, 9.40.xC2	2
9.40.xC3 and later	3

Revert from the current onpload database

Starting with Version 10.00 of the database server, reversion of the onpload database is automatic.

Performance Guide

These topics describe how to configure and operate your IBM® Informix® database server to improve overall system throughput and to improve the performance of SQL queries.

This information contains performance tuning issues and methods that are relevant to daily database server administration and query execution. Performance measurement and tuning encompass a broad area of research and practice and can involve information beyond the scope of this publication.

This information is intended for the following users:

- Database administrators
- Database server administrators
- Database-application programmers
- Performance engineers

This information assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with database server administration, operating-system administration, or network administration

Information in these topics can help you perform the following tasks:

- Monitor the system resources that are critical to performance
- Identify database activities that affect these critical resources
- Identify and monitor queries that are critical to performance
- Use the database server utilities (especially **onperf** and **onstat**) for performance monitoring and tuning
- Eliminate performance bottlenecks by:
  - Balancing the load on system resources
  - Adjusting the configuration parameters or environment variables of your database server
  - Adjusting the arrangement of your data

- Allocating resources for decision-support queries
- Creating indexes to speed up retrieval of your data

Performance measurement and tuning encompass a broad area of research and practice and can involve information beyond the scope of these topics.

These topics are taken from the *IBM Informix Performance Guide*.

- Performance basics
  Performance measurement and tuning issues and methods are relevant to daily database server administration and query execution.
- Performance monitoring and the tools you use
  You can use performance monitoring tools to create a performance history, to monitor database resources at scheduled times, or to monitor ongoing transaction or query performance.
- Effect of configuration on CPU utilization
  The combination of operating-system and Informix configuration parameters can affect CPU utilization. You can change the settings of the Informix configuration parameters that directly affect CPU utilization, and you can adjust the settings for different types of workloads.
- Effect of configuration on memory utilization
  The combination of operating-system and Informix configuration parameters can affect memory utilization.
- Effect of configuration on I/O activity
  The configuration of your database server affects I/O activity.
- Table performance considerations
  Some performance issues are associated with unfragmented tables and table fragments.
- Boosted Partition Free Space Caches (PFSC)
- Indexes and index performance considerations
  Informix provides several types of indexes. Some performance issues are associated with indexes.
- Locking
  The database server uses locks, which can affect concurrency and performance. You can monitor and administer locks.
- Fragmentation guidelines
  One of the most frequent causes of poor performance in relational database systems is contention for data that resides on a single I/O device. Proper fragmentation of high-use tables can significantly reduce I/O contention. These topics discuss the performance considerations that are involved when you use table fragmentation.
- Queries and the query optimizer
  These topics describe query plans, explain how the database server manages query optimization, and discuss factors that you can use to influence the query plan. These topics also describe performance considerations for SPL routines, the UDR cache, and triggers.
- Optimizer directives
  Optimizer directives are comments that tell the query optimizer how to execute a query. You can use optimizer directives to improve query performance.
- Parallel database query (PDQ)
  You can manage how the database server performs PDQ and you can monitor the resources that the database server uses for PDQ.
- Improving individual query performance
  You can test, monitor, and improve queries.
- The onperf utility on UNIX
  The **onperf** utility is a windowing environment that you can use to monitor the database server performance. The **onperf** utility monitors the database server running on the UNIX operating system.
- Appendix

Performance basics

Performance measurement and tuning issues and methods are relevant to daily database server administration and query execution.

These topics:

- Describe a basic approach for performance measurement and tuning
- Provide guidelines for a quick start to obtain acceptable initial performance on a small database
- Describe roles in maintaining good performance

- Developing a basic approach to performance measurement and tuning
  To maintain optimum performance for your database applications, develop a plan for measuring system performance, making adjustments to maintain good performance and taking corrective measures when performance degrades. Regular, specific measurements can help you to anticipate and correct performance problems.
- Quick start for acceptable performance on a small database
  If you have a small database with each table residing on only one disk and using only one CPU virtual processor, you can take specific measurements to help you anticipate and correct performance problems.
- Performance goals
  When you plan for measuring and tuning performance, you should consider performance goals and determine which goals are the most important.
- Measurements of performance
  You can use throughput, response time, cost per transaction, and resource utilization measures to evaluate performance.
- Resource utilization and performance
  A typical transaction-processing application undergoes different demands throughout its various operating cycles. Peak loads during the day, week, month, and year, as well as the loads imposed by decision-support (DSS) queries or backup operations, can significantly impact any system that is running near capacity. You can use direct historical data derived from your particular system to pinpoint this impact.
- Factors that affect resource utilization
  The performance of your database server application depends many factors, including hardware and software configuration, your network configuration, and the design of your database.
- Maintenance of good performance
  Performance is affected in some way by all system users: the database server administrator, the database administrator, the application designers, and the client application users.

Developing a basic approach to performance measurement and tuning

To maintain optimum performance for your database applications, develop a plan for measuring system performance, making adjustments to maintain good performance and taking corrective measures when performance degrades. Regular, specific measurements can help you to anticipate and correct performance problems.

By recognizing problems early, you can prevent them from affecting users significantly. Early indications of a performance problem are often vague; users might report that the system seems sluggish. Users might complain that they cannot get all their work done, that transactions take too long to complete, that queries take too long to process, or that the application slows down at certain times during the day.

To determine the nature of the problem, you must measure the actual use of system resources and evaluate the results.

Users typically report performance problems in the following situations:

- Response times for transactions or queries take longer than expected.
- Transaction throughput is insufficient to complete the required workload.
- Transaction throughput decreases.

An iterative approach to optimizing database server performance is recommended. If repeating the steps found in the following list does not produce the desired improvement, insufficient hardware resources or inefficient code in one or more client applications might be causing the problem.

To optimize performance:

1. Establish performance objectives.
2. Take regular measurements of resource utilization and database activity.
3. Identify symptoms of performance problems: disproportionate utilization of CPU, memory, or disks.
4. Tune the operating-system configuration.
5. Tune the database server configuration.
6. Optimize the chunk and dbspace configuration, including placement of logs, sort space, and space for temporary tables and sort files.
7. Optimize the table placement, extent sizing, and fragmentation.
8. Improve the indexes.
9. Optimize background I/O activities, including logging, checkpoints, and page cleaning.
10. Schedule backup and batch operations for off-peak hours.
11. Optimize the implementation of the database application.
12. Repeat steps 2 through 11.

Related concepts:
Performance goals
Measurements of performance
Resource utilization and performance
Factors that affect resource utilization
Maintenance of good performance
Related tasks:
Quick start for acceptable performance on a small database

Quick start for acceptable performance on a small database

If you have a small database with each table residing on only one disk and using only one CPU virtual processor, you can take specific measurements to help you anticipate and correct performance problems.

To achieve acceptable initial performance on a small database:

1. Generate statistics of your tables and indexes to provide information to the query optimizer to enable it to choose query plans with the lowest estimated cost. These statistics are a minimum starting point to obtain good performance for individual queries. For guidelines, see Update statistics when they are not generated automatically. To see the query plan that the optimizer chooses for each query, see Display the query plan.

2. If you want a query to run in parallel with other queries, you must turn on the Parallel Database Query (PDQ) feature.
   Without table fragmentation across multiple disks, parallel scans do not occur. With only one CPU virtual processor, parallel joins or parallel sorts do not occur. However, PDQ priority can obtain more memory to perform the sort. For more information, see Parallel database query (PDQ).

3. If you want to mix online transaction processing (OLTP) and decision-support system (DSS) query applications, you can control the amount of resources a long-running query can obtain so that your OLTP transactions are not affected.
   For information about how to control PDQ resources, see The allocation of resources for parallel database queries.

4. Monitor sessions and drill down into various details to improve the performance of individual queries.
   For information about the various tools and session details to monitor, see Monitoring memory usage for each session and Monitor sessions and threads.

Related concepts:
Performance goals
Measurements of performance
Resource utilization and performance
Factors that affect resource utilization
Maintenance of good performance
Related tasks:
Developing a basic approach to performance measurement and tuning

Performance goals

When you plan for measuring and tuning performance, you should consider performance goals and determine which goals are the most important.

Many considerations go into establishing performance goals for the database server and the applications that it supports. Be clear and consistent about articulating performance goals and priorities, so that you can provide realistic and consistent expectations about the performance objectives for your application. Consider the following questions when you establish performance goals:

- Is your top priority to maximize transaction throughput, minimize response time for specific queries, or achieve the best overall mix?
- What sort of mix between simple transactions, extended decision-support queries, and other types of requests does the database server typically handle?
- At what point are you willing to trade transaction-processing speed for availability or the risk of loss for a particular transaction?
- Is this database server instance used in a client/server configuration? If so, what are the networking characteristics that affect its performance?
- What is the maximum number of users that you expect?
- Is your configuration limited by memory, disk space, or CPU resources?

The answers to these questions can help you set realistic performance goals for your resources and your mix of applications.

Related concepts:
Measurements of performance
Resource utilization and performance
Factors that affect resource utilization
Maintenance of good performance
Related tasks:
Developing a basic approach to performance measurement and tuning
Quick start for acceptable performance on a small database

Measurements of performance

You can use throughput, response time, cost per transaction, and resource utilization measures to evaluate performance.

Throughput, response time, and cost per transaction are described in the topics that follow.

Resource utilization can have one of two meanings, depending on the context. The term can refer to the amount of a resource that a particular operation requires or uses, or it can refer to the current load on a particular system component. The term is used in the former sense to compare approaches for accomplishing a given task. For instance, if a given sort operation requires 10 megabytes of disk space, its resource utilization is greater than another sort operation that requires only 5 megabytes of disk space. The term is used in the latter sense to refer, for instance, to the number of CPU cycles that are devoted to a particular query during a specific time interval.

For a discussion about the performance impact of different load levels on various system components, see Resource utilization and performance.

- Throughput
  Throughput measures the overall performance of the system. For transaction processing systems, throughput is typically measured in *transactions per second* (TPS) or *transactions per minute* (TPM).
- Response time
  Response time measures the performance of an individual transaction or query. Response time is typically treated as the elapsed time from the moment that a user enters a command or activates a function until the time that the application indicates that the command or function has completed.
- Cost per transaction
  The cost per transaction is a financial measure that is typically used to compare overall operating costs among applications, database servers, or hardware platforms. You can measure the cost per transaction.

Related concepts:
Performance goals
Resource utilization and performance
Factors that affect resource utilization
Maintenance of good performance
Related tasks:
Developing a basic approach to performance measurement and tuning
Quick start for acceptable performance on a small database

Throughput

Throughput measures the overall performance of the system. For transaction processing systems, throughput is typically measured in *transactions per second* (TPS) or *transactions per minute* (TPM).

Throughput depends on the following factors:

- The specifications of the host computer
- The processing overhead in the software
- The layout of data on disk

- The degree of parallelism that both hardware and software support
- The types of transactions being processed

- Ways to measure throughput
  The best way to measure throughput for an application is to include code in the application that logs the time stamps of transactions as they commit.
- Standard throughput benchmarks
  The Transaction Processing Performance Council (TPC) provides standard benchmarks that allow reasonable throughput comparisons across hardware configurations and database servers. is an active member in good standing of the TPC.

Ways to measure throughput

The best way to measure throughput for an application is to include code in the application that logs the time stamps of transactions as they commit.

If your application does not provide support for measuring throughput directly, you can obtain an estimate by tracking the number of COMMIT WORK statements that the database server logs during a given time interval. You can use the **onlog** utility to obtain a listing of logical-log records that are written to log files. You can use information from this command to track insert, delete, and update operations as well as committed transactions. However, you cannot obtain information stored in the logical-log buffer until that information is written to a log file.

If you need more immediate feedback, you can use **onstat -p** to gather an estimate. You can use the SET LOG statement to set the logging mode to unbuffered for the databases that contain tables of interest. You can also use the trusted auditing facility in the database server to record successful COMMIT WORK events or other events of interest in an audit log file. Using the auditing facility can increase the overhead involved in processing any audited event, which can reduce overall throughput.

Related information:
Auditing data security

Standard throughput benchmarks

The Transaction Processing Performance Council (TPC) provides standard benchmarks that allow reasonable throughput comparisons across hardware configurations and database servers. is an active member in good standing of the TPC.

The TPC provides the following standardized benchmarks for measuring throughput:

- TPC-A
  This benchmark is used for simple online transaction-processing (OLTP) comparisons. It characterizes the performance of a simple transaction-processing system, emphasizing update-intensive services. TPC-A simulates a workload that consists of multiple user sessions connected over a network with significant disk I/O activity.
- TPC-B
  This benchmark is used for stress-testing peak database throughput. It uses the same transaction load as TPC-A but removes any networking and interactive operations to provide a best-case throughput measurement.
- TPC-C
  This benchmark is used for complex OLTP applications. It is derived from TPC-A and uses a mix of updates, read-only transactions, batch operations, transaction rollback requests, resource contentions, and other types of operations on a complex database to provide a better representation of typical workloads.
- TPC-D
  This benchmark measures query-processing power in terms of completion times for very large queries. TPC-D is a decision-support benchmark built around a set of typical business questions phrased as SQL queries against large databases (in the gigabyte or terabyte range).

Because every database application has its own particular workload, you cannot use TPC benchmarks to predict the throughput for your application. The actual throughput that you achieve depends largely on your application.

Response time

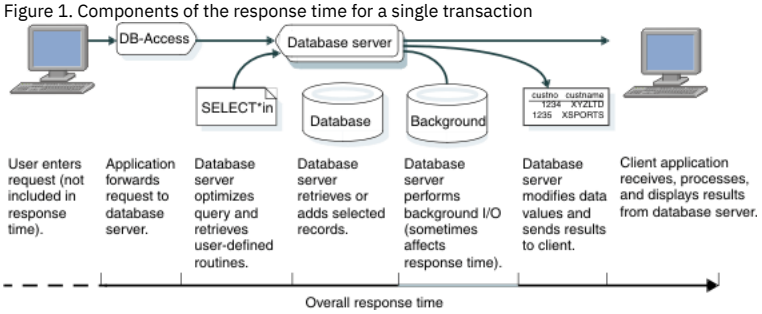
Response time measures the performance of an individual transaction or query. Response time is typically treated as the elapsed time from the moment that a user enters a command or activates a function until the time that the application indicates that the command or function has completed.

The response time for a typical Informix® application includes the following sequence of actions. Each action requires a certain amount of time. The response time does not include the time that it takes for the user to think of and enter a query or request:

1. The application forwards a query to the database server.
2. The database server performs query optimization and retrieves any user-defined routines (UDRs). UDRs include both SPL routines and external routines.
3. The database server retrieves, adds, or updates the appropriate records and performs disk I/O operations directly related to the query.
4. The database server performs any background I/O operations, such as logging and page cleaning, that occur during the period in which the query or transaction is still pending.
5. The database server returns a result to the application.
6. The application displays the information or issues a confirmation and then issues a new prompt to the user.

Figure 1 contains a diagram that shows how the actions just described in steps 1 through 6 contribute to the overall response time.

Figure 1. Components of the response time for a single transaction



- Response time and throughput
  Response time and throughput are related. The response time for an average transaction tends to decrease as you increase overall throughput.
- Response-time measurement
  To measure the response time for a query or application, you can use the timing commands and performance monitoring and timing functions that your operating system provides.

Response time and throughput

Response time and throughput are related. The response time for an average transaction tends to decrease as you increase overall throughput.

However, you can decrease the response time for a specific query, at the expense of overall throughput, by allocating a disproportionate amount of resources to that query. Conversely, you can maintain overall throughput by restricting the resources that the database allocates to a large query.

The trade-off between throughput and response time becomes evident when you try to balance the ongoing need for high transaction throughput with an immediate need to perform a large decision-support query. The more resources that you apply to the query, the fewer you have available to process transactions, and the larger the impact your query can have on transaction throughput. Conversely, the fewer resources you allow the query, the longer the query takes.

Response-time measurement

To measure the response time for a query or application, you can use the timing commands and performance monitoring and timing functions that your operating system provides.

- Operating-system timing commands
  Your operating system typically has a utility that you can use to time a command. You can often use this timing utility to measure the response times to SQL statements that a DB-Access command file issues.
- Operating-system tools for monitoring performance
  Operating systems usually have a performance monitor that you can use to measure response time for a query or process.
- Timing functions within your application
  Most programming languages have a library function for the time of day. If you have access to the source code, you can insert pairs of calls to this function to measure the elapsed time between specific actions.

Operating-system timing commands

Your operating system typically has a utility that you can use to time a command. You can often use this timing utility to measure the response times to SQL statements that a DB-Access command file issues.

UNIX Only

If you have a command file that performs a standard set of SQL statements, you can use the **time** command on many systems to obtain an accurate timing for those commands.

The following example shows the output of the UNIX **time** command:

```
time commands.dba
...
4.3 real          1.5 user             1.3 sys
```

The **time** output lists the amount of elapsed time (real), the user CPU time, and the system CPU time. If you use the C shell, the first three columns of output from the C shell **time** command show the user, system, and elapsed times, respectively. In general, an application often performs poorly when the proportion of system CPU time exceeds one-third of the total elapsed time.

The **time** command gathers timing information about your application. You can use this command to invoke an instance of your application, perform a database operation, and then exit to obtain timing figures, as the following example illustrates:

```
time sqlapp
   (enter SQL command through sqlapp, then exit)
10.1 real          6.4 user          3.7 sys
```

You can use a script to run the same test repeatedly, which allows you to obtain comparable results under different conditions. You can also obtain estimates of your average response time by dividing the elapsed time for the script by the number of database operations that the script performs.

Operating-system tools for monitoring performance

Operating systems usually have a performance monitor that you can use to measure response time for a query or process.

Windows Only
: You can often use the Performance Logs and Alerts that the Windows operating system supplies to measure the following times:

- User time
- Processor time
- Elapsed time

Timing functions within your application

Most programming languages have a library function for the time of day. If you have access to the source code, you can insert pairs of calls to this function to measure the elapsed time between specific actions.

ESQL/C Only
: For example, if the application is written in IBM® Informix® ESQL/C, you can use the **dtcurrent()** function to obtain the current time. To measure response time, you can call **dtcurrent()** to report the time at the start of a transaction and again to report the time when the transaction commits.

Elapsed time, in a multiprogramming system or network environment where resources are shared among multiple processes, does not always correspond to execution time. Most operating systems and C libraries contain functions that return the CPU time of a program.

Cost per transaction

The cost per transaction is a financial measure that is typically used to compare overall operating costs among applications, database servers, or hardware platforms. You can measure the cost per transaction.

To measure the cost per transaction:

1. Calculate all the costs associated with operating an application. These costs can include the installed price of the hardware and software; operating costs, including salaries; and other expenses. These costs can include the installed price of the hardware and software; operating costs, including salaries; and other expenses.
2. Project the total number of transactions and queries for the effective life of an application.
3. Divide the total cost over the total number of transactions.

Although this measure is useful for planning and evaluation, it is seldom relevant to the daily issues of achieving optimum performance.

Resource utilization and performance

A typical transaction-processing application undergoes different demands throughout its various operating cycles. Peak loads during the day, week, month, and year, as well as the loads imposed by decision-support (DSS) queries or backup operations, can significantly impact any system that is running near capacity. You can use direct historical data derived from your particular system to pinpoint this impact.

You must take regular measurements of the workload and performance of your system to predict peak loads and compare performance measurements at different points in your usage cycle. Regular measurements help you to develop an overall performance profile for your database server applications. This profile is critical in determining how to improve performance reliably.

For the measurement tools that the database server provides, see Database server tools. For the tools that your operating system provides for measuring performance impacts on system and hardware resources, see Operating-system tools.

Utilization is the percentage of time that a component is actually occupied, as compared with the total time that the component is available for use. For instance, if a CPU processes transactions for a total of 40 seconds during a single minute, its utilization during that interval is 67 percent.

Measure and record utilization of the following system resources regularly:

- CPU
- Memory

- Disk

A resource is said to be *critical* to performance when it becomes overused or when its utilization is disproportionate to that of other components. For instance, you might consider a disk to be critical or overused when it has a utilization of 70 percent and all other disks on the system have 30 percent. Although 70 percent does not indicate that the disk is severely overused, you can improve performance by rearranging data to balance I/O requests across the entire set of disks.

How you measure resource utilization depends on the tools that your operating system provides for reporting system activity and resource utilization. After you identify a resource that seems overused, you can use the performance-monitoring utilities that the database server provides to gather data and make inferences about the database activities that might account for the load on that component. You can adjust your database server configuration or your operating system to reduce those database activities or spread them among other components. In some cases, you might need to provide additional hardware resources to resolve a performance bottleneck.

- Resource utilization
  Whenever a system resource, such as a CPU or a particular disk, is occupied by a transaction or query, the resource is unavailable for processing other requests. Pending requests must wait for the resources to become available before they can complete.
- CPU utilization
  Estimates of CPU utilization and response time can help you determine if you need to eliminate or reschedule some activities.
- Memory utilization
  Memory is not managed as a single component, such as a CPU or disk, but as a collection of small components called *pages*.
- Disk utilization
  Because transfer rates vary among disks, most operating systems do not report disk utilization directly. Instead, they report the number of data transfers per second (in operating-system memory-page-size units.)

Related concepts:
Performance goals
Measurements of performance
Factors that affect resource utilization
Maintenance of good performance
Related tasks:
Developing a basic approach to performance measurement and tuning
Quick start for acceptable performance on a small database

Resource utilization

Whenever a system resource, such as a CPU or a particular disk, is occupied by a transaction or query, the resource is unavailable for processing other requests. Pending requests must wait for the resources to become available before they can complete.

When a component is too busy to keep up with all its requests, the overused component becomes a bottleneck in the flow of activity. The higher the percentage of time that the resource is occupied, the longer each operation must wait for its turn.

You can use the following formula to estimate the service time for a request based on the overall utilization of the component that services the request. The expected service time includes the time that is spent both waiting for and using the resource in question. Think of service time as that portion of the response time accounted for by a single component within your computer, as the following formula shows:

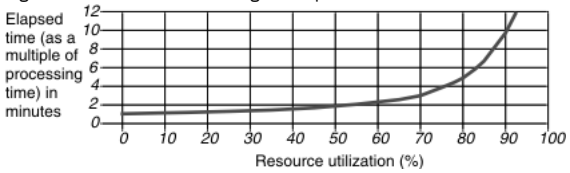
$$S = P / (1 - U)$$

S
: is the expected service time.

P
: is the processing time that the operation requires after it obtains the resource.

U
: is the utilization for the resource (expressed as a decimal).

As Figure 1 shows, the service time for a single component increases dramatically as the utilization increases beyond 70 percent. For instance, if a transaction requires 1 second of processing by a given component, you can expect it to take 2 seconds on a component at 50 percent utilization and 5 seconds on a component at 80 percent utilization. When utilization for the resource reaches 90 percent, you can expect the transaction to take 10 seconds to make its way through that component.
Figure 1. Service Time for a Single Component as a Function of Resource Utilization



If the average response time for a typical transaction soars from 2 or 3 seconds to 10 seconds or more, users are certain to notice and complain.
Important: Monitor any system resource that shows a utilization of over 70 percent or any resource that exhibits symptoms of overuse as described in the following sections.
When you consider resource utilization, also consider whether increasing the page size of a standard or temporary dbspace is beneficial in your environment. If you want a longer key length than is available for the default page size of a standard or temporary dbspace, you can increase the page size.

CPU utilization

Estimates of CPU utilization and response time can help you determine if you need to eliminate or reschedule some activities.

You can use the resource-utilization formula in the previous topic (Resource utilization) to estimate the response time for a heavily loaded CPU. However, high utilization for the CPU does not always indicate a performance problem. The CPU performs all calculations that are needed to process transactions. The more transaction-related calculations that it performs within a given period, the higher the throughput will be for that period. As long as transaction throughput is high and seems to remain proportional to CPU utilization, a high CPU utilization indicates that the computer is being used to the fullest advantage.

On the other hand, when CPU utilization is high but transaction throughput does not keep pace, the CPU is either processing transactions inefficiently or it is engaged in activity not directly related to transaction processing. CPU cycles are being diverted to internal housekeeping tasks such as memory management.

You can easily eliminate the following activities:

- Large queries that might be better scheduled at an off-peak time
- Unrelated application programs that might be better performed on another computer

If the response time for transactions increases to such an extent that delays become unacceptable, the processor might be swamped; the transaction load might be too high for the computer to manage. Slow response time can also indicate that the CPU is processing transactions inefficiently or that CPU cycles are being diverted.

When CPU utilization is high, a detailed analysis of the activities that the database server performs can reveal any sources of inefficiency that might be present due to improper configuration. For information about analyzing database server activity, see Database server tools.

Memory utilization

Memory is not managed as a single component, such as a CPU or disk, but as a collection of small components called *pages*.

The size of a typical page in memory can range from 1 to 8 kilobytes, depending on your operating system. A computer with 64 megabytes of memory and a page size of 2 kilobytes contains approximately 32,000 pages.

When the operating system needs to allocate memory for use by a process, it scavenges any unused pages within memory that it can find. If no free pages exist, the memory-management system has to choose pages that other processes are still using and that seem least likely to be needed in the short run. CPU cycles are required to select those pages. The process of locating such pages is called a *page scan*. CPU utilization increases when a page scan is required.

Memory-management systems typically use a *least recently used* algorithm to select pages that can be copied out to disk and then freed for use by other processes. When the CPU has identified pages that it can appropriate, it *pages out* the old page images by copying the old data from those pages to a dedicated disk. The disk or disk partition that stores the page images is called the *swap disk, swap space,* or *swap area*. This paging activity requires CPU cycles as well as I/O operations.

Eventually, page images that have been copied to the swap disk must be brought back in for use by the processes that require them. If there are still too few free pages, more must be paged out to make room. As memory comes under increasing demand and paging activity increases, this activity can reach a point at which the CPU is almost fully occupied with paging activity. A system in this condition is said to be *thrashing*. When a computer is thrashing, all useful work comes to a halt.

To prevent thrashing, some operating systems use a coarser memory-management algorithm after paging activity crosses a certain threshold. This algorithm is called *swapping*. When the memory-management system resorts to swapping, it appropriates all pages that constitute an entire process image at once, rather than a page at a time.

Swapping frees up more memory with each operation. However, as swapping continues, every process that is swapped out must be read in again, dramatically increasing disk I/O to the swap device and the time required to switch between processes. Performance is then limited to the speed at which data can be transferred from the swap disk back into memory. Swapping is a symptom of a system that is severely overloaded, and throughput is impaired.

Many systems provide information about paging activity that includes the number of page scans performed, the number of pages sent out of memory (*paged out*), and the number of pages brought in from memory (*paged in)*:

- Paging out is the critical factor because the operating system pages out only when it cannot find pages that are free already.
- A high rate of page scans provides an early indicator that memory utilization is becoming a bottleneck.
- Pages for terminated processes are freed in place and simply reused, so paging-in activity does not provide an accurate reflection of the load on memory. A high rate of paging in can result from a high rate of process turnover with no significant performance impact.

Although the principle for estimating the service time for memory is the same as that described in Resource utilization and performance, you use a different formula to estimate the performance impact of memory utilization than you do for other system components.

You can use the following formula to calculate the expected paging delay for a given CPU utilization level and paging rate:

$$PD = (C/(1-U)) * R * T$$

PD
: is the paging delay.

C
: is the CPU service time for a transaction.

U
: is the CPU utilization (expressed as a decimal).

R
: is the paging-out rate.

T
: is the service time for the swap device.

As paging increases, CPU utilization also increases, and these increases are compounded. If a paging rate of 10 per second accounts for 5 percent of CPU utilization, increasing the paging rate to 20 per second might increase CPU utilization by an additional 5 percent. Further increases in paging lead to even sharper increases in CPU utilization, until the expected service time for CPU requests becomes unacceptable.

Disk utilization

Because transfer rates vary among disks, most operating systems do not report disk utilization directly. Instead, they report the number of data transfers per second (in operating-system memory-page-size units.)

Because each disk acts as a single resource, you can use the following basic formula to estimate the service time, which is described in detail in Resource utilization:

```
S= P/(1-U)
```

To compare the load on disks with similar access times, simply compare the average number of transfers per second.

If you know the access time for a given disk, you can use the number of transfers per second that the operating system reports to calculate utilization for the disk. To do so, multiply the average number of transfers per second by the access time for the disk as listed by the disk manufacturer. Depending on how your data is laid out on the disk, your access times can vary from the rating of the manufacturer. To account for this variability, you should add 20 percent to the access-time specification of the manufacturer.

The following example shows how to calculate the utilization for a disk with a 30-millisecond access time and an average of 10 transfer requests per second:

```
U =     (A * 1.2) * X
  =     (.03 * 1.2) * 10
  =     .36
```

U
: is the resource utilization (this time of a disk).

A
: is the access time (in seconds) that the manufacturer lists.

X
: is the number of transfers per second that your operating system reports.

You can use the utilization to estimate the processing time at the disk for a transaction that requires a given number of disk transfers. To calculate the processing time at the disk, multiply the number of disk transfers by the average access time. Include an extra 20 percent to account for access-time variability:

```
P = D (A * 1.2)
```

P
: is the processing time at the disk.

D
: is the number of disk transfers.

A
: is the access time (in seconds) that the manufacturer lists.

For example, you can calculate the processing time for a transaction that requires 20 disk transfers from a 30-millisecond disk as follows:

```
P =    20 (.03 * 1.2)
  =    20 * .036
  =    .72
```

Use the processing time and utilization values that you calculated to estimate the expected service time for I/O at the particular disk, as the following example shows:

```
S =    P/(1-U)
  =    .72 / (1 - .36)
  =    .72 / .64
  =    1.13
```


Factors that affect resource utilization

The performance of your database server application depends many factors, including hardware and software configuration, your network configuration, and the design of your database.

You must consider these factors when you attempt to identify performance problems or make adjustments to your system:

- Hardware resources
  As discussed earlier in this chapter, hardware resources include the CPU, physical memory, and disk I/O subsystems.
- Operating-system configuration
  The database server depends on the operating system to provide low-level access to devices, process scheduling, interprocess communication, and other vital services.

  The configuration of your operating system has a direct impact on how well the database server performs. The operating-system kernel takes up a significant amount of physical memory that the database server or other applications cannot use. However, you must reserve adequate kernel resources for the database server to use.
- Network configuration and traffic
  Applications that depend on a network for communication with the database server, and systems that rely on data replication to maintain high availability, are subject to the performance constraints of that network. Data transfers over a network are typically slower than data transfers from a disk. Network delays can have a significant impact on the performance of the database server and other application programs that run on the host computer.
- Database server configuration
  Characteristics of your database server instance, such as the number of CPU virtual processors (VPs), the size of your resident and virtual shared-memory portions, and the number of users, play an important role in determining the capacity and performance of your applications.

- Dbspace, blobspace, and chunk configuration
  The following factors can affect the time that it takes the database server to perform disk I/O and process transactions:
  - The placement of the root dbspace, physical logs, logical logs, and temporary-table dbspaces
  - The presence or absence of mirroring
  - The use of devices that are buffered or unbuffered by the operation system
- Database and table placement
  The placement of tables and fragments within dbspaces, the isolation of high-use fragments in separate dbspaces, and the spreading of fragments across multiple dbspaces can affect the speed at which the database server can locate data pages and transfer them to memory.
- Tblspace organization and extent sizing
  Fragmentation strategy and the size and placement of extents can affect the ability of the database server to scan a table rapidly for data. Avoid interleaved extents and allocate extents that are sufficient to accommodate growth of a table to prevent performance problems.
- Query efficiency
  Proper query construction and cursor use can decrease the load that any one application or user imposes. Remind users and application developers that others require access to the database and that each person's activities affect the resources that are available to others.
- Scheduling background I/O activities
  Logging, checkpoints, page cleaning, and other operations, such as making backups or running large decision-support queries, can impose constant overhead and large temporary loads on the system. Schedule backup and batch operations for off-peak times whenever possible.
- Remote client/server operations and distributed join operations
  These operations have an important impact on performance, especially on a host system that coordinates distributed joins.
- Application-code efficiency
  Application programs introduce their own load on the operating system, the network, and the database server. These programs can introduce performance problems if they make poor use of system resources, generate undue network traffic, or create unnecessary contention in the database server. Application developers must make proper use of cursors and locking levels to ensure good database server performance.

Related concepts:
Performance goals
Measurements of performance
Resource utilization and performance
Maintenance of good performance
Related tasks:
Developing a basic approach to performance measurement and tuning
Quick start for acceptable performance on a small database

Maintenance of good performance

Performance is affected in some way by all system users: the database server administrator, the database administrator, the application designers, and the client application users.

The database server administrator usually coordinates the activities of all users to ensure that system performance meets overall expectations. For example, the operating-system administrator might need to reconfigure the operating system to increase the amount of shared memory. Bringing down the operating system to install the new configuration requires bringing the database server down. The database server administrator must schedule this downtime and notify all affected users when the system will be unavailable.

The database server administrator should:

- Be aware of all performance-related activities that occur.
- Educate users about the importance of performance, how performance-related activities affect them, and how they can assist in achieving and maintaining optimal performance.

The database administrator should pay attention to:

- How tables and queries affect the overall performance of the database server
- The placement of tables and fragments
- How the distribution of data across disks affects performance

Application developers should:

- Carefully design applications to use the concurrency and sorting facilities that the database server provides, rather than attempt to implement similar facilities in the application.
- Keep the scope and duration of locks to the minimum to avoid contention for database resources.
- Include routines within applications that, when temporarily enabled at runtime, allow the database server administrator to monitor response times and transaction throughput.

Database users should:

- Pay attention to performance and report problems to the database server administrator promptly.
- Be courteous when they schedule large, decision-support queries and request as few resources as possible to get the work done.

Related concepts:
Performance goals
Measurements of performance
Resource utilization and performance
Factors that affect resource utilization

Related tasks:
Developing a basic approach to performance measurement and tuning
Quick start for acceptable performance on a small database

Performance monitoring and the tools you use

You can use performance monitoring tools to create a performance history, to monitor database resources at scheduled times, or to monitor ongoing transaction or query performance.

This chapter also contains cross-references to topics that about how to interpret the results of performance monitoring

The kinds of data that you need to collect depend on the kinds of applications that you run on your system. The causes of performance problems on OLTP (online transaction processing) systems are different from the causes of problems on systems that are used primarily for DSS query applications. Systems with mixed use provide a greater performance-tuning challenge and require a sophisticated analysis of performance-problem causes.

- Evaluate the current configuration
  Before you begin to adjust the configuration of your database server, evaluate the performance of your current configuration. You can view the contents of your configuration file with **onstat** commands.
- Create a performance history
  As soon as you set up your database server and begin to run applications on it, you should begin scheduled monitoring of resource use. As you accumulate data, you can analyze performance information.
- Monitor database server resources
  Monitor specific database server resources to identify performance bottlenecks and potential trouble spots and to improve resource use and response time.
- Monitor transactions
  You can use the **onlog** and **onstat** utilities to monitor transactions.
- Monitor sessions and queries
  Monitoring sessions and threads is important for sessions that perform queries as well as sessions that perform inserts, updates, and deletes. Some of the information that you can monitor for sessions and threads allows you to determine if an application is using a disproportionate amount of the resources.

Evaluate the current configuration

Before you begin to adjust the configuration of your database server, evaluate the performance of your current configuration. You can view the contents of your configuration file with **onstat** commands.

To alter certain database server characteristics, you must bring down the database server, which can affect your production system. Some configuration adjustments can unintentionally decrease performance or cause other negative side effects.

If your database applications satisfy user expectations, avoid frequent adjustments, even if those adjustments might theoretically improve performance. If your users are reasonably satisfied, take a measured approach to reconfiguring the database server. When possible, use a test instance of the database server to evaluate configuration changes before you reconfigure your production system.

When performance problems relate to backup operations, you might also examine the number or transfer rates for tape drives. You might need to alter the layout or fragmentation of your tables to reduce the impact of backup operations. For information about disk layout and table fragmentation, see Table performance considerations and Indexes and index performance considerations.

For client/server configurations, consider network performance and availability. Evaluating network performance is beyond the scope of this publication. For information about monitoring network activity and improving network availability, see your network administrator or see the documentation for your networking package.

Determine whether you want to set the configuration parameters that help maintain server performance by automatically adjusting properties of the database server while it is running, for example:

- AUTO_AIOVPS: Adds AIO virtual processors when I/O workload increases.
- AUTO_CKPTS: Increases the frequency of checkpoints to avoid transaction blocking.
- AUTO_LRU_TUNING: Manages cached data flushing as the server load changes.
- AUTO_READAHEAD: Changes the automatic read-ahead mode or disables automatic read-ahead operations for a query.
- AUTO_REPREPARE: Reoptimizes SPL routines and reprepares prepared objects after a schema change.
- AUTO_STAT_MODE: Enables or disables the mode for selectively updating only stale or missing data distributions in UPDATE STATISTICS operations.
- AUTO_TUNE: Enables or disables all automatic tuning configuration parameters that have values that are not present in your configuration file.
- DYNAMIC_LOGS: Allocates additional log files when necessary.
- LOCKS: Allocates additional locks when necessary.
- RTO_SERVER_RESTART: Provides the best performance possible while meeting the recovery time objective after a problem.

Related concepts:
Create a performance history
Monitor database server resources
Monitor transactions
Monitor sessions and queries
Related information:
onstat -c command: Print ONCONFIG file contents
onstat -g cfg command: Print the current values of configuration parameters

Create a performance history

As soon as you set up your database server and begin to run applications on it, you should begin scheduled monitoring of resource use. As you accumulate data, you can analyze performance information.

To accumulate data for performance analysis, use the command-line utilities described in Database server tools and Operating-system tools in operating scripts or batch files.

- The importance of a performance history
  If you have a history of the performance of your system, you can begin to track the cause of problems as soon as users report slow response or inadequate throughput.
- Tools that create a performance history
  When you monitor database server performance, you use tools from the host operating system and command-line utilities that you can run at regular intervals from scripts or batch files.

Related concepts:
Evaluate the current configuration
Monitor database server resources
Monitor transactions
Monitor sessions and queries

The importance of a performance history

If you have a history of the performance of your system, you can begin to track the cause of problems as soon as users report slow response or inadequate throughput.

If a history is not available, you must start tracking performance after a problem arises, and you might not be able to tell when and how the problem began. Trying to identify problems after the fact significantly delays resolution of a performance problem.

To build a performance history and profile of your system, take regular snapshots of resource-utilization information.

For example, if you chart the CPU utilization, paging-out rate, and the I/O transfer rates for the various disks on your system, you can begin to identify peak-use levels, peak-use intervals, and heavily loaded resources.

If you monitor fragment use, you can determine whether your fragmentation scheme is correctly configured. Monitor other resource use as appropriate for your database server configuration and the applications that run on it.

Choose tools from those described in the following sections, and create jobs that build up a history of disk, memory, I/O, and other database server resource use. To help you decide which tools to use to create a performance history, this chapter briefly describes the output of each tool.

Tools that create a performance history

When you monitor database server performance, you use tools from the host operating system and command-line utilities that you can run at regular intervals from scripts or batch files.

You also use performance monitoring tools with a graphical interface to monitor critical aspects of performance as queries and transactions are performed.

- Operating-system tools
  The database server relies on the operating system of the host computer to provide access to system resources such as the CPU, memory, and various unbuffered disk I/O interfaces and files. Each operating system has its own set of utilities for reporting how system resources are used.
- Database server tools
  The database server provides tools and utilities that capture snapshot information about your configuration and performance.

Operating-system tools

The database server relies on the operating system of the host computer to provide access to system resources such as the CPU, memory, and various unbuffered disk I/O interfaces and files. Each operating system has its own set of utilities for reporting how system resources are used.

Different implementations of some operating systems have monitoring utilities with the same name but different options and informational displays.

UNIX Only

The following table lists some UNIX utilities that monitor system resources.

UNIX Utility	Description

UNIX Utility	Description
vmstat utility	Displays virtual-memory statistics
iostat utility	Displays I/O utilization statistics
sar utility	Displays a variety of resource statistics
ps utility	Displays active process information

For details on how to monitor your operating-system resources, consult the reference manual or your system administration guide.

To capture the status of system resources at regular intervals, use scheduling tools that are available with your host operating system (for example, **cron**) as part of your performance monitoring system.

Windows Only

You can often use the Performance Logs and Alerts that the Windows operating system supplies to monitor resources such as processor, memory, cache, threads, and processes. The Performance Logs and Alerts also provide charts, alerts, reports, and the ability to save information to log files for later analysis.

For more information about how to use the Performance Logs and Alerts, consult your operating-system manuals.

Related reference:
Database server tools

Database server tools

The database server provides tools and utilities that capture snapshot information about your configuration and performance.

You can use these utilities regularly to build a historical profile of database activity, which you can compare with current operating-system resource-utilization data. These comparisons can help you discover which database server activities have the greatest impact on system-resource utilization. You can use this information to identify and manage your high-impact activities or adjust your database server or operating-system configuration.

The database server tools and utilities that you can use for performance monitoring include:

- The **onstat** utility
- The **onlog** utility
- The **oncheck** utility
- The **onperf** utility (on UNIX only)
- DB-Access and the system-monitoring interface (SMI), which you can use to monitor performance from within your application
- SQL administration API commands

You can use **onstat**, **onlog**, or **oncheck** commands invoked by the **cron** scheduling facility to capture performance-related information at regular intervals and build a historical performance profile of your database server application. The following sections describe these utilities.

You can use SQL SELECT statements to query the system-monitoring interface (SMI) from within your application.

The SMI tables are a collection of tables and pseudo-tables in the **sysmaster** database that contain dynamically updated information about the operation of the database server. The database server constructs these tables in memory but does not record them on disk. The **onstat** utility options obtain information from these SMI tables.

You can use **cron** and SQL scripts with DB-Access or **onstat** utility options to query SMI tables at regular intervals.
Tip: The SMI tables are different from the system catalog tables. System catalog tables contain permanently stored and updated information about each database and its tables (sometimes referred to as *metadata* or a *data dictionary*).
You can use **onperf** to display database server activity with the Motif window manager.

- Performance information that the onstat utility displays
  The **onstat** utility displays a wide variety of performance-related and status information contained within the SMI tables. You can use the **onstat** utility to check the current status of the database server and monitor the activities of the database server.

Related concepts:
The onperf utility on UNIX
Related reference:
Operating-system tools
Related information:
The onstat utility
The onlog utility
The oncheck Utility
DB-Access User's Guide
The System-Monitoring Interface Tables
System catalog tables
SQL administration API portal: Arguments by privilege groups

Performance information that the onstat utility displays

The **onstat** utility displays a wide variety of performance-related and status information contained within the SMI tables. You can use the **onstat** utility to check the current status of the database server and monitor the activities of the database server.

For a complete list of all **onstat** options, use the **onstat - -** command. For a complete display of all the information that **onstat** gathers, use the **onstat -a** command.
Tip: Profile information displayed by **onstat** commands, such as **onstat -p**, accumulates from the time the database server was started. To clear performance profile statistics so that you can create a new profile, run the **onstat -z**. If you use **onstat -z** to reset statistics for a performance history or appraisal, ensure that other users do not also enter the command at different intervals.
The following table lists some of the **onstat** commands that display general performance-related information.

Table 1. onstat commands that display performance information

onstat command	Description
onstat -p	Displays a performance profile that includes the number of reads and writes, the number of times that a resource was requested but was not available, and other miscellaneous information
onstat -b	Displays information about buffers currently in use
onstat -l	Displays information about the physical and logical logs
onstat -x	Displays information about transactions, including the thread identifier of the user who owns the transaction
onstat -u	Displays a user activity profile that provides information about user threads including the thread owner's session ID and login name
onstat -R	Displays information about buffer pools, including information about buffer pool page size.
onstat -F	Displays page-cleaning statistics that include the number of writes of each type that flushes pages to disk
onstat -g	Requires an additional argument that specifies the information to be displayed For example, **onstat -g mem** displays memory statistics.

For more information about options that provide performance-related information, see Monitoring fragmentation with the onstat -g ppf command and Monitor database server resources.

Related information:
onstat -g monitoring options

Monitor database server resources

Monitor specific database server resources to identify performance bottlenecks and potential trouble spots and to improve resource use and response time.

One of the most useful commands for monitoring system resources is **onstat -g** and its many options.

- Monitor resources that impact CPU utilization
  Threads, network communications, and virtual processors impact CPU utilization. You can use **onstat -g** arguments to monitor threads, network communications, and virtual processors.
- Monitor memory utilization
  You can use some specific **onstat -g** command options to monitor memory utilization.
- Monitor disk I/O utilization
  You can use some specific **onstat -g** arguments and the **oncheck** utility to determine if your disk I/O operations are efficient for your applications.

Related concepts:
Evaluate the current configuration
Create a performance history
Monitor transactions
Monitor sessions and queries

Monitor resources that impact CPU utilization

Threads, network communications, and virtual processors impact CPU utilization. You can use **onstat -g** arguments to monitor threads, network communications, and virtual processors.

Use the following **onstat -g** command options to monitor threads.

onstat -g Option	Description
act	Displays active threads.
ath	Displays all threads. The **sqlexec** threads represent portions of client sessions; the **rstcb** value corresponds to the user field of the **onstat -u** command.
cpu	Displays the last time the thread ran, how much CPU time the thread used, the number of times the thread ran, and other statistics about all the threads running in the server.
rea	Displays ready threads.
sle	Displays all sleeping threads.

onstat -g Option	Description
sts	Displays maximum and current stack use per thread.
tpf *tid*	Displays a thread profile for *tid*. If *tid* is 0, this argument displays profiles for all threads.
wai	Displays waiting threads, including all threads waiting on mutex or condition, or yielding.

Use the following **onstat -g** command options to monitor the network.

onstat -g Command Option	Description
ntd	Displays network statistics by service.
ntt	Displays network user times.
ntu	Displays network user statistics.
qst	Displays queue statistics.

Use the following **onstat -g** command options to monitor virtual processors.

onstat -g Command Option	Description
glo	Displays global multithreading information, including CPU-use information about virtual processors, the total number of sessions, and other multithreading global counters.
sch	Displays the number of semaphore operations, spins, and busy waits for each VP.
spi	Displays spin locks that are acquired by virtual processors after they have spun more than 10,000 times. To reduce contention, reduce the number of virtual processors, reduce the load on the computer, or, on some platforms, use the **no-age** or **processor affinity** options of virtual processors. If **sh_lock** mutexes have highly contended spin locks, create private memory caches for CPU virtual processors by setting the VP_MEMORY_CACHE_KB configuration parameter.
wst	Displays wait statistics.

Monitor memory utilization

You can use some specific **onstat -g** command options to monitor memory utilization.

Use the following **onstat -g** options to monitor memory utilization. For overall memory information, omit *table name*, *pool name*, or *session id* from the commands that permit those optional parameters.

Table 1. **onstat -g** Options for monitoring memory utilization

Argument	Description
ffr *pool name | session id*	Displays free fragments for a pool of shared memory or by session
dic *table name*	Displays one line of information for each table cached in the shared-memory dictionary If you provide a specific table name as a parameter, this argument displays internal SQL information about that table.
dsc	Displays one line of information for each column of distribution statistics cached in the data distribution cache.
mem *pool name | session id*	Displays memory statistics for the pools that are associated with a session If you omit *pool_name | session id*, this argument displays pool information for all sessions.
mgm	Displays Memory Grant Manager resource information, including: • The values of the PDQ configuration parameters • Memory and scan information • Load information, such as the number of queries that are waiting for memory, the number of queries that are waiting for scans, the number of queries that are waiting for queries with higher PDQ priority to run, and the number of queries that are waiting for a query slot • Active queries and the number of queries at each gate • Statistics on free resources • Statistics on queries • The resource/lock cycle prevention count, which shows the number of times the system immediately activated a query to avoid a potential deadlock
nsc *client id*	Displays shared-memory status by client ID If you omit *client id*, this argument displays all client status areas.
nsd	Displays network shared-memory data for poll threads
nss *session id*	Displays network shared-memory status by session id If you omit session id, this argument displays all session status areas.
osi	Displays information about your operating system resources and parameters, including shared memory and semaphore parameters, the amount of memory currently configured on the computer, and the amount of memory that is unused Use this option when the server is not online.

Argument	Description
prc	Displays one line of information for each user-defined routine (SPL routine or external routine written in C or Java™ programming language) cached in the UDR cache
seg	Displays shared-memory-segment statistics This argument shows the number and size of all attached segments.
ses *session id*	Displays memory usage for session id If you omit session id, this argument displays memory usage for all sessions.
ssc	Displays one line of information for each query cached in the SQL statement cache
stm *session id*	Displays memory usage of each SQL statement for session id If you omit session id, this argument displays memory usage for all sessions.
ufr *pool name* | *session id*	Displays allocated pool fragments by user or session

Related information:
onstat -g monitoring options

Monitor disk I/O utilization

You can use some specific **onstat -g** arguments and the **oncheck** utility to determine if your disk I/O operations are efficient for your applications.

- Using onstat -g to monitor I/O utilization
  You can use some specific **onstat -g** command arguments to monitor disk IO.
- Using the oncheck utility to monitor I/O utilization
  Disk I/O operations are usually the longest component of the response time for a query. You can use the **oncheck** Utility to monitor disk I/O operations.

Using onstat -g to monitor I/O utilization

You can use some specific **onstat -g** command arguments to monitor disk IO.

Use the following **onstat -g** command arguments to monitor disk I/O utilization.

onstat -g Argument	Description
iof	Displays asynchronous I/O statistics by chunk or file This argument is similar to the **onstat -d**, except that information about nonchunk files also appears. This argument displays information about temporary dbspaces and sort files.
iog	Displays asynchronous I/O global information
ioq	Displays asynchronous I/O queuing statistics
iov	Displays asynchronous I/O statistics by virtual processor

For a detailed case study that uses various **onstat** outputs, see Case studies and examples.

Related concepts:
Using the oncheck utility to monitor I/O utilization

Using the oncheck utility to monitor I/O utilization

Disk I/O operations are usually the longest component of the response time for a query. You can use the **oncheck** Utility to monitor disk I/O operations.

Contiguously allocated disk space improves sequential disk I/O operations, because the database server can read in larger blocks of data and use the read-ahead feature to reduce the number of I/O operations.

The **oncheck** utility displays information about storage structures on a disk, including chunks, dbspaces, blobspaces, extents, data rows, system catalog tables, and other options. You can also use **oncheck** to determine the number of extents that exist within a table and whether or not a table occupies contiguous space.

The **oncheck** utility provides the following options and information that apply to contiguous space and extents.

Option	Information
-pB	Blobspace simple large object (TEXT or BYTE data) For information about how to use this option to determine the efficiency of blobpage size, see Determine blobpage fullness with oncheck -pB output.
-pe	Chunks and extents For information about how to use this option to monitor extents, see Checking for extent interleaving and Eliminating interleaved extents.

Option	Information
-pk	Index key values. For information about how to improve the performance of this option, see Improving performance for index checks.
-pK	Index keys and row IDs For information about how to improve the performance of this option, see Improving performance for index checks.
-pl	Index-leaf key values For information about how to improve the performance of this option, see Improving performance for index checks.
-pL	Index-leaf key values and row IDs For information about how to improve the performance of this option, see Improving performance for index checks.
-pp	Pages by table or fragment For information about how to use this option to monitor space, see Considering the upper limit on extents.
-pP	Pages by chunk For information about how to use this option to monitor extents, see Considering the upper limit on extents.
-pr	Root reserved pages For information about how to use this option, see Estimating tables with fixed-length rows.
-ps	Space used by smart large objects and metadata in sbspace.
-pS	Space used by smart large objects and metadata in sbspace and storage characteristics For information about how to use this option to monitor space, see Monitoring sbspaces.
-pt	Space used by table or fragment For information about how to use this option to monitor space, see Estimating table size.
-pT	Space used by table, including indexes For information about how to use this option to monitor space, see Performance of in-place alters for DDL operations.

For more information about using **oncheck** to monitor space, see Estimating table size. For more information about concurrency during **oncheck** execution, see Improving performance for index checks.

Related concepts:
Using onstat -g to monitor I/O utilization
Related information:
The oncheck Utility

Monitor transactions

You can use the **onlog** and **onstat** utilities to monitor transactions.

- Using the onlog utility to monitor transactions
  The **onlog** utility displays all or selected portions of the logical log. This utility can help you identify a problematic transaction or gauge transaction activity that corresponds to a period of high utilization, as indicated by your periodic snapshots of database activity and system-resource consumption.
- Using the onstat utility to monitor transactions
  If the throughput of transactions is not very high, you can use some **onstat** utility commands to identify a transaction that might be a bottleneck.

Related concepts:
Evaluate the current configuration
Create a performance history
Monitor database server resources
Monitor sessions and queries

Using the onlog utility to monitor transactions

The **onlog** utility displays all or selected portions of the logical log. This utility can help you identify a problematic transaction or gauge transaction activity that corresponds to a period of high utilization, as indicated by your periodic snapshots of database activity and system-resource consumption.

This **onlog** utility can take input from selected log files, the entire logical log, or a backup tape of previous log files.

Use **onlog** with caution when you read logical-log files still on disk, because attempting to read unreleased log files stops other database activity. For greatest safety, back up the logical-log files first and then read the contents of the backup files. With proper care, you can use the **onlog -n** option to restrict **onlog** only to logical-log files that have been released.

To check on the status of logical-log files, use **onstat -l**.

Related information:
The onlog utility

Using the onstat utility to monitor transactions

If the throughput of transactions is not very high, you can use some **onstat** utility commands to identify a transaction that might be a bottleneck.

Use the following **onstat** utility commands to monitor transactions.

onstat command	Description
onstat -x	Displays transaction information such as number of locks held and isolation level.
onstat -u	Displays information about each user thread
onstat -k	Displays locks held by each session
onstat -g sql	Displays last SQL statement this session executed

Related information:
The onstat utility

Monitor sessions and queries

Monitoring sessions and threads is important for sessions that perform queries as well as sessions that perform inserts, updates, and deletes. Some of the information that you can monitor for sessions and threads allows you to determine if an application is using a disproportionate amount of the resources.

To monitor database server activity, you can view the number of active sessions and the amount of resources that they are using.

- Monitoring memory usage for each session
  You can use some specific **onstat -g** command arguments to get memory information for each session.
- Using the SET EXPLAIN statement
  You can use the SET EXPLAIN statement or the EXPLAIN directive to display the query plan that the optimizer creates for an individual query.

Related concepts:
Evaluate the current configuration
Create a performance history
Monitor database server resources
Monitor transactions

Monitoring memory usage for each session

You can use some specific **onstat -g** command arguments to get memory information for each session.

Use the following command arguments to get memory information for each session.

onstat -g command argument	Description
ses	Displays one-line summaries of all active sessions
ses *session id*	Displays session information by *session id*
sql *session id*	Displays SQL information by session If you omit *session id*, this argument displays summaries of all sessions.
stm *session id*	Displays amount of memory used by each prepared SQL statement in a session If you omit *session id*, this argument displays information for all prepared statements.

For examples and discussions of session-monitoring command-line utilities, see Monitoring memory usage for each session and Monitor sessions and threads.

Related tasks:
Using the SET EXPLAIN statement

Using the SET EXPLAIN statement

You can use the SET EXPLAIN statement or the EXPLAIN directive to display the query plan that the optimizer creates for an individual query.

For more information, see Display the query plan.

Related reference:
Monitoring memory usage for each session

Effect of configuration on CPU utilization

The combination of operating-system and Informix® configuration parameters can affect CPU utilization. You can change the settings of the Informix configuration parameters that directly affect CPU utilization, and you can adjust the settings for different types of workloads.

Multiple database server instances that run on the same host computer perform poorly when compared with a single database server instance that manages multiple databases. Multiple database server instances cannot balance their loads as effectively as a single database server. Avoid multiple residency for production environments in which performance is critical.

- UNIX configuration parameters that affect CPU utilization
  Your database server distribution includes a machine notes file that contains recommended values for UNIX configuration parameters. Because the UNIX parameters affect CPU utilization, you should compare the values in the machine notes file with your current operating-system configuration.
- Windows configuration parameters that affect CPU utilization
  The Informix distribution includes a machine notes file that contains recommended values for Informix configuration parameters on Windows. Compare the values in this file with your current ONCONFIG configuration file settings.
- Configuration parameters and environment variables that affect CPU utilization
  Some configuration parameters and environment variables affect CPU utilization. You might need to adjust the settings of these parameters and variables when you consider methods of improving performance.
- Network buffer pools
  The sizes of buffers for TCP/IP connections affect memory and CPU utilization. Sizing these buffers to accommodate a typical request can improve CPU utilization by eliminating the need to break up requests into multiple messages.
- Virtual processors and CPU utilization
  While the database server is online, you can start and stop virtual processors (VPs) that belong to certain classes.
- Connections and CPU utilization
  Some applications have a large number of client/server connections. Opening and closing connections can consume a large amount of system CPU time.

UNIX configuration parameters that affect CPU utilization

Your database server distribution includes a machine notes file that contains recommended values for UNIX configuration parameters. Because the UNIX parameters affect CPU utilization, you should compare the values in the machine notes file with your current operating-system configuration.

The following UNIX parameters affect CPU utilization:

- Semaphore parameters
- Parameters that set the maximum number of open file descriptors
- Memory configuration parameters

- UNIX semaphore parameters
  Semaphores are kernel resources with a typical size of 1 byte each. Semaphores for the database server are in addition to any that you allocate for other software packages. You can set some UNIX semaphore parameters.
- UNIX file-descriptor parameters
  Some operating systems require you to specify a limit on the number of file descriptors that a process can have open at any one time. To specify this limit, use an operating-system configuration parameter, typically NOFILE, NOFILES, NFILE, or NFILES.
- UNIX memory configuration parameters
  The configuration of memory in the operating system can affect other resources, including CPU and I/O.

Related concepts:
Configuration parameters and environment variables that affect CPU utilization
Network buffer pools
Virtual processors and CPU utilization
Connections and CPU utilization
Related tasks:
Windows configuration parameters that affect CPU utilization

UNIX semaphore parameters

Semaphores are kernel resources with a typical size of 1 byte each. Semaphores for the database server are in addition to any that you allocate for other software packages. You can set some UNIX semaphore parameters.

Each instance of the database server requires the following semaphore sets:

- One set for each group of up to 100 virtual processors (VPs) that are started with the database server
- One set for each additional VP that you might add dynamically while the database server is running
- One set for each group of 100 or fewer user sessions connected through the shared-memory communication interface

Tip: For best performance, allocate enough semaphores for double the number of **ipcshm** connections that you expect. Use the NETTYPE configuration parameter to configure database server poll threads for this doubled number of connections.
Because utilities such as **onmode** use shared-memory connections, you must configure a minimum of two semaphore sets for each instance of the database server: one for the initial set of VPs and one for the shared-memory connections that database server utilities use. The SEMMNI operating-system configuration parameter typically

specifies the number of semaphore sets to allocate. For information about how to set semaphore-related parameters, see the configuration instructions for your operating system.

The SEMMSL operating-system configuration parameter typically specifies the maximum number of semaphores per set. Set this parameter to at least `100`.

Some operating systems require that you configure a maximum total number of semaphores across all sets, which the SEMMNS operating-system configuration parameter typically specifies. Use the following formula to calculate the total number of semaphores that each instance of the database server requires:

```
SEMMNS = init_vps + added_vps + (2 * shmem_users) + concurrent_utils
```

init_vps
: is the number of virtual processors (VPs) that are started with the database server. This number includes CPU, PIO, LIO, AIO, SHM, TLI, SOC, and ADM VPs. The minimum value is `15`.

added_vps
: is the number of VPs that you intend to add dynamically.

shmem_users
: is the number of shared-memory connections that you allow for this instance of the database server.

concurrent_utils
: is the number of concurrent database server utilities that can connect to this instance. It is suggested that you allow for a minimum of six utility connections: two for ON-Bar and four for other utilities such as **onstat**, and **oncheck**.

If you use software packages that require semaphores, the SEMMNI configuration parameter must include the total number of semaphore sets that the database server and your other software packages require. You must set the SEMMSL configuration parameter to the largest number of semaphores per set that any of your software packages require. For systems that require the SEMMNS configuration parameter, multiply SEMMNI by the value of SEMMSL to calculate an acceptable value.

Related concepts:
Configuring poll threads

UNIX file-descriptor parameters

Some operating systems require you to specify a limit on the number of file descriptors that a process can have open at any one time. To specify this limit, use an operating-system configuration parameter, typically NOFILE, NOFILES, NFILE, or NFILES.

The number of open file descriptors that each instance of the database server needs depends on the number of chunks in your database, the number of VPs that you run, and the number of network connections that your database server instance must support.

Use the following formula to calculate the number of file descriptors that your instance of the database server requires:

```
NFILES = (chunks * NUMBER_OF_AIO_VPS) + NUMBER_of_CPU_VPS + net_connections
```

chunks
: is the number of chunks to be configured.

net_connections
: is the number of network connections that you specify in either of the following places:

  - **sqlhosts** file
  - NETTYPE configuration entries

  Network connections include all but those specified as the **ipcshm** connection type.

Each open file descriptor is about the same length as an integer within the kernel. Allocating extra file descriptors is an inexpensive way to allow for growth in the number of chunks or connections on your system.

UNIX memory configuration parameters

The configuration of memory in the operating system can affect other resources, including CPU and I/O.

Insufficient physical memory for the overall system load can lead to thrashing, as Memory utilization describes. Insufficient memory for the database server can result in excessive buffer-management activity. For more information about configuring memory, see Configuring UNIX shared memory.

Windows configuration parameters that affect CPU utilization

The Informix® distribution includes a machine notes file that contains recommended values for Informix configuration parameters on Windows. Compare the values in this file with your current ONCONFIG configuration file settings.

Informix runs in the background. For best performance, give the same priority to foreground and background applications.

On Windows, to change the priorities of foreground and background applications, go to **Start > Settings > Control Panel,** open the **System** icon, and click the **Advanced Tab.** Select the **Performance Options** button and select either the **Applications** or **Background Services** radio button.

The configuration of memory in the operating system can impact other resources, including CPU and I/O. Insufficient physical memory for the overall system load can lead to thrashing, as Memory utilization describes. Insufficient memory for Informix can result in excessive buffer-management activity. When you set the **Virtual Memory** values in the **System** icon on the **Control Panel**, ensure that you have enough paging space for the total amount of physical memory.

Related concepts:
UNIX configuration parameters that affect CPU utilization
Configuration parameters and environment variables that affect CPU utilization
Network buffer pools
Virtual processors and CPU utilization
Connections and CPU utilization

Configuration parameters and environment variables that affect CPU utilization

Some configuration parameters and environment variables affect CPU utilization. You might need to adjust the settings of these parameters and variables when you consider methods of improving performance.

The following configuration parameters in the database server configuration file have a significant impact on CPU utilization:

- DS_MAX_QUERIES
- DS_MAX_SCANS
- FASTPOLL
- MAX_PDQPRIORITY
- MULTIPROCESSOR
- NETTYPE
- OPTCOMPIND
- SINGLE_CPU_VP
- VPCLASS
- VP_MEMORY_CACHE_KB

The following environment variables affect CPU utilization:

- **OPTCOMPIND**
- **PDQPRIORITY**
- **PSORT_NPROCS**

The **OPTCOMPIND** environment variable, when set in the environment of a client application, indicates the preferred way to perform join operations. This variable overrides the value that the OPTCOMPIND configuration parameter sets. For details on how to select a preferred join method, see Optimizing access methods.

The **PDQPRIORITY** environment variable, when set in the environment of a client application, places a limit on the percentage of CPU VP utilization, shared memory, and other resources that can be allocated to any query that the client starts.

A client can also use the SET PDQPRIORITY statement in SQL to set a value for PDQ priority. The actual percentage allocated to any query is subject to the factor that the MAX_PDQPRIORITY configuration parameter sets. For more information about how to limit resources that can be allocated to a query, see Limiting PDQ resources in queries.

PSORT_NPROCS, when set in the environment of a client application, indicates the number of parallel sort threads that the application can use. The database server imposes an upper limit of 10 sort threads per query for any application. For more information about parallel sorts and **PSORT_NPROCS**, see Configure dbspaces for temporary tables and sort files.

- Specifying virtual processor class information
  Use the VPCLASS configuration parameter to specify a class of virtual processors, the number of virtual processors that the database server should start for a specific class, and the maximum number allowed.
- Setting the MULTIPROCESSOR configuration parameter when using multiple CPU VPs
  If you are running multiple CPU VPs, set the MULTIPROCESSOR configuration parameter to `1`. When you set MULTIPROCESSOR to `1`, the database server performs locking in a manner that is appropriate for a multiprocessor. Otherwise, set this parameter to `0`.
- Setting the SINGLE_CPU_VP configuration parameter when using one CPU VP
  If you are running only one CPU VP, set the SINGLE_CPU_VP configuration parameter to `1`. Otherwise, set this parameter to `0`.
- Optimizing access methods
  The OPTCOMPIND configuration parameter helps the query optimizer choose an appropriate access method for your application. When the optimizer examines join plans, OPTCOMPIND indicates the preferred method for performing the join operation for an ordered pair of tables.
- Limiting PDQ resources in queries
  The MAX_PDQPRIORITY configuration parameter limits the percentage of parallel database query (PDQ) resources that a query can use. Use MAX_PDQPRIORITY to limit the impact of large CPU-intensive queries on transaction throughput.
- Limiting the performance impact of CPU-intensive queries
  The DS_MAX_QUERIES configuration parameter specifies a maximum number of decision-support queries that can run at any one time. Queries with a low PDQ priority use proportionally fewer resources, so a larger number of those queries can run simultaneously. You can use the DS_MAX_QUERIES configuration parameter to limit the performance impact of CPU-intensive queries.
- Limiting the number of PDQ scan threads that can run concurrently
  The DS_MAX_SCANS configuration parameter limits the number of PDQ scan threads that can run concurrently. This configuration parameter prevents the database server from being flooded with scan threads from multiple decision-support queries.
- Configuring poll threads
  The NETTYPE configuration parameter configures poll threads for each connection type that your instance of the database server supports. If your database server instance supports connections over more than one interface or protocol, you must specify a separate NETTYPE configuration parameter for each connection type.
- Enabling fast polling
  You can use the FASTPOLL configuration parameter to enable or disable fast polling of your network, if your operating-system platform supports fast polling. Fast polling is beneficial if you have a large number of connections.

Related concepts:

UNIX configuration parameters that affect CPU utilization
Network buffer pools
Virtual processors and CPU utilization
Connections and CPU utilization
Related tasks:
Windows configuration parameters that affect CPU utilization
Related information:
Database configuration parameters
Environment variables

Specifying virtual processor class information

Use the VPCLASS configuration parameter to specify a class of virtual processors, the number of virtual processors that the database server should start for a specific class, and the maximum number allowed.

To execute user-defined routines (UDRs), you can define a new class of virtual processors to isolate UDR execution from other transactions that execute on the CPU virtual processors. Typically you write user-defined routines to support user-defined data types.

If you do not want a user-defined routine to affect the normal processing of user queries in the CPU class, you can use the CREATE FUNCTION statement to assign the routine to a user-defined class of virtual processors. The class name that you specify in the VPCLASS configuration parameter must match the name specified in the CLASS modifier of the CREATE FUNCTION statement.

For guidelines, on using the **cpu** and **num** options of the VPCLASS configuration parameter, see Setting the number of CPU VPs.

- Setting the number of CPU VPs
  You can configure the number of CPU virtual processors (VPs) that the database server uses. Do not allocate more CPU VPs than there are CPU processors available to service them.
- Disabling process priority aging for CPU VPs
  Use the **noage** option of the VPCLASS configuration parameter to disable process priority aging for database server CPU VPs on operating systems that support this feature. Priority aging occurs when the operating system lowers the priority of long-running processes as they accumulate processing time. You might want to disable priority aging because it can cause the performance of the database server processes to decline over time.
- Specifying processor affinity
  Use the **aff** option of the VPCLASS parameter to specify the processors to which you want to bind CPU VPs or AIO VPs. When you assign a CPU VP to a specific CPU, the VP runs only on that CPU. However, other processes can also run on that CPU.
- Setting the number of AIO VPs
  Use the **aio** and **num** options of the VPCLASS configuration parameter to indicate the number of AIO virtual processors that the database server starts initially.

Related information:
VPCLASS configuration parameter
CREATE FUNCTION statement

Setting the number of CPU VPs

You can configure the number of CPU virtual processors (VPs) that the database server uses. Do not allocate more CPU VPs than there are CPU processors available to service them.

When the database server starts, the number of CPU VPs is automatically increased to half the number of CPU processors on the database server computer, unless the SINGLE_CPU_VP configuration parameter is enabled. However, you might want to change the number of CPU VPs based on your performance needs.

You can enable the database server to add CPU VPs as needed, up to the number of CPU processors on the computer. Include the **autotune=1** option in the VPCLASS setting:

```
VPCLASS cpu,autotune=1
```

If you do not set the VPCLASS configuration parameter to **autotune=1**, use the following guidelines to set the number of CPU VPs.

Uniprocessor computers
: For uniprocessor computers, specify one CPU VP:

  ```
  VPCLASS cpu,num=1
  ```

Dual-processor computers
: For dual-processor systems, you might improve performance by running with two CPU VPs. To test if performance improves, set the **num** field of the VPCLASS configuration parameter to 1 in the onconfig file and then add a CPU VP dynamically at run time by running the **onmode -p** command.

Multiprocessor computers that are primarily database servers
: For multiprocessor systems with four or more CPUs that are primarily used as database servers, set the **num** option of the VPCLASS configuration parameter in the onconfig file to one less than the total number of processors. For example, if you have four CPUs, use the following specification:

  ```
  VPCLASS cpu,num=3
  ```

  When you use this setting, one processor is available to run the database server utilities or the client application.

Multiprocessor computers that are not primarily database servers
: For multiprocessor systems that you do not use primarily to support database servers, you can start with somewhat fewer CPU VPs to allow for other activities on the system and then gradually add more if necessary.

Multi-core or hardware multithreading computers with logical CPUs

For multiprocessor systems that use multi-core processors or hardware multithreading to support more logical CPUs than physical processors, you can assign the number of CPU VPs according to the number of logical CPU VPs available for that purpose. The amount of processing that an additional logical CPU can provide might be only a fraction of what a dedicated physical processor can support.

On systems, where multi-core processors are installed, the optimal configuration in most cases is the same as for systems with a number of individual processors equal to the total number of cores. Setting the number of CPU VPs to N-1, where N is number of cores is close to optimal for CPU-intensive workloads.

On computers where the CPU uses multiple threads per core, operating systems show more logical processors than actual processing cores. To take advantage of more CPU threads, the database server must be configured with the number of CPU VPs in the range between N and M, where N is number of cores and M is total number of logical CPUs reported by system. The number of CPU VPs where optimal performance is achieved depends on the workload.

When increasing the number of CPU VPs to use more threads per core, the expected gain in performance is only a fraction of what dedicated physical processor or core can provide.

If you are migrating Informix® from multi-CPU/multicore systems to systems with multiple threads per core, take special care in regard to processor affinity. When binding Informix CPU VPs to the logical processors of the operating system, you must be aware of the architecture for the CPU. If you are not sure, do not use the CPU affinity so that the operating system schedules CPU VPs to logical processors with available resources. Using affinity without understanding the relationship between the logical CPUs and processing cores can result in severe performance degradation.

For example, to bind each of 8 configured CPU VPs to a separate core on an 8-core system with two threads per core (16 logical CPUs), use the following setting:

```
VPCLASS cpu,num=8,aff=(0-14/2)
```

Related information:
VPCLASS configuration parameter

Disabling process priority aging for CPU VPs

Use the **noage** option of the VPCLASS configuration parameter to disable process priority aging for database server CPU VPs on operating systems that support this feature. Priority aging occurs when the operating system lowers the priority of long-running processes as they accumulate processing time. You might want to disable priority aging because it can cause the performance of the database server processes to decline over time.

Your database server distribution includes a machine notes file that contains information about whether your version of the database server supports this feature.

Specify the **noage** option of VPCLASS if your operating system supports this feature.

Related information:
VPCLASS configuration parameter

Specifying processor affinity

Use the **aff** option of the VPCLASS parameter to specify the processors to which you want to bind CPU VPs or AIO VPs. When you assign a CPU VP to a specific CPU, the VP runs only on that CPU. However, other processes can also run on that CPU.

The database server supports automatic binding of CPU VPs to processors on multiprocessor host computers that support processor affinity. Your database server distribution includes a machine notes file that contains information about whether your version of the database server supports this feature.

You can use processor affinity for the purposes that the following sections describe.

- Distributing computation impact
  You can use processor affinity to distribute the computation impact of CPU virtual processors (VPs) and other processes. On computers that are dedicated to the database server, assigning CPU VPs to all but one of the CPUs achieves maximum CPU utilization.
- Isolating AIO VPs from CPU VPs
  On a system that runs database server and client (or other) applications, you can bind asynchronous I/O (AIO) VPs to the same CPUs to which you bind other application processes through the operating system. In this way, you isolate client applications and database I/O operations from the CPU VPs.
- Avoiding a certain CPU
  The database server assigns CPU VPs to CPUs serially, starting with the CPU number you specify in this parameter. You might want to avoid assigning CPU VPs to a certain CPU that has a specialized hardware or operating-system function (such as interrupt handling).

Related information:
VPCLASS configuration parameter

Distributing computation impact

You can use processor affinity to distribute the computation impact of CPU virtual processors (VPs) and other processes. On computers that are dedicated to the database server, assigning CPU VPs to all but one of the CPUs achieves maximum CPU utilization.

On computers that support both database server and client applications, you can bind applications to certain CPUs through the operating system. By doing so, you effectively reserve the remaining CPUs for use by database server CPU VPs, which you bind to the remaining CPUs with the VPCLASS configuration parameter. Set the **aff**

option of the VPCLASS configuration parameter to the numbers of the CPUs on which to bind CPU VPs. For example, the following VPCLASS setting assigns CPU VPs to processors 4 to 7:

```
VPCLASS cpu,num=4,aff=(4-7)
```

When specifying a range of processors, you can also specify an incremental value with the range that indicates which CPUs in the range should be assigned to the virtual processors. For example, you can specify that the virtual processors are assigned to every other CPU in the range 0-6, starting with CPU 0.

```
VPCLASS CPU,num=4,aff=(0-6/2)
```

The virtual processors are assigned to CPUs 0, 2, 4, 6.
If you specify `VPCLASS CPU,num=4,aff=(1-10/3)`, the virtual processors are assigned to every third CPU in the range 1-10, starting with CPU 1. The virtual processors are assigned to CPUs 1, 4, 7, 10.

When you specify more than one value or range, the values and ranges do not have to be incremental or in any particular order. For example you can specify `aff=(8,12,7-9,0-6/2)`.

The database server assigns CPU virtual processors to CPUs in a circular pattern, starting with the first processor number that you specify in the *aff* option. If you specify a larger number of CPU virtual processors than physical CPUs, the database server continues to assign CPU virtual processors starting with the first CPU. For example, suppose you specify the following VPCLASS settings:

```
VPCLASS cpu,num=8,aff=(4-7)
```

The database server makes the following assignments:

- CPU virtual processor number `0` to CPU `4`
- CPU virtual processor number `1` to CPU `5`
- CPU virtual processor number `2` to CPU `6`
- CPU virtual processor number `3` to CPU `7`
- CPU virtual processor number `4` to CPU `4`
- CPU virtual processor number `5` to CPU `5`
- CPU virtual processor number `6` to CPU `6`
- CPU virtual processor number `7` to CPU `7`

Related information:
VPCLASS configuration parameter

Isolating AIO VPs from CPU VPs

On a system that runs database server and client (or other) applications, you can bind asynchronous I/O (AIO) VPs to the same CPUs to which you bind other application processes through the operating system. In this way, you isolate client applications and database I/O operations from the CPU VPs.

This isolation can be especially helpful when client processes are used for data entry or other operations that require waiting for user input. Because AIO VP activity usually comes in quick bursts followed by idle periods waiting for the disk, you can often interweave client and I/O operations without their unduly impacting each other.

Binding a CPU VP to a processor does not prevent other processes from running on that processor. Application (or other) processes that you do not bind to a CPU are free to run on any available processor. On a computer that is dedicated to the database server, you can leave AIO VPs free to run on any processor, which reduces delays on database operations that are waiting for I/O. Increasing the priority of AIO VPs can further improve performance by ensuring that data is processed quickly once it arrives from disk.

Avoiding a certain CPU

The database server assigns CPU VPs to CPUs serially, starting with the CPU number you specify in this parameter. You might want to avoid assigning CPU VPs to a certain CPU that has a specialized hardware or operating-system function (such as interrupt handling).

Setting the number of AIO VPs

Use the **aio** and **num** options of the VPCLASS configuration parameter to indicate the number of AIO virtual processors that the database server starts initially.

If your operating system does not support kernel asynchronous I/O (KAIO), the database server uses AIO virtual processors (VPs) to manage all database I/O requests.

If the VPCLASS configuration parameter does not specify the number of AIO VPs to start in the onconfig file, the number of AIO VPs initially started is equal to the number of chunks that use AIO, up to a maximum of 128.

You can enable the database server to increase the number of AIO VPs as needed to improve performance. Include the **autotune=1** option in the VPCLASS configuration parameter setting:

```
VPCLASS aio,autotune=1
```

The recommended number of AIO virtual processors depends on how many disks your configuration supports. If KAIO is not implemented on your platform, you should allocate one AIO virtual processor for each disk that contains database tables. You can add an additional AIO virtual processor for each chunk that the database server accesses frequently.

The machine notes file for your version of the database server indicates whether the operating system supports KAIO. If KAIO is supported, the machine notes describe how to enable KAIO on your specific operating system.

If your operating system supports KAIO, the CPU VPs make asynchronous I/O requests to the operating system instead of AIO virtual processors. In this case, configure only one AIO virtual processor, plus two additional AIO virtual processor for every file chunk that does not use KAIO.

If you use cooked files and if you enable direct I/O using the DIRECT_IO configuration parameter, you can reduce the number of AIO virtual processors. If the database server implements KAIO and if direct I/O is enabled, the database server will attempt to use KAIO, so you probably do not need more than one AIO virtual processor. Temporary dbspaces do not use direct I/O. If you have temporary dbspaces, you will probably need more than one AIO virtual processors.

Even when direct I/O is enabled with the DIRECT_IO configuration parameter, if the file system does not support either direct I/O or KAIO, you still must allocate two additional AIO virtual processors for every active dbspace chunk that is not using KAIO.

The goal in allocating AIO virtual processors is to allocate enough of them so that the lengths of the I/O request queues are kept short (that is, the queues have as few I/O requests in them as possible). When the I/O request queues remain consistently short, I/O requests are processed as fast as they occur. Use the **onstat -g ioq** command to monitor the length of the I/O queues for the AIO virtual processors.

Allocate enough AIO VPs to accommodate the peak number of I/O requests. Generally, allocating a few extra AIO VPs is not detrimental. To start additional AIO VPs while the database server is in online mode, use the **onmode -p** command. You cannot drop AIO VPs in online mode.

Related information:
AUTO_AIOVPS configuration parameter
VPCLASS configuration parameter

Setting the MULTIPROCESSOR configuration parameter when using multiple CPU VPs

If you are running multiple CPU VPs, set the MULTIPROCESSOR configuration parameter to `1`. When you set MULTIPROCESSOR to `1`, the database server performs locking in a manner that is appropriate for a multiprocessor. Otherwise, set this parameter to `0`.

The number of CPU VPs is used as a factor in determining the number of scan threads for a query. Queries perform best when the number of scan threads is a multiple (or factor) of the number of CPU VPs. Adding or removing a CPU VP can improve performance for a large query because it produces an equal distribution of scan threads among CPU VPs. For instance, if you have 6 CPU VPs and scan 10 table fragments, you might see a faster response time if you reduce the number of CPU VPs to 5, which divides evenly into 10. You can use **onstat -g ath** to monitor the number of scan threads per CPU VP or use **onstat -g ses** to focus on a particular session.

Related information:
MULTIPROCESSOR configuration parameter

Setting the SINGLE_CPU_VP configuration parameter when using one CPU VP

If you are running only one CPU VP, set the SINGLE_CPU_VP configuration parameter to `1`. Otherwise, set this parameter to `0`.

Important: If you set the SINGLE_CPU_VP parameter to *1*, the value of the `num` option of the VPCLASS configuration parameter must also be *1* .
Note: The database server treats user-defined virtual-processor classes (that is, VPs defined with VPCLASS) as if they were CPU VPs. Thus, if you set SINGLE_CPU_VP to nonzero, you cannot create any user-defined classes.
When you set the SINGLE_CPU_VP parameter to `1`, you cannot add CPU VPs while the database server is in online mode.

Related information:
SINGLE_CPU_VP configuration parameter
VPCLASS configuration parameter

Optimizing access methods

The OPTCOMPIND configuration parameter helps the query optimizer choose an appropriate access method for your application. When the optimizer examines join plans, OPTCOMPIND indicates the preferred method for performing the join operation for an ordered pair of tables.

If OPTCOMPIND is equal to `0`, the optimizer gives preference to an existing index (nested-loop join) even when a table scan might be faster. If OPTCOMPIND is set to `1` and the isolation level for a given query is set to Repeatable Read, the optimizer uses nested-loop joins.

When OPTCOMPIND is equal to `2`, the optimizer selects a join method based on cost alone even though table scans can temporarily lock an entire table. For more information about OPTCOMPIND and the different join methods, see Effect of OPTCOMPIND on the query plan.

To set the value for OPTCOMPIND for specific applications or user sessions, set the **OPTCOMPIND** environment variable for those sessions. Values for this environment variable have the same range and semantics as for the configuration parameter.

- Setting the value of OPTCOMPIND within a session
  You can set or change the value of OPTCOMPIND within a session for different kinds of queries. To do this, use the SET ENVIRONMENT OPTCOMPIND statement, not the OPTCOMPIND configuration parameter or the **OPTCOMPIND** environment variable.

Related information:
OPTCOMPIND configuration parameter

Setting the value of OPTCOMPIND within a session

You can set or change the value of OPTCOMPIND within a session for different kinds of queries. To do this, use the SET ENVIRONMENT OPTCOMPIND statement, not the OPTCOMPIND configuration parameter or the **OPTCOMPIND** environment variable.

For a DSS query, you should set the value of OPTCOMPIND to `2` or `1`, and you should be sure that the isolation level is not set to Repeatable Read. For an OLTP query, you could set the value to `0` or `1` with the isolation level not set to Repeatable Read.

The value that you enter using the SET ENVIRONMENT OPTCOMPIND command takes precedence over the default setting specified by the **OPTCOMPIND** environment variable or by the OPTCOMPIND configuration parameter in the **ONCONFIG** file. The default OPTCOMPIND setting is restored when the routine that issued the SET ENVIRONMENT OPTCOMPIND statement exits, or until the same routine resets the value of OPTCOMPIND to the system default by issuing the following statement:

```
SET ENVIRONMENT OPTCOMPIND DEFAULT;
```

No other user sessions or routines are affected by SET ENVIRONMENT OPTCOMPIND statements that you execute, because their scope is local to the routine in which they are issued, rather than the entire session.

Related information:
OPTCOMPIND session environment option

Limiting PDQ resources in queries

The MAX_PDQPRIORITY configuration parameter limits the percentage of parallel database query (PDQ) resources that a query can use. Use MAX_PDQPRIORITY to limit the impact of large CPU-intensive queries on transaction throughput.

To limit the impact of large CPU-intensive queries on transaction throughput

Set the value of the MAX_PDQPRIORITY configuration parameter to an integer that represents a percentage of the following PDQ resources that a query can request:

- Memory
- CPU VPs
- Disk I/O
- Scan threads

When a query requests a percentage of PDQ resources, the database server allocates the MAX_PDQPRIORITY percentage of the amount requested, as the following formula shows:

```
Resources allocated = PDQPRIORITY/100 * MAX_PDQPRIORITY/100
```

For example, if a client uses the SET PDQPRIORITY `80` statement to request 80 percent of PDQ resources, but MAX_PDQPRIORITY is set to `50`, the database server allocates only 40 percent of the resources (50 percent of the request) to the client.

For decision support and online transaction processing (OLTP), setting MAX_PDQPRIORITY allows the database server administrator to control the impact that individual decision-support queries have on concurrent OLTP performance. Reduce the value of MAX_PDQPRIORITY when you want to allocate more resources to OLTP processing. Increase the value of MAX_PDQPRIORITY when you want to allocate more resources to decision-support processing.

For more information about how to control the use of PDQ resources, see The allocation of resources for parallel database queries.

Related information:
MAX_PDQPRIORITY configuration parameter

Limiting the performance impact of CPU-intensive queries

The DS_MAX_QUERIES configuration parameter specifies a maximum number of decision-support queries that can run at any one time. Queries with a low PDQ priority use proportionally fewer resources, so a larger number of those queries can run simultaneously. You can use the DS_MAX_QUERIES configuration parameter to limit the performance impact of CPU-intensive queries.

The DS_MAX_QUERIES configuration parameter controls only queries with a PDQ priority that is nonzero.

The database server uses the value of DS_MAX_QUERIES with DS_TOTAL_MEMORY to calculate quantum units of memory to allocate to a query. For more information about how the database server allocates memory to queries, see The DS_TOTAL_MEMORY configuration parameter and memory utilization.

Related concepts:
The DS_TOTAL_MEMORY configuration parameter and memory utilization

Related information:
DS_MAX_QUERIES configuration parameter

Limiting the number of PDQ scan threads that can run concurrently

The DS_MAX_SCANS configuration parameter limits the number of PDQ scan threads that can run concurrently. This configuration parameter prevents the database server from being flooded with scan threads from multiple decision-support queries.

To calculate the number of scan threads allocated to a query, use the following formula:

```
scan_threads = min (nfrags, (DS_MAX_SCANS * pdqpriority / 100
   * MAX_PDQPRIORITY / 100) )
```

nfrags
is the number of fragments in the table with the largest number of fragments.
pdqpriority
is the PDQ priority value set by either the **PDQPRIORITY** environment variable or the SQL statement SET PDQPRIORITY.

Reducing the number of scan threads can reduce the time that a large query waits in the ready queue, particularly when many large queries are submitted concurrently. However, if the number of scan threads is less than *nfrags*, the query takes longer once it is underway.

For example, if a query needs to scan 20 fragments in a table, but the *scan_threads* formula lets the query begin when only 10 scan threads are available, each scan thread scans two fragments serially. Query execution takes approximately twice as long as if 20 scan threads were used.

Related information:
DS_MAX_SCANS configuration parameter

Configuring poll threads

The NETTYPE configuration parameter configures poll threads for each connection type that your instance of the database server supports. If your database server instance supports connections over more than one interface or protocol, you must specify a separate NETTYPE configuration parameter for each connection type.

You typically include a separate NETTYPE parameter for each connection type that is associated with a dbservername. You list dbservernames in the DBSERVERNAME and DBSERVERALIASES configuration parameters. You associate connection types with dbservernames in the sqlhosts information. For details about connection types and the sqlhosts information, see Connectivity configuration in your*IBM® Informix® Administrator's Guide*.

- Specifying the connection protocol
  The first NETTYPE entry, which specifies the protocol for a given connection type, applies to all dbservernames associated with that type. Subsequent NETTYPE entries for that connection type are ignored.
- Specifying virtual-processor classes for poll threads
  Each poll thread that is configured or added dynamically by a NETTYPE entry runs in a separate VP. A poll thread can run in one of two types of VP classes: NET (network) and CPU. Network VP classes include SOC, STR, SHM, and TLI. For best performance, use a NETTYPE entry to assign only one poll thread to the CPU VP class. Assign all additional poll threads to network VP classes by specifying NET in the NETTYPE configuration parameter values.
- Specifying the number of connections and poll threads
  The optimum number of connections per poll thread is approximately 300 for uniprocessor computers and up to 350 for multiprocessor computers, although this can vary depending on the platform and database server workload.
- Improve connection performance and scalability
  You can improve connection performance and scalability by specifying information in the NUMFDSERVERS and NS_CACHE configuration parameters and by using multiple listen threads.

Related reference:
UNIX semaphore parameters
Related information:
NETTYPE configuration parameter

Specifying the connection protocol

The first NETTYPE entry, which specifies the protocol for a given connection type, applies to all dbservernames associated with that type. Subsequent NETTYPE entries for that connection type are ignored.

NETTYPE entries are required for connection types that are used for outgoing communication only even if those connection types are not listed in the sqlhosts information.

UNIX Only
The following protocols apply to UNIX platforms:

- IPCSHM
- TLITCP
- IPCSTR

- SOCTCP
- TLIIMC
- SOCIMC
- SQLMUX
- SOCSSL

Windows Only
The following protocols apply to Windows platforms:

- SOCTCP
- IPCNMP
- SQLMUX
- SOCSSL

Related information:
NETTYPE configuration parameter

Specifying virtual-processor classes for poll threads

Each poll thread that is configured or added dynamically by a NETTYPE entry runs in a separate VP. A poll thread can run in one of two types of VP classes: NET (network) and CPU. Network VP classes include SOC, STR, SHM, and TLI. For best performance, use a NETTYPE entry to assign only one poll thread to the CPU VP class. Assign all additional poll threads to network VP classes by specifying NET in the NETTYPE configuration parameter values.

Related information:
NETTYPE configuration parameter

Specifying the number of connections and poll threads

The optimum number of connections per poll thread is approximately 300 for uniprocessor computers and up to 350 for multiprocessor computers, although this can vary depending on the platform and database server workload.

A poll thread can support 1024 or more connections. If the FASTPOLL configuration parameter is enabled, you might be able to configure fewer poll threads, but test the performance to determine the optimal configuration for your environment.

Each NETTYPE entry configures the number of poll threads for a specific connection type, the number of connections per poll thread, and the type of virtual-processor class in which those poll threads run. If the number of connections per thread exceeds 350 and the number of poll threads for the current connection type is less than the number of CPU VPs, you can improve performance by specifying the CPU VP class, adding poll threads (do not exceed the number of CPU VPs), and resetting the number of connections per thread. The default number of connections per thread is 50.

Important: Each ipcshm connection requires a semaphore. Some operating systems require that you configure a maximum number of semaphores that can be requested by all software packages that run on the computer. For best performance, double the number of actual ipcshm connections when you allocate semaphores for shared-memory communications. See UNIX semaphore parameters.
If your computer is a uniprocessor and your database server instance is configured for only one connection type, you can omit the NETTYPE parameter. The database server uses the information that is provided in the sqlhosts information to establish client/server connections.

If your computer is a uniprocessor and your database server instance is configured for more than one connection type, include a separate NETTYPE entry for each connection type. If the number of connections of any one type significantly exceeds 300, assign two or more poll threads, up to a maximum of the number of CPU VPs, and specify NET for a network VP class, as the following example shows:

```
NETTYPE ipcshm,1,50,CPU
NETTYPE tlitcp,2,200,NET # supports 400 connections
```

For **ipcshm**, the number of poll threads correspond to the number of memory segments. For example, if NETTYPE is set to `3,100` and you want one poll thread, set the poll thread to `1,300`.

If your computer is a multiprocessor, your database server instance is configured for only one connection type, and the number of connections does not exceed 350, you can use NETTYPE to specify a single poll thread on either the CPU or a network VP class. If the number of connections exceeds 350, set the VP class type to NET, increase the number of poll threads, and recalculate *conn_per_thread*.

Important: Carefully distinguish between poll threads for network connections and poll threads for shared memory connections, which run one per CPU virtual processor. Configure TCP connections to run in network virtual processors, and configure the minimum that is needed to maintain responsiveness. Configure shared memory connections to run in every CPU virtual processor.
Related concepts:
Improve connection performance and scalability
Related information:
NETTYPE configuration parameter
VPCLASS configuration parameter

Improve connection performance and scalability

You can improve connection performance and scalability by specifying information in the NUMFDSERVERS and NS_CACHE configuration parameters and by using multiple listen threads.

Informix® SQL sessions can migrate across CPU VPs. You can improve the performance and scalability of network connections on UNIX by using the NUMFDSERVERS configuration parameter to specify a number for the poll threads to use when distributing a TCP/IP connection across VPs. Specifying NUMFDSERVERS information is useful if the database server has a high rate of new connect and disconnect requests or if you find a high amount of contention between network shared file (NSF) locks.

You should also review and, if necessary, change the information in the NETTYPE configuration parameter, which defines the number of poll threads for a specific connection type, the number of connections per poll thread, and the virtual-processor class in which those poll threads run. You specify NETTYPE configuration parameter information as follows:

```
NETTYPE connection_type,poll_threads,conn_per_thread,vp_class
```

On UNIX, if `vp_class` is `NET`, `poll_threads` can be a value that is greater than or equal to 1. If `vp_class` is `CPU`, the number of `poll_threads` can be 1 through the number of CPU VPs. On Windows, `poll_threads` can be value that is greater than or equal to 1.

For example, suppose you specify 8 poll threads in the NETTYPE configuration parameter, as follows:

```
NETTYPE soctcp,8,300,NET
```

You can also specify 8 in the NUMFDSERVERS configuration parameter to enable the server to use all 8 poll thread to handle network connections migrating between VPs.

You can use the NS_CACHE configuration parameter to define the maximum retention time for an individual entry in the host name/IP address cache, the service cache, the user cache, and the group cache. The server can get information from the cache faster than it does when querying the operating system.

You can improve service for connection requests by using multiple listen threads. When you specify DBSERVERNAME and DBSERVERALIASES configuration parameter information for **onimcsoc** or **onsoctcp** protocols, you can specify the number of multiple listen threads for the database server aliases in your sqlhosts information. The default value of number is `1`.

The DBSERVERNAME and DBSERVERALIASES configuration parameters define database server names (dbservernames) that have corresponding entries in the sqlhosts information. Each dbservername parameter in the sqlhosts information has a **nettype** entry that specifies an interface/protocol combination. The database server runs one or more poll threads for each unique **nettype** entry.

You can use the **onstat -g ath** command to display information about all threads.

Related concepts:
Specifying the number of connections and poll threads
Monitor threads with onstat -g ath output
Related information:
NETTYPE configuration parameter
NUMFDSERVERS configuration parameter
NS_CACHE configuration parameter
DBSERVERNAME configuration parameter
DBSERVERALIASES configuration parameter
Multiple listen threads
Name service maximum retention time set in the NS_CACHE configuration parameter

Enabling fast polling

You can use the FASTPOLL configuration parameter to enable or disable fast polling of your network, if your operating-system platform supports fast polling. Fast polling is beneficial if you have a large number of connections.

For example, if you have more than 300 concurrent connections with the database server, you can enable the FASTPOLL configuration parameter for better performance.

Related information:
FASTPOLL configuration parameter

Network buffer pools

The sizes of buffers for TCP/IP connections affect memory and CPU utilization. Sizing these buffers to accommodate a typical request can improve CPU utilization by eliminating the need to break up requests into multiple messages.

However, you must use this capability with care; the database server dynamically allocates buffers of the indicated sizes for active connections. Unless you carefully size buffers, they can use large amounts of memory. For details on how to size network buffers, see Network buffer size.

The database server dynamically allocates network buffers from the global memory pool for request messages from clients. After the database server processes client requests, it returns buffers to a common network buffer pool that is shared among sessions that use SOCTCP, IPCSTR, or TLITCP network connections.

This common network buffer pool provides the following advantages:

- Prevents frequent allocations and deallocations from the global memory pool
- Uses fewer CPU resources to allocate and deallocate network buffers to and from the common network buffer pool for each network transfer
- Reduces contention for allocation and deallocation of shared memory

The free network buffer pool can grow during peak activity periods. To prevent large amounts of unused memory from remaining in these network buffer pools when network activity is no longer high, the database server returns free buffers when the number of free buffers reaches specific thresholds.

The database server provides the following features to further reduce the allocation and deallocation of and contention for the free network buffers:

- A private free network buffer pool for each session to prevent frequent allocations and deallocations of network buffers from the common network buffer pool or from the global memory pool in shared memory
- Capability to specify a larger than 4-kilobyte buffer size to receive network packets or messages from clients

As the system administrator, you can control the free buffer thresholds and the size of each buffer with the following methods:

- NETTYPE configuration parameter
- **IFX_NETBUF_PVTPOOL_SIZE** environment variable
- **IFX_NETBUF_SIZE** environment variable and b (client buffer size) option in the sqlhosts information

- Network buffers
  The database server implements a threshold of free network buffers to prevent frequent allocations and deallocations of shared memory for the network buffer pool. This threshold enables the database server to correlate the number of free network buffers with the number of connections that you specify in the NETTYPE configuration parameter.
- Support for private network buffers
  The database server provides support for private network buffers for each session that uses SOCTCP, IPCSTR, or TLITCP network connections.
- Network buffer size
  The **IFX_NETBUF_SIZE** environment variable specifies the size of each network buffer in the common network buffer pool and the private network buffer pool.

Related concepts:
UNIX configuration parameters that affect CPU utilization
Configuration parameters and environment variables that affect CPU utilization
Virtual processors and CPU utilization
Connections and CPU utilization
Related tasks:
Windows configuration parameters that affect CPU utilization

Network buffers

The database server implements a threshold of free network buffers to prevent frequent allocations and deallocations of shared memory for the network buffer pool. This threshold enables the database server to correlate the number of free network buffers with the number of connections that you specify in the NETTYPE configuration parameter.

The database server dynamically allocates network buffers for request messages from clients. After the database server processes client requests, it returns buffers to the network free-buffer pool.

If the number of free buffers is greater than the threshold, the database server returns the memory allocated to buffers over the threshold to the global pool.

The database server uses the following formula to calculate the threshold for the free buffers in the network buffer pool:

```
free network buffers threshold =
   100 + (0.7 * number_connections)
```

The value for *number_connections* is the total number of connections that you specified in the third field of the NETTYPE entry for the different type of network connections (SOCTCP, IPCSTR, or TLITCP). This formula does not use the NETTYPE entry for shared memory (IPCSHM).

If you do not specify a value in the third field of the NETTYPE parameter, the database server uses the default value of 50 connections for each NETTYPE entry corresponding to the SOCTCP, TLITCP, and IPCSTR protocols.

Support for private network buffers

The database server provides support for private network buffers for each session that uses SOCTCP, IPCSTR, or TLITCP network connections.

For situations in which many connections and sessions are constantly active, these private network buffers have the following advantages:

- Less contention for the common network buffer pool
- Fewer CPU resources to allocate and deallocate network buffers to and from the common network buffer pool for each network transfer

The **IFX_NETBUF_PVTPOOL_SIZE** environment variable specifies the size of the private network buffer pool for each session. The default size is one buffer.

Use the **onstat** utility commands in the following table to monitor the network buffer usage.

Command	Output Field	Description
onstat -g ntu	q-pvt	The current number and highest number of buffers that are free in the private pool for this session
onstat -g ntm	q-exceeds	The number of times that the free buffer threshold was exceeded

The **onstat -g ntu** command displays the following format for the **q-pvt** output field:

```
current number / highest number
```

If the number of free buffers (value in **q-pvt** field) is consistently 0, you can perform one of the following actions:

- Increase the number of buffers with the environment variable **IFX_NETBUF_PVTPOOL_SIZE**.

- Increase the size of each buffer with the environment variable **IFX_NETBUF_SIZE**.

The **q-exceeds** field indicates the number of times that the threshold for the shared network free-buffer pool was exceeded. When this threshold is exceeded, the database server returns the unused network buffers (over this threshold) to the global memory pool in shared memory. Optimally, this value should be 0 or a low number so that the server is not allocating or deallocating network buffers from the global memory pool.

Related information:
IFX_NETBUF_PVTPOOL_SIZE environment variable (UNIX)
IFX_NETBUF_SIZE environment variable

Network buffer size

The **IFX_NETBUF_SIZE** environment variable specifies the size of each network buffer in the common network buffer pool and the private network buffer pool.

The default buffer size is 4 kilobytes.

The **IFX_NETBUF_SIZE** environment variable allows the database server to receive messages longer than 4 kilobytes in one system call. The larger buffer size reduces the amount of overhead required to receive each packet.

Increase the value of **IFX_NETBUF_SIZE** if you know that clients send greater than 4-kilobyte packets. Clients send large packets during any of the following situations:

- Loading a table
- Inserting rows greater than 4 kilobytes
- Sending simple large objects

The **b** option for **sqlhosts** allows the client to send and receive greater than 4 kilobytes. The value for the **sqlhosts** option should typically match the value for IFX_NETBUF_SIZE.

You can use the following **onstat** command to see the network buffer size:

```
onstat -g afr global | grep net
```

The **size** field in the output shows the network buffer size in bytes.

Related information:
Connectivity configuration
IFX_NETBUF_SIZE environment variable

Virtual processors and CPU utilization

While the database server is online, you can start and stop virtual processors (VPs) that belong to certain classes.

You can use **onmode -p** to start additional VPs for the following classes while the database server is online: CPU, AIO, PIO, LIO, SHM, TLI, and SOC. You can drop VPs of the CPU class only while the database server is online.

You should carefully distinguish between poll threads for network connections and poll threads for shared memory connections, which should run one per CPU virtual processor. TCP connections should only be in network virtual processors, and you should only have the minimum needed to maintain responsiveness. Shared memory connections should only be in CPU virtual processors and should run in every CPU virtual processor

- Adding virtual processors
  Whenever you add a network VP (SOC or TLI), you also add a poll thread. Every poll thread runs in a separate VP, which can be either a CPU VP or a network VP of the appropriate network type.
- Monitoring virtual processors
  Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity.
- Private memory caches
  Each CPU virtual processor (VP) or tenant VP can have a private memory cache to speed access time to memory blocks.

Related concepts:
UNIX configuration parameters that affect CPU utilization
Configuration parameters and environment variables that affect CPU utilization
Network buffer pools
Connections and CPU utilization
Related tasks:
Windows configuration parameters that affect CPU utilization

Adding virtual processors

Whenever you add a network VP (SOC or TLI), you also add a poll thread. Every poll thread runs in a separate VP, which can be either a CPU VP or a network VP of the appropriate network type.

Adding more VPs can increase the load on CPU resources, so if the NETTYPE value indicates that an available CPU VP can handle the poll thread, the database server assigns the poll thread to that CPU VP. If all the CPU VPs have poll threads assigned to them, the database server adds a second network VP to handle the poll thread.

Monitoring virtual processors

Monitor the virtual processors to determine if the number of virtual processors configured for the database server is optimal for the current level of activity.

To monitor virtual processors:

- Use command-line utilities, such as **onstat-g ioq** to view information. See Using some onstat-g commands to monitor virtual processors
- Use the AUTO_AIOVPS configuration parameter to enable the database server to automatically increase the number of AIO virtual processors and page-cleaner threads when the server detects that AIO virtual processors are not keeping up with the I/O workload.
- Query SMI tables. See Using SMI tables to monitor virtual processors.

- Using some onstat-g commands to monitor virtual processors
  You can use the **onstat-g glo**, **onstat-g rea**, and **onstat-g ioq** commands to monitor virtual processors.
- Using SMI tables to monitor virtual processors
  You can get information from system-monitoring interface (SMI) tables to use to monitor virtual processors.

Using some onstat-g commands to monitor virtual processors

You can use the **onstat-g glo**, **onstat-g rea**, and **onstat-g ioq** commands to monitor virtual processors.

- Monitor virtual processors with the onstat-g glo command
  Use the **onstat-g glo** command to display information about each virtual processor that is running and to display cumulative statistics for each virtual-processor class.
- Monitor virtual processors with the onstat-g rea command
  Use the **onstat-g rea** command to monitor the number of threads in the ready queue.
- Monitor virtual processors with the onstat-g ioq command
  Use the **onstat-g ioq** command to determine whether you need to allocate additional AIO virtual processors.

Monitor virtual processors with the onstat-g glo command

Use the **onstat-g glo** command to display information about each virtual processor that is running and to display cumulative statistics for each virtual-processor class.

The **onstat -g glo** command provides the following types of information:

- How many session threads that are running
- How often threads switch, yield, or need to spin many times to obtain a latch or resource
- The virtual processor classes that are running and how much time each class spent running
- The number of virtual processors that are running for each virtual processor class
- The virtual processors that are running and how much time each virtual processor spent running
- The efficiency of each virtual processor

Use the **onstat -g rea** command to determine whether you need to increase the number of virtual processors.

Related concepts:
Monitor virtual processors with the onstat-g rea command
Related information:
onstat -g glo command: Print global multithreading information

Monitor virtual processors with the onstat-g rea command

Use the **onstat-g rea** command to monitor the number of threads in the ready queue.

onstat-g rea displays this information:

- The **status** field in the output shows the value `ready` when the thread is in the ready queue.
- The **vp-class** output field shows the virtual processor class on which the thread executes.

If the number of threads in the ready queue is growing for a class of virtual processors (for example, the CPU class), you might have to add more of those virtual processors to your configuration.

Figure 1. **onstat-g rea** output

```
Ready threads:
tid      tcb      rstcb    prty     status                  vp-class    name

6        536a38   406464   4        ready                     3cpu     main_loop()
28       60cfe8   40a124   4        ready                     1cpu     onmode_mon
33       672a20   409dc4   2        ready                     3cpu     sqlexec
```

Related concepts:
Monitor virtual processors with the onstat-g glo command
Related information:
onstat -g rea command: Print ready threads

Monitor virtual processors with the onstat-g ioq command

Use the **onstat-g ioq** command to determine whether you need to allocate additional AIO virtual processors.

The **onstat-g ioq** command displays the length of the I/O queues under the column **len**, as the figure below shows. You can also see the maximum queue length (since the database server started) in the **maxlen** column. If the length of the I/O queue is growing, I/O requests are accumulating faster than the AIO virtual processors can process them. If the length of the I/O queue continues to show that I/O requests are accumulating, consider adding AIO virtual processors.
Figure 1. **onstat-g ioq** and **onstat -d** output

```
onstat -g ioq

AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
  adt   0      0      0        0        0        0        0
  msc   0      0      1       12        0        0        0
  aio   0      0      4       89       68        0        0
  pio   0      0      1        1        0        1        0
  lio   0      0      1       17        0       17        0
  kio   0      0      0        0        0        0        0
  gfd   3      0      3      254      242       12        0
  gfd   4      0     17      614      261      353        0


onstat -d
Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
a1de1d8  1        1        1        1        N        informix rootdbs
a1df550  2        1        2        1        N        informix space1
 2 active, 32,678 maximum
Chunks
address  chk/dbs offset   size     free     bpages   flags pathname
a1de320  1   1   0        75000    66447              PO-   /ix/root_chunk
a1df698  2   2   0        500      447                PO-   /ix//chunk1
 2 active, 32,678 maximum
```

Each chunk serviced by the AIO virtual processors has one line in the **onstat-g ioq** output, identified by the value `gfd` in the **q name** column. You can correlate the line in **onstat -g ioq** with the actual chunk because the chunks are in the same order as in the **onstat -d** output. For example, in the **onstat-g ioq** output, there are two `gfd` queues. The first `gfd` queue holds requests for **root_chunk** because it corresponds to the first chunk shown in the **onstat -d** output. Likewise, the second `gfd` queue holds requests for **chunk1** because it corresponds to the second chunk in the **onstat -d** output.

If the database server has a mixture of raw devices and cooked files, the `gfd` queues correspond only to the cooked files in **onstat -d** output.

Related information:
onstat -g ioq command: Print I/O queue information

Using SMI tables to monitor virtual processors

You can get information from system-monitoring interface (SMI) tables to use to monitor virtual processors.

You must connect to the **sysmaster** database to query the SMI tables. Query the **sysvpprof** SMI table to obtain information about the virtual processors that are currently running. This table contains the following columns.

Column	Description
vpid	ID number of the virtual processor
class	Class of the virtual processor
usercpu	Seconds of user CPU consumed
syscpu	Seconds of system CPU consumed

Private memory caches

Each CPU virtual processor (VP) or tenant VP can have a private memory cache to speed access time to memory blocks.

All memory allocations that are requested by threads in the database server are fulfilled by memory pools. When a memory pool has insufficient memory blocks to satisfy a memory allocation request, blocks are allocated from the global memory pool. Because all threads use the same global memory pool, contention can occur. Private memory caches allow each virtual processor to retain its own set of memory blocks that can be used to bypass the global memory pool. The initial allocation for private memory caches is from the global memory pool. When the blocks are freed, they are freed to the private memory cache on a specific virtual process. When a memory allocation is requested, the thread first checks whether the allocation can be satisfied by blocks in the private memory cache. Otherwise, the thread requests memory from the global memory pool.

To determine whether private memory caches might improve performance for your database server, run the **onstat -g spi** command and look for the **sh_lock** mutex. If **onstat -g spi** command output shows contention for the **sh_lock** mutex, try creating private memory caches.

You set the VP_MEMORY_CACHE_KB configuration parameter to enable private memory caches by specifying the initial combined size of all private memory caches. By default, the total size of private memory caches is limited to the size value of the VP_MEMORY_CACHE_KB configuration parameter. You can set the mode to DYNAMIC to allow the size of each private memory cache to increase or decrease automatically based on the workload of the associated VP. In dynamic mode, the total size of private memory caches can exceed the value of the VP_MEMORY_CACHE_KB configuration parameter, but cannot exceed the value of the SHMTOTAL configuration parameter.

You can view statistics about VP private memory caches by running the **onstat -g vpcache** command. You can view statistics about memory pools by running the **onstat -g mem** command.

Attention: If you have multiple VPs, private memory caches can increase the amount of memory that the database server uses.
Related information:
VP_MEMORY_CACHE_KB configuration parameter
onstat -g vpcache command: Print CPU virtual processor and tenant virtual processor private memory cache statistics
onstat -g mem command: Print pool memory statistics
onstat -g spi command: Print spin locks with long spins

Connections and CPU utilization

Some applications have a large number of client/server connections. Opening and closing connections can consume a large amount of system CPU time.

The following topics describe ways that you might be able to reduce the system CPU time required to open and close connections.

- Multiplexed connections and CPU utilization
  Many traditional nonthreaded SQL client applications use multiple database connections to perform work for a single user. Each database connection establishes a separate network connection to the database server. The multiplexed connection facility provides the ability for one network connection in the database server to handle multiple database connections from a client application.
- MaxConnect for multiple connections UNIX
  IBM® Informix MaxConnect is a networking product for Informix database server environments on UNIX. You can use Informix MaxConnect to manage large numbers (from several hundred to tens of thousands) of client/server connections. Informix MaxConnect is best for OLTP data transfers, but is not recommended for large multimedia data transfers.

Related concepts:
UNIX configuration parameters that affect CPU utilization
Configuration parameters and environment variables that affect CPU utilization
Network buffer pools
Virtual processors and CPU utilization
Related tasks:
Windows configuration parameters that affect CPU utilization

Multiplexed connections and CPU utilization

Many traditional nonthreaded SQL client applications use multiple database connections to perform work for a single user. Each database connection establishes a separate network connection to the database server. The multiplexed connection facility provides the ability for one network connection in the database server to handle multiple database connections from a client application.

Multiplexed connections enable the database server to create multiple database connections without consuming the additional computer resources that are required for additional network connections.

When a nonthreaded client uses a multiplexed connection, the database server still creates the same number of user sessions and user threads as with a nonmultiplexed connection. However, the number of network connections decreases when you use multiplexed connections. Instead, the database server uses a multiplex listener thread to allow the multiple database connections to share the same network connection.

To improve response time for nonthreaded clients, you can use multiplexed connections to execute SQL queries. The amount of performance improvement depends on the following factors:

- The decrease in total number of network connections and the resulting decrease in system CPU time
  The usual cause for a large amount of system CPU time is the processing of system calls for the network connection. Therefore, the maximum decrease in system CPU time is proportional to the decrease in the total number of network connections.

- The ratio of this decrease in system CPU time to the user CPU time
  If the queries are simple and use little user CPU time, you might experience a sizable reduction in response time when you use a multiplexed connection. But if the queries are complex and use a large amount of user CPU time, you might not experience a performance improvement.

  To get an idea of the amounts of system CPU time and user CPU times per virtual processor, use the **onstat -g glo** option.

To use multiplexed connections for a nonthreaded client application, you must take the following steps before you bring up the database server:

1. Define an alias using the DBSERVERALIASES configuration parameter. For example, specify:

   ```
   DBSERVERALIASES ids_mux
   ```

2. Add an SQLHOSTS entry for the alias using `sqlmux` as the **nettype** entry, which is the second column in the SQLHOSTS file. For example, specify:

   ```
   ids_mux  onsqlmux  ......
   ```

   The other fields in this entry, the **hostname** and **servicename**, must be present, but they are ignored.

3. Enable multiplexing for the selected connection types by specifying `m=1` in the **sqlhosts** file or registry that the client uses for the database server connection.
4. On Windows platforms, you must also set the **IFX_SESSION_MUX** environment variable.

Warning: On Windows, a multithreaded application must not use the multiplexed connection feature. If a multithreaded application enables the multiplexing option in the **sqlhosts** registry entry and also defines the IFX_SESSION_MUX environment variable, it can produce disastrous results, including crashing and data corruption.
Related information:
Multiplexed connections
Supporting multiplexed connections

MaxConnect for multiple connections UNIX

IBM® Informix® MaxConnect is a networking product for Informix database server environments on UNIX. You can use Informix MaxConnect to manage large numbers (from several hundred to tens of thousands) of client/server connections. Informix MaxConnect is best for OLTP data transfers, but is not recommended for large multimedia data transfers.

Informix MaxConnect provides the following performance advantages for medium to large OLTP configurations:

- Reduces CPU requirements on the database server by reducing the number of physical connections.
  Informix MaxConnect multiplexes connections so that the ratio of client connections to database connections can be 100:1 or higher.
- Improves end-user response time by increasing system scalability to many thousands of connections
- Reduces operating-system overhead by aggregating multiple small packets into one transfer operation

To obtain maximum performance benefit, install Informix MaxConnect on either a dedicated computer to which Informix clients connect or on the client application server. Either of these configurations offloads the CPU requirements of handling a large number of connections from the database server computer.

To monitor Informix MaxConnect, use the **onstat -g imc** command on the database server computer and use the **imcadmin** command on the computer where Informix MaxConnect is located.

For more information about installing, configuring, monitoring, and tuning Informix MaxConnect, see the *IBM Informix MaxConnect User's Guide*.
Important: Informix MaxConnect and the *IBM Informix MaxConnect User's Guide* ship separately from IBM Informix.

Effect of configuration on memory utilization

The combination of operating-system and Informix® configuration parameters can affect memory utilization.

You can change the settings of the Informix configuration parameters that directly affect memory utilization, and you can adjust the settings for different types of workloads.

Consider the amount of physical memory that is available on your host when you allocate shared memory for the database server by setting operating-system configuration parameters. In general, if you increase space for database server shared memory, you can enhance the performance of your database server. You must balance the amount of shared memory that is dedicated to the database server against the memory requirements for VPs and other processes.

- Shared memory
  You must configure adequate shared-memory resources for the database server in your operating system. Insufficient shared memory can adversely affect performance.
- Configuration parameters that affect memory utilization
  A large number of configuration parameters in the ONCONFIG file affect memory utilization and performance.
- Configure and monitor memory caches
  The database server uses caches to store information in memory instead of performing a disk read or another operation to obtain the information. These memory caches improve performance for multiple queries that access the same tables. You can set some configuration parameters to increase the effectiveness of each cache. You can view information about memory caches by running **onstat** commands.
- Session memory
  The database server uses the virtual portion of shared memory mainly for user sessions. Most of the memory that each user session allocates is for SQL statements. You can determine which session and which statements are using large amounts of memory. If necessary, you can set the SESSION_LIMIT_MEMORY configuration parameter to limit the amount of memory available to a session.

- Data-replication buffers and memory utilization
  Data replication requires two instances of the database server, a primary one and a secondary one, running on two computers. If you implement data replication for your database server, the database server holds logical-log records in the data-replication buffer before it sends them to the secondary database server.
- Memory latches
  The database server uses latches to control access to shared memory structures such as the buffer pool or the memory pools for the SQL statement cache. You can obtain statistics on latch use and information about specific latches. These statistics provide a measure of the system activity.
- Encrypted values
  An encrypted value uses more storage space than the corresponding plain text value because all of the information needed to decrypt the value except the encryption key is stored with the value.

Related concepts:
The Memory Grant Manager

Shared memory

You must configure adequate shared-memory resources for the database server in your operating system. Insufficient shared memory can adversely affect performance.

The database server threads and processes require shared memory to share data by sharing access to segments of memory.

The shared memory that Informix® uses can be divided into the following parts, each of which has one or more shared memory segments:

- Resident portion
- Virtual portion
- Message portion
- Buffer pool portion

The resident and message portions are static; you must allocate sufficient memory for them before you bring the database server into online mode. (Typically, you must reboot the operating system to reconfigure shared memory.) The virtual portion of shared memory for the database server grows dynamically, but you must still include an adequate initial amount for this portion in your allocation of operating-system shared memory.

The amount of space that is required is the total that all portions of database server shared memory need. You specify the total amount of shared memory with the SHMTOTAL configuration parameter.

The LOCKS configuration parameter specifies the initial size of the lock table. If the number of locks that sessions allocate exceeds the value of LOCKS, the database server dynamically increases the size of the lock table. If you expect the lock table to grow dynamically, set SHMTOTAL to 0. When SHMTOTAL is 0, there is no limit on total memory (including shared memory) allocation.

- Resident portion of shared memory
  The resident portion of shared memory includes areas of shared memory that record the state of the database server, including locks, log files, and the locations of dbspaces, chunks, and tblspaces.
- Virtual portion of shared memory
  Informix uses the virtual portion of shared memory to allocate memory to each database server subsystem, as needed.
- Message portion of shared memory
  The message portion of shared memory contains the message buffers that the shared-memory communication interface uses. The amount of space required for these buffers depends on the number of user connections that you allow using a given networking interface.
- Buffer pool portion of shared memory
  The buffer pool portion of shared memory contains one or more buffer pools. Each page size that is used by a dbspace has a buffer pool.
- Estimating the size of the resident portion of shared memory
  You can use formulas to estimate the size of the resident portion (in KB) of shared memory when you allocate operating-system shared memory.
- Estimating the size of the virtual portion of shared memory
  You can use a formula to estimate the initial size of the virtual portion of shared memory. You specify the initial size in the SHMVIRTSIZE configuration parameter.
- Estimating the size of the message portion of shared memory
  You can estimate the size of the message portion of shared memory in kilobytes.
- Configuring UNIX shared memory
  On UNIX, you can configure shared-memory segments for the database server.
- Freeing shared memory with onmode -F
  You can run the **onmode -F** command to free shared-memory segments that are unavailable or no longer needed for a process.

Related information:
LOCKS configuration parameter
SHMTOTAL configuration parameter

Resident portion of shared memory

The resident portion of shared memory includes areas of shared memory that record the state of the database server, including locks, log files, and the locations of dbspaces, chunks, and tblspaces.

The settings that you use for the LOCKS, LOGBUFF, and PHYSBUFF configuration parameters help determine the size of the resident portion.

In addition to these configuration parameters, which affect the size of the resident portion, the RESIDENT configuration parameter can affect memory use. When a computer supports forced residency and the RESIDENT configuration parameter is set to a value that locks the resident or resident and virtual portions, the resident portion is never paged out.

The machine notes file for your database server indicates whether your operating system supports forced residency.

On AIX®, Solaris, and Linux systems that support large pages, the IFX_LARGE_PAGES environment variable can enable the use of large pages for non-message shared memory segments that are locked in physical memory. If large pages are configured by operating system commands and the RESIDENT configuration parameter specifies that some or all of the resident and virtual portions of shared memory are locked in physical memory, Informix® uses large pages for the corresponding shared memory segments, provided sufficient large pages are available. The use of large pages can offer significant performance benefits in large memory configurations.

Related reference:
Configuration parameters that affect memory utilization
Related information:
IFX_LARGE_PAGES environment variable

Virtual portion of shared memory

Informix® uses the virtual portion of shared memory to allocate memory to each database server subsystem, as needed.

The virtual portion of shared memory for the database server includes the following components:

- Large buffers, which are used for large read and write I/O operations
- Sort-space pools
- Active thread-control blocks, stacks, and heaps
- User-session data
- Caches for SQL statements, data-dictionary information, and user-defined routines
- A global pool for network-interface message buffers and other information

The SHMVIRTSIZE configuration parameter in the onconfig file provides the initial size of the virtual portion. As the need for additional space in the virtual portion arises, the database server adds shared memory in increments that the SHMADD configuration parameter specifies. The EXTSHMADD configuration parameter configures the size of the virtual-extension shared memory segments that are added for user-defined routines and DataBlade routines. The limit on the total shared memory allocated to the database server is specified by the SHMTOTAL parameter.

The size of the virtual portion depends primarily on the types of applications and queries that you are running. Depending on your application, an initial estimate for the virtual portion might be as low as 100 KB per user or as high as 500 KB per user, plus an additional 4 megabytes if you intend to use data distributions.

When a computer supports forced residency and the RESIDENT configuration parameter is set to a value that locks virtual segments, the virtual segments that are locked are never paged out.

On AIX®, Solaris, and Linux systems that support large pages, the IFX_LARGE_PAGES environment variable can enable the use of large pages for non-message shared memory segments that are locked in physical memory. If large pages are configured by operating system commands and the RESIDENT configuration parameter specifies that some or all of the resident and virtual portions of shared memory are locked in physical memory, Informix uses large pages for the corresponding shared memory segments, provided sufficient large pages are available. The use of large pages can offer significant performance benefits in large memory configurations.

Related tasks:
Creating data distributions
Related reference:
Configuration parameters that affect memory utilization
Related information:
IFX_LARGE_PAGES environment variable
EXTSHMADD configuration parameter
SHMADD configuration parameter
SHMTOTAL configuration parameter
SHMVIRTSIZE configuration parameter

Message portion of shared memory

The message portion of shared memory contains the message buffers that the shared-memory communication interface uses. The amount of space required for these buffers depends on the number of user connections that you allow using a given networking interface.

If a particular interface is not used, you do not need to include space for it when you allocate shared memory in the operating system.

Buffer pool portion of shared memory

The buffer pool portion of shared memory contains one or more buffer pools. Each page size that is used by a dbspace has a buffer pool.

The BUFFERPOOL configuration parameter specifies the size of the buffer pool when the database server is started. If the buffer pool is extendable, the database server increases the size of the buffer pool in the buffer pool portion of shared memory.

You can determine the current size of the buffer pool portion of shared memory by running the **onstat -g buf** command and adding the values in the **Total Mem** field for each buffer pool. For example, the following output shows that the memory for one buffer pool is 32 MB:

```
Fg Writes     LRU Writes    Avg. LRU Time Chunk Writes  Total Mem
0             0             nan           10883         32Mb
```

The maximum size of each buffer pool depends on the amount of available shared memory and the values of the BUFFERPOOL configuration parameters.

Related information:
Buffer pool portion of shared memory
BUFFERPOOL configuration parameter
onstat -g buf command: Print buffer pool profile information

Estimating the size of the resident portion of shared memory

You can use formulas to estimate the size of the resident portion (in KB) of shared memory when you allocate operating-system shared memory.

The result of your calculations is an estimate that normally, slightly exceeds the actual memory that is used for the resident portion of shared memory.

The following estimate was calculated to determine the resident portion of shared memory on a 64-bit server. The sizes that are shown are subject to change, and the calculation is approximate.

To estimate the size of the resident portion of shared memory

1. Calculate the values in the following formulas:

```
locks_value = LOCKS * 136
logbuff_value = LOGBUFF * 1024 * 3
physbuff_value = PHYSBUFF * 1024 * 2
```

2. Calculate the estimated size of the resident portion in KB, using the following formula:

```
rsegsize = 1.02 * (locks_value + logbuff_value
           + physbuff_value + 1,200,000) / 1024
```


Estimating the size of the virtual portion of shared memory

You can use a formula to estimate the initial size of the virtual portion of shared memory. You specify the initial size in the SHMVIRTSIZE configuration parameter.

The formula for estimating an initial size of the virtual portion of shared memory is as follows:

```
shmvirtsize = fixed overhead + shared structures +
              (mncs * private structures) +
              other buffers
```

To estimate an SHMVIRTSIZE value with the preceding formula:

1. Estimate the value for the `fixed overhead` portion of the formula as follows:

```
fixed overhead = global pool +
                 thread pool after booting
```

   a. Run the **onstat -g mem** command to obtain the pool sizes allocated to sessions.
   b. Subtract the value in the **freesize** field from the value in the **totalsize** to obtain the number of bytes allocated per session.
   c. Estimate a value for the *thread pool after booting* variable. This variable is partially dependent on the number of virtual processors.
2. Estimate the value of `shared structures` with the following formula:

```
shared structures = AIO vectors + sort memory +
              dbspace backup buffers +
              data-dictionary cache size +
              size of user-defined routine cache +
              histogram pool +
              STMT_CACHE_SIZE (SQL statement cache) +
              other pools (See onstat display.)
```

3. Estimate the next part of the formula, as follows:
   a. Estimate the value of `mncs` (which is the maximum number of concurrent sessions) with the following formula:

```
mncs = number of poll threads *
       number connections per poll thread
```

   The value for number of poll threads is the value that you specify in the second field of the NETTYPE configuration parameter.

   The value for *`number of connections per poll thread`* is the value that you specify in the third field of the NETTYPE configuration parameter.

   You can also obtain an estimate of the maximum number of concurrent sessions when you run the **onstat -u** command during peak processing. The last line of the **onstat -u** output contains the maximum number of concurrent user threads.

   b. Estimate the value of *`private structures`*, as follows:

```
private structures = stack + heap +
                     session control-block structures
```

   stack
   Generally 32 KB but dependent on recursion in user-defined routines. You can obtain the stack size for each thread with the **onstat -g sts** option.
   heap

About 15 KB. You can obtain the heap size for an SQL statement when you use the **onstat -g stm** option.

session control-block structures
The amount of memory used per session. The **onstat -g ses** option displays the amount of memory, in bytes, in the **total memory** column listed for each session id.

c. Multiply the results of steps 3a and 3b to obtain the following part of the formula:

```
mncs * private structures
```

4. Estimate the value of *other buffers* to account for private buffers allocated for features such as lightweight I/O operations for smart large objects (about 180 KB per user).
5. Add the results of steps 1 through 4 to obtain an estimate for the SHMVIRTSIZE configuration parameter.

Tip: When the database server is running with a stable workload, you can use **onstat -g seg** to obtain a precise value for the actual size of the virtual portion of shared memory. You can then use the value for shared memory that this command reports to reconfigure SHMVIRTSIZE.
To specify the size of segments that are added later to the virtual shared memory, set the SHMADD configuration parameter. Use the EXTSHMADD configuration parameter to specify the size of virtual-extension segments that are added for user-defined routines and DataBlade routines.

The following table contains a list of additional topics for estimating the size of shared structures in memory.

Table 1. Information for shared-memory structures

Shared-Memory Structure	More Information
Sort memory	Estimating memory needed for sorting
Data-dictionary cache	Data-dictionary configuration
Data-distribution cache (histogram pool)	Data-distribution configuration
User-defined routine (UDR) cache	SPL routine executable format stored in UDR cache
SQL statement cache	Enabling the SQL statement cache Monitor and tune the SQL statement cache
Other pools	To see how much memory is allocated to the different pools, use the **onstat -g mem** command.

Related concepts:
Session memory
Related information:
SHMVIRTSIZE configuration parameter
NETTYPE configuration parameter
onstat -g mem command: Print pool memory statistics

Estimating the size of the message portion of shared memory

You can estimate the size of the message portion of shared memory in kilobytes.

Estimate the size of the message portion of shared memory, using the following formula:

```
msegsize = (10,531 * ipcshm_conn + 50,000)/1024
```

ipcshm_conn
is the number of connections that can be made using the shared-memory interface, as determined by the NETTYPE parameter for the **ipcshm** protocol.

Related information:
NETTYPE configuration parameter

Configuring UNIX shared memory

On UNIX, you can configure shared-memory segments for the database server.

On UNIX, perform the following steps to configure the shared-memory segments that your database server configuration needs. For information about how to set parameters related to shared memory, see the configuration instructions for your operating system.

To configure shared-memory segments for the database server:

1. If your operating system does not have a size limit for shared-memory segments, take the following actions:
   a. Set the operating-system configuration parameter for maximum segment size, typically SHMMAX or SHMSIZE, to the total size that your database server configuration requires. This size includes the amount of memory that is required to start your database server instance and the amount of shared memory that you allocate for dynamic growth of the virtual portion.
   b. Set the operating-system configuration parameter for the maximum number of segments, typically SHMMNI, to at least `1` per instance of the database server.
2. If your operating system has a segment-size limit, take the following actions:
   a. Set the operating-system configuration parameter for the maximum segment size, typically SHMMAX or SHMSIZE, to the largest value that your system allows.
   b. Use the following formula to calculate the number of segments for your instance of the database server. If there is a remainder, round up to the nearest integer.

```
SHMMNI = total_shmem_size / SHMMAX
```

total_shmem_size
is the total amount of shared memory that you allocate for the database server use.

3. Set the operating-system configuration parameter for the maximum number of segments, typically SHMMNI, to a value that yields the total amount of shared memory for the database server when multiplied by SHMMAX or SHMSIZE. If your computer is dedicated to a single instance of the database server, that total can be up to 90 percent of the size of virtual memory (physical memory plus swap space).
4. If your operating system uses the SHMSEG configuration parameter to indicate the maximum number of shared-memory segments that a process can attach, set this parameter to a value that is equal to or greater than the largest number of segments that you allocate for any instance of the database server.

For additional tips on configuring shared memory in the operating system, see the machine notes file for UNIX or the release notes file for Windows.

Related concepts:
The SHMADD and EXTSHMADD configuration parameters and memory utilization

Freeing shared memory with onmode -F

You can run the **onmode -F** command to free shared-memory segments that are unavailable or no longer needed for a process.

The database server does not automatically free the shared-memory segments that it adds during its operations. After memory has been allocated to the database server virtual portion, the memory remains unavailable for use by other processes running on the host computer. When the database server runs a large decision-support query, it might acquire a large amount of shared memory. After the query completes, the database server no longer requires that shared memory. However, the shared memory that the database server allocated to service the query remains assigned to the virtual portion even though it is no longer needed.

The **onmode -F** command locates and returns unused 8-kilobyte blocks of shared memory that the database server still holds. Although this command runs only briefly (one or two seconds), **onmode -F** dramatically inhibits user activity while it runs. Systems with multiple CPUs and CPU VPs typically experience less degradation while this utility runs.

You should run **onmode -F** during slack periods with an operating-system scheduling facility (such as **cron** on UNIX). In addition, consider running this utility after you perform any task that substantially increases the size of database server shared memory, such as large decision-support queries, index builds, sorts, or backup operations.

Related information:
onmode -F: Free unused memory segments

Configuration parameters that affect memory utilization

A large number of configuration parameters in the ONCONFIG file affect memory utilization and performance.

The following configuration parameters significantly affect memory utilization:

- BUFFERPOOL
- DS_NONPDQ_QUERY_MEM
- DS_TOTAL_MEMORY
- EXTSHMADD
- LOCKS
- LOGBUFF
- LOW_MEMORY_MGR
- LOW_MEMORY_RESERVE
- PHYSBUFF
- RESIDENT
- SHMADD
- SHMBASE
- SHMTOTAL
- SHMVIRTSIZE
- SHMVIRT_ALLOCSEG
- STACKSIZE
- Memory cache parameters (see Configure and monitor memory caches)
- Network buffer size (see Network buffer pools)

The SHMBASE parameter indicates the starting address for database server shared memory. When set according to the instructions in the machine notes file or release notes file, this parameter has no appreciable effect on performance. For the path name of each file, see the Introduction to this guide.

The DS_NONPDQ_QUERY_MEM parameter increases the amount of memory that is available for non-PDQ queries. You can only use this parameter if PDQ priority is set to zero. For more information, see Configuring memory for queries with hash joins, aggregates, and other memory-intensive elements.

The following sections describe the performance effects and considerations associated with some of the configuration parameters that are listed at the beginning of this section.

- Setting the size of the buffer pool, logical-log buffer, and physical-log buffer
  The values that you specify for the BUFFERPOOL, DS_TOTAL_MEMORY, LOGBUFF, and PHYSBUFF configuration parameters depend on the type of applications that you are using (OLTP or DSS) and the page size.

- The LOCKS configuration parameter and memory utilization
  The LOCKS configuration parameter specifies the initial size of the lock table. The lock table holds an entry for each lock that a session uses. Each lock uses 120 bytes within a lock table. You must provide for this amount of memory when you configure shared memory.
- The RESIDENT configuration parameter and memory utilization
  The RESIDENT configuration parameter specifies whether shared-memory residency is enforced for the resident portion of database server shared memory. This configuration parameter works only on computers that support forced residency.
- The SHMADD and EXTSHMADD configuration parameters and memory utilization
  The SHMADD configuration parameter specifies the size of each increment of shared memory that the database server dynamically adds to the virtual portion. The EXTSHMADD configuration parameter specifies the size of a virtual-extension segment that is added when user-defined routines or DataBlade routines run in user-defined virtual processors. Trade-offs are involved in determining the size of an increment.
- The SHMTOTAL configuration parameter and memory utilization
  The SHMTOTAL configuration parameter places an absolute upper limit on the amount of shared memory that an instance of the database server can use.
- The SHMVIRTSIZE configuration parameter and memory utilization
  The SHMVIRTSIZE parameter specifies the size of the virtual portion of shared memory to allocate when you start the database server. The virtual portion of shared memory holds session- and request-specific data as well as other information.
- The SHMVIRT_ALLOCSEG configuration parameter and memory utilization
  The SHMVIRT_ALLOCSEG configuration parameter specifies a threshold at which the database server should allocate memory. This configuration parameter also defines an alarm event security-code that is activated if the server cannot allocate the new memory segment, thus ensuring that the database server never runs out of memory.
- The STACKSIZE configuration parameter and memory utilization
  The STACKSIZE configuration parameter indicates the initial stack size for each thread. The database server assigns the amount of space that this parameter indicates to each active thread. This space comes from the virtual portion of database server shared memory. You can reduce the amount of shared memory that the database server adds dynamically.

Related concepts:
Resident portion of shared memory
Virtual portion of shared memory
Related information:
LOW_MEMORY_MGR configuration parameter
LOW_MEMORY_RESERVE configuration parameter

Setting the size of the buffer pool, logical-log buffer, and physical-log buffer

The values that you specify for the BUFFERPOOL, DS_TOTAL_MEMORY, LOGBUFF, and PHYSBUFF configuration parameters depend on the type of applications that you are using (OLTP or DSS) and the page size.

Table 1 lists suggested settings for these parameters or guidelines for setting the parameters.

For information about estimating the size of the resident portion of shared memory, see Estimating the size of the resident portion of shared memory. This calculation includes figuring the size of the buffer pool, logical-log buffer, physical-log buffer, and lock table.

Table 1. Guidelines for OLTP and DSS applications

Configuration Parameter	OLTP Applications	DSS Applications
BUFFERPOOL	The percentage of physical memory that you need for buffer space depends on the amount of memory that is available on your system and the amount of memory that is used for other applications.	Set to a small buffer value and increase the DS_TOTAL_MEMORY value for light scans, queries, and sorts. For operations such as index builds that read data through the buffer pool, configure a larger number of buffers.
DS_TOTAL_MEMORY	Set to a value from 20 to 50 percent of the value of SHMTOTAL, in kilobytes.	Set to a value from 50 to 90 percent of SHMTOTAL.
LOGBUFF	The default value for the logical log buffer size is 64 KB. If you decide to use a smaller value, the database server generates a message a message that indicates that optimal performance might not be obtained. Using a logical log buffer smaller than 64 KB, impacts performance, not transaction integrity. If the database or application is defined to use buffered logging, increasing the LOGBUFF size beyond 64 KB improves performance.	Because database or table logging is usually turned off for DSS applications, you can set LOGBUFF to 32 KB.
PHYSBUFF	The default value for the physical log buffer size is 128 KB. If the RTO_SERVER_RESTART configuration parameter is enabled, use the 512 kilobyte default value for PHYSBUFF. If you decide to use a value that is smaller than the default value, the database server generates a message that indicates that optimal performance might not be obtained. Using a physical log buffer that is smaller than the default size impacts performance, not transaction integrity.	Because most DSS applications do not physically log, you can set PHYSBUFF to 32 KB.

- The BUFFERPOOL configuration parameter and memory utilization
  The BUFFERPOOL configuration parameter specifies the properties of buffer pools. The information that you define in the BUFFERPOOL configuration parameter fields affects memory use.
- The DS_TOTAL_MEMORY configuration parameter and memory utilization
  The DS_TOTAL_MEMORY configuration parameter places a ceiling on the amount of shared memory that a query can obtain. You can use this parameter to limit the performance impact of large, memory-intensive queries. The higher you set this parameter, the more memory a large query can use, and the less memory is available for processing other queries and transactions.
- The LOGBUFF configuration parameter and memory utilization
  The LOGBUFF configuration parameter determines the amount of shared memory that is reserved for each of the three buffers that hold the logical-log records until they are flushed to the logical-log file on disk. The size of a buffer determines how often it fills and therefore how often it must be flushed to the logical-log file on disk.
- The LOW_MEMORY_RESERVE configuration parameter and memory utilization
  The LOW_MEMORY_RESERVE configuration parameter reserves a specific amount of memory, in kilobytes, for the database server to use when critical activities are needed and the server has limited free memory.
- The PHYSBUFF configuration parameter and memory utilization
  The PHYSBUFF configuration parameter determines the amount of shared memory that is reserved for each of the two buffers that serve as temporary storage space for data pages that are about to be modified. The size of a buffer determines how often it fills and therefore how often it must be flushed to the physical log on disk.

Related information:
BUFFERPOOL configuration parameter
DS_TOTAL_MEMORY configuration parameter
LOGBUFF configuration parameter
PHYSBUFF configuration parameter
RTO_SERVER_RESTART configuration parameter

The BUFFERPOOL configuration parameter and memory utilization

The BUFFERPOOL configuration parameter specifies the properties of buffer pools. The information that you define in the BUFFERPOOL configuration parameter fields affects memory use.

You can have multiple buffer pools if you have dbspaces that use different page sizes. The onconfig configuration file contains a BUFFERPOOL line for each page size. For example, on a computer with a 2 KB page size, the onconfig file can contain up to nine lines, including the default specification. When you create a dbspace with a different page size, a buffer pool for that page size is created automatically, if it does not exist. A BUFFERPOOL entry for the page size is added to the onconfig file. The values of the BUFFERPOOL configuration parameter fields are the same as the default specification.

The BUFFERPOOL configuration parameter controls the number of data buffers available to the database server. These buffers are in the buffer pool portion of shared memory and are used to cache database data pages in memory.

Increasing the number of buffers increases the likelihood that a needed data page might already be in memory as the result of a previous request. However, allocating too many buffers can affect the memory-management system and lead to excess operating system paging activity. To take advantage of the large memory available on 64-bit addressing machines, you can increase the size of the buffer pool.

The size of the buffer pool has a significant effect on database I/O and transaction throughput. You can ensure that the buffer pool has enough buffers by making the buffer pool extendable. When the buffer pool is extendable, the database server expands the buffer pool as needed to improve performance.

The size of the buffer pool is equal to the number of buffers multiplied by the page size. The percentage of physical memory that you need for buffer space depends on the amount of memory that you have available on your system and the amount that is used for other applications. For systems with a large amount of available physical memory (4 GB or more), buffer space might be as much as 90 percent of physical memory. For systems with smaller amounts of available physical memory, buffer space might range from 20 to 25 percent of physical memory.

For example, suppose that your system has a page size of 2 KB and 100 MB of physical memory. You can set the value in the **buffers** field to 10,000 - 12,500, which allocates 20 - 25 MB of memory.

Calculate all other shared-memory parameters after you specify the size of the buffer pool.

Note: If you use non-default page sizes, you might need to increase the size of your physical log. If you frequently update non-default pages, you might need a 150 - 200 percent increase of the physical log size. Some experimentation might be needed to tune the physical log. You can adjust the size of the physical log as necessary according to how frequently the filling of the physical log triggers checkpoints.
You can use **onstat -g buf** to monitor buffer pool statistics, including the read-cache rate of the buffer pool. This rate represents the percentage of database pages that are already present in a shared-memory buffer when a query requests a page. (If a page is not already present, the database server must copy it into memory from disk.) If the database server finds the page in the buffer pool, it spends less time on disk I/O. Therefore, you want a high read-cache rate for good performance. For OLTP applications where many users read small sets of data, the goal is to achieve a read cache rate of 95 percent or better. If the buffer pool is extendable, you can specify the read cache hit ratio below which the database server extends the buffer pool.

Use the memory-management monitor utility in your operating system (such as **vmstat** or **sar** on UNIX) to note the level of page scans and paging-out activity. If these levels rise suddenly or rise to unacceptable levels during peak database activity, reduce the size of the buffer pool.

Smart large objects and buffers

Depending upon your situation, you can take one of the following actions to achieve better performance for applications that use smart large objects:

- If your applications frequently access smart large objects that are 2 KB or 4 KB in size, use the buffer pool to keep them in memory longer. Use the following formula to increase the value of the **buffers** field:

```
Additional_buffers = numcur_open_lo *
                     (lo_userdata / pagesize)
```

In this formula:
- *numcur_open_lo* is the number of concurrently opened smart large objects that you can obtain from the **onstat -g smb fdd** command.
- *lo_userdata* is the number of bytes of smart-large-object data that you want to buffer.
- *pagesize* is the default page size in bytes for the computer.

As a rule, try to have enough buffers to hold two smart-large-object pages for each concurrently open smart large object. The additional page is available for read-ahead purposes.

- Use lightweight I/O buffers in the virtual portion of shared memory.
  Use lightweight I/O buffers only when you read or write smart large objects in operations greater than 8000 bytes and seldom access them. That is, if the read or write function calls read large amounts of data in a single-function invocation, use lightweight I/O buffers.

  When you use lightweight I/O buffers, you can prevent the flood of smart large objects into the buffer pool and leave more buffers available for other data pages that multiple users frequently access.

Related concepts:
Lightweight I/O for smart large objects
BUFFERPOOL and its effect on page cleaning
Related information:
BUFFERPOOL configuration parameter
Monitor buffers

The DS_TOTAL_MEMORY configuration parameter and memory utilization

The DS_TOTAL_MEMORY configuration parameter places a ceiling on the amount of shared memory that a query can obtain. You can use this parameter to limit the performance impact of large, memory-intensive queries. The higher you set this parameter, the more memory a large query can use, and the less memory is available for processing other queries and transactions.

For OLTP applications, set DS_TOTAL_MEMORY to 20 - 50 percent of the value of SHMTOTAL, in KB. For applications that involve large decision-support (DSS) queries, increase the value of DS_TOTAL_MEMORY to 50 - 80 percent of SHMTOTAL. If you use your database server instance exclusively for DSS queries, set this parameter to 90 percent of SHMTOTAL.

A *quantum unit* is the minimum increment of memory that is allocated to a query. The Memory Grant Manager (MGM) allocates memory to queries in quantum units. The database server uses the value of DS_MAX_QUERIES with the value of DS_TOTAL_MEMORY to calculate a quantum of memory, according to the following formula:

```
quantum = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

The database server can adjust the size of the quantum dynamically when it grants memory. To allow for more simultaneous queries with smaller quanta each, increase the value of the DS_MAX_QUERIES configuration parameter.

- Algorithm for determining DS_TOTAL_MEMORY
  The database server derives a value for DS_TOTAL_MEMORY if you do not set the DS_TOTAL_MEMORY configuration parameter or if you set this configuration parameter to an inappropriate value.
- Deriving a minimum for decision-support memory
  In the first part of the algorithm that the database server uses to derive the new value for the DS_TOTAL_MEMORY configuration parameter, the database server establishes a minimum amount for decision-support memory.
- Deriving a working value for decision-support memory
  In the second part of the algorithm that the database server uses to derive the new value for the DS_TOTAL_MEMORY configuration parameter, the database server establishes a working value for the amount of decision-support memory.
- Checking the derived value for decision-support memory
  In the final part of the algorithm that the database server uses to derive the new value for the DS_TOTAL_MEMORY configuration parameter, the database server verifies that the amount of shared memory is greater than *min_ds_total_memory* and less than the maximum possible memory space for your computer.

Related concepts:
The Memory Grant Manager
Limiting the performance impact of CPU-intensive queries
Related information:
DS_TOTAL_MEMORY configuration parameter

Algorithm for determining DS_TOTAL_MEMORY

The database server derives a value for DS_TOTAL_MEMORY if you do not set the DS_TOTAL_MEMORY configuration parameter or if you set this configuration parameter to an inappropriate value.

Whenever the database server changes the value that you assigned to DS_TOTAL_MEMORY, it sends the following message to your console:

```
DS_TOTAL_MEMORY recalculated and changed from old_value Kb
            to new_value Kb
```

The variable *old_value* represents the value that you assigned to DS_TOTAL_MEMORY in your configuration file. The variable *new_value* represents the value that the database server derived.

When you receive the preceding message, you can use the algorithm to investigate what values the database server considers inappropriate. You can then take corrective action based on your investigation.

The following sections document the algorithm that the database server uses to derive the new value for DS_TOTAL_MEMORY.

Deriving a minimum for decision-support memory

In the first part of the algorithm that the database server uses to derive the new value for the DS_TOTAL_MEMORY configuration parameter, the database server establishes a minimum amount for decision-support memory.

When you assign a value to the DS_MAX_QUERIES configuration parameter, the database server sets the minimum amount of decision-support memory according to the following formula:

```
min_ds_total_memory = DS_MAX_QUERIES * 128 kilobytes
```

When you do not assign a value to the DS_MAX_QUERIES configuration parameter, the database server uses the following formula instead, which is based on the value of information in the VPCLASS configuration parameter:

```
min_ds_total_memory = NUMBER_CPUVPS * 2 * 128 kilobytes
```


Deriving a working value for decision-support memory

In the second part of the algorithm that the database server uses to derive the new value for the DS_TOTAL_MEMORY configuration parameter, the database server establishes a working value for the amount of decision-support memory.

The database server verifies this amount in the third and final part of the algorithm.

- When the DS_TOTAL_MEMORY configuration parameter is set
  When the DS_TOTAL_MEMORY configuration parameter is set, the database server checks whether the SHMTOTAL configuration parameter is set and then determines which formula to use to calculate the amount of decision-support memory.
- When the DS_TOTAL_MEMORY configuration parameter is not set
  When the DS_TOTAL_MEMORY configuration parameter is not set, the database server uses other sources to calculate a value for the amount of decision-support memory.

When the DS_TOTAL_MEMORY configuration parameter is set

When the DS_TOTAL_MEMORY configuration parameter is set, the database server checks whether the SHMTOTAL configuration parameter is set and then determines which formula to use to calculate the amount of decision-support memory.

When SHMTOTAL is set, the database server uses the following formula to calculate the amount of decision-support memory:

```
IF DS_TOTAL_MEMORY <= SHMTOTAL - nondecision_support_memory THEN
   decision_support_memory = DS_TOTAL_MEMORY
ELSE
    decision_support_memory = SHMTOTAL -
                  nondecision_support_memory
```

This algorithm effectively prevents you from setting DS_TOTAL_MEMORY to values that the database server cannot possibly allocate to decision-support memory.

When SHMTOTAL is not set, the database server sets decision-support memory equal to the value that you specified in DS_TOTAL_MEMORY.

Related information:
DS_TOTAL_MEMORY configuration parameter

When the DS_TOTAL_MEMORY configuration parameter is not set

When the DS_TOTAL_MEMORY configuration parameter is not set, the database server uses other sources to calculate a value for the amount of decision-support memory.

When SHMTOTAL is set, the database server uses the following formula to calculate the amount of decision-support memory:

```
decision_support_memory = SHMTOTAL -
                  nondecision_support_memory
```

When the database server finds that you did not set SHMTOTAL, it sets decision-support memory as in the following example:

```
decision_support_memory = min_ds_total_memory
```

For a description of the variable min_ds_total_memory, see Deriving a minimum for decision-support memory.

Checking the derived value for decision-support memory

In the final part of the algorithm that the database server uses to derive the new value for the DS_TOTAL_MEMORY configuration parameter, the database server verifies that the amount of shared memory is greater than *min_ds_total_memory* and less than the maximum possible memory space for your computer.

When the database server finds that the derived value for decision-support memory is less than the value of the *min_ds_total_memory* variable, it sets decision-support memory equal to the value of *min_ds_total_memory*.

When the database server finds that the derived value for decision-support memory is greater than the maximum possible memory space for your computer, it sets decision-support memory equal to the maximum possible memory space.

The LOGBUFF configuration parameter and memory utilization

The LOGBUFF configuration parameter determines the amount of shared memory that is reserved for each of the three buffers that hold the logical-log records until they are flushed to the logical-log file on disk. The size of a buffer determines how often it fills and therefore how often it must be flushed to the logical-log file on disk.

If you log smart large objects, increase the size of the logical-log buffers to prevent frequent flushing to the logical-log file on disk.

Related reference:
Configuration parameters that affect critical data
Related information:
LOGBUFF configuration parameter

The LOW_MEMORY_RESERVE configuration parameter and memory utilization

The LOW_MEMORY_RESERVE configuration parameter reserves a specific amount of memory, in kilobytes, for the database server to use when critical activities are needed and the server has limited free memory.

If you enable the new LOW_MEMORY_RESERVE configuration parameter by setting it to a specified value in kilobytes, critical activities, such as rollback activities, can complete even when you receive out-of-memory errors.

Related information:
LOW_MEMORY_RESERVE configuration parameter
onstat -g seg command: Print shared memory segment statistics

The PHYSBUFF configuration parameter and memory utilization

The PHYSBUFF configuration parameter determines the amount of shared memory that is reserved for each of the two buffers that serve as temporary storage space for data pages that are about to be modified. The size of a buffer determines how often it fills and therefore how often it must be flushed to the physical log on disk.

Choose a value for PHYSBUFF that is an even increment of the system page size.

Related information:
PHYSBUFF configuration parameter

The LOCKS configuration parameter and memory utilization

The LOCKS configuration parameter specifies the initial size of the lock table. The lock table holds an entry for each lock that a session uses. Each lock uses 120 bytes within a lock table. You must provide for this amount of memory when you configure shared memory.

If the number of locks needed by sessions exceeds the value set in the LOCKS configuration parameter, the database server attempts to increase the lock table by doubling its size. Each time that the lock table overflows (when the number of locks needed is greater than the current size of the lock table), the database server increases the size of the lock table, up to 99 times. Each time that the database server increases the size of the lock table, the server attempts to double its size. However, the server will limit each actual increase to no more than the maximum number of added locks shown in Table 1. After the 99th time that the database server increases the lock table, the server no longer increases the size of the lock table, and an application needing a lock receives an error.

The following table shows the maximum number of locks allowed on 32-bit and 64-bit platforms

Table 1. Maximum number of locks on 32-bit and 64-bit platforms

Platform	Maximum Number of Initial Locks	Maximum Number of Dynamic Lock Table Extensions	Maximum Number of Locks Added Per Lock Table Extension	Maximum Number of Locks Allowed

Platform	Maximum Number of Initial Locks	Maximum Number of Dynamic Lock Table Extensions	Maximum Number of Locks Added Per Lock Table Extension	Maximum Number of Locks Allowed
32-bit	8,000,000	99	100,000	8,000,000 + (99 x 100,000)
64-bit	500,000,000	99	1,000,000	500,000,000 + (99 x 1,000,000)

The default value for the LOCKS configuration parameter is 20,000.

To estimate a different value for the LOCKS configuration parameter, estimate the maximum number of locks that a query needs and multiply this estimate by the number of concurrent users. You can use the guidelines in the following table to estimate the number of locks that a query needs.

Locks per Statement	Isolation Level	Table	Row	Key	TEXT or BYTE Data	CLOB or BLOB Data
SELECT	Dirty Read	0	0	0	0	0
SELECT	Committed Read	1	0	0	0	0
SELECT	Cursor Stability	1	1	0	0	1 lock for the CLOB or BLOB value or (if byte-range locking is used) 1 lock for each range
SELECT	Indexed Repeatable Read	1	Number of rows that satisfy conditions	Number of rows that satisfy conditions	0	1 lock for the CLOB or BLOB value or (if byte-range locking is used) 1 lock for each range
SELECT	Sequential Repeatable Read	1	0	0	0	1 lock for the CLOB or BLOB value or (if byte-range locking is used) 1 lock for each range
INSERT	Not applicable	1	1	Number of indexes	Number of pages in TEXT or BYTE data	1 lock for the CLOB or BLOB value
DELETE	Not applicable	1	1	Number of indexes	Number of pages in TEXT or BYTE data	1 lock for the CLOB or BLOB value
UPDATE	Not applicable	1	1	2 per changed key value	Number of pages in old plus new TEXT or BYTE data	1 lock for the CLOB or BLOB value or (if byte-range locking is used) 1 lock for each range

Important: During the execution of the SQL statement DROP DATABASE, the database server acquires and holds a lock on each table in the database until the entire DROP operation completes. Make sure that the value for LOCKS is large enough to accommodate the largest number of tables in a database.

Related concepts:
Configuring and managing lock usage
Related information:
LOCKS configuration parameter

The RESIDENT configuration parameter and memory utilization

The RESIDENT configuration parameter specifies whether shared-memory residency is enforced for the resident portion of database server shared memory. This configuration parameter works only on computers that support forced residency.

The resident portion in the database server contains the buffer pools that are used for database read and write activity. Performance improves when these buffers remain in physical memory.

You should set the RESIDENT parameter to `1`. If forced residency is not an option on your computer, the database server issues an error message and ignores this configuration parameter.

On machines that support 64-bit addressing, you can have a very large buffer pool and the virtual portion of database server shared memory can also be very large. The virtual portion contains various memory caches that improve performance of multiple queries that access the same tables (see Configure and monitor memory caches). To make the virtual portion resident in physical memory in addition to the resident portion, set the RESIDENT parameter to `-1`.

If your buffer pool is very large, but your physical memory is not very large, you can set RESIDENT to a value greater than 1 to indicate the number of memory segments to stay in physical memory. This specification makes only a subset of the buffer pool resident.

You can turn residency on or off for the resident portion of shared memory in the following ways:

- Use the **onmode** utility to reverse temporarily the state of shared-memory residency while the database server is online.
- Change the RESIDENT parameter to turn shared-memory residency on or off the next time that you start database server shared memory.

Related information:
RESIDENT configuration parameter

The SHMADD and EXTSHMADD configuration parameters and memory utilization

The SHMADD configuration parameter specifies the size of each increment of shared memory that the database server dynamically adds to the virtual portion. The EXTSHMADD configuration parameter specifies the size of a virtual-extension segment that is added when user-defined routines or DataBlade routines run in user-defined

virtual processors. Trade-offs are involved in determining the size of an increment.

Adding shared memory uses CPU cycles. The larger each increment, the fewer increments are required, but less memory is available for other processes. Adding large increments is generally preferred; but when memory is heavily loaded (the scan rate or paging-out rate is high), smaller increments allow better sharing of memory resources among competing programs.

The range of values for SHMADD is `1024` through `4294967296` KB for a 64-bit operating system and `1024` through `524288` KB for a 32-bit operating system. The following table contains recommendations for setting SHMADD according to the size of physical memory.

Memory Size	SHMADD Value
256 MB or less	8192 KB (the default)
257 - 512 MB	16,384 KB
Larger than 512 MB	32,768 KB

The range of values for EXTSHMADD is the same as the range of values of SHMADD.

Note: A shared memory segment can be as large as 4 terabytes, depending on platform limits and the value of the SHMMAX kernel parameter. Use the **onstat -g seg** command to display the number of shared-memory segments that the database server is currently using.

Related tasks:
Configuring UNIX shared memory
Related information:
SHMADD configuration parameter
EXTSHMADD configuration parameter

The SHMTOTAL configuration parameter and memory utilization

The SHMTOTAL configuration parameter places an absolute upper limit on the amount of shared memory that an instance of the database server can use.

If the SHMTOTAL configuration parameter is set to `0` or left unassigned, the database server continues to attach additional shared memory as needed until no virtual memory is available on the system.

You can usually set the SHMTOTAL configuration parameter to `0`, except in the following cases:

- You must limit the amount of virtual memory that the database server uses for other applications or other reasons.
- Your operating system runs out of swap space and performs abnormally. In this case, you can set SHMTOTAL to a value that is a few megabytes less than the total swap space that is available on your computer.
- You are using automatic low memory management.

Related information:
SHMTOTAL configuration parameter

The SHMVIRTSIZE configuration parameter and memory utilization

The SHMVIRTSIZE parameter specifies the size of the virtual portion of shared memory to allocate when you start the database server. The virtual portion of shared memory holds session- and request-specific data as well as other information.

Although the database server adds increments of shared memory to the virtual portion as needed to process large queries or peak loads, allocation of shared memory increases time for transaction processing. Therefore, you should set SHMVIRTSIZE to provide a virtual portion large enough to cover your normal daily operating requirements. The size of SHMVIRTSIZE can be as large the SHMMAX configuration parameter allows.

The maximum value of SHMVIRTSIZE, which must be a positive integer, is:

- 4 terabytes on a 64-bit database server
- 2 gigabytes on a 32-bit database server

For an initial setting, it is suggested that you use the larger of the following values:

- `8000`
- *connections* `* 350`

The *connections* variable is the number of connections for all network types that are specified in the sqlhosts information by one or more NETTYPE configuration parameters. (The database server uses *connections* `* 200` by default.)

Once system utilization reaches a stable workload, you can reconfigure a new value for SHMVIRTSIZE. As noted in Freeing shared memory with onmode -F, you can instruct the database server to release shared-memory segments that are no longer in use after a peak workload or large query.

Related information:
SHMVIRTSIZE configuration parameter

The SHMVIRT_ALLOCSEG configuration parameter and memory utilization

The SHMVIRT_ALLOCSEG configuration parameter specifies a threshold at which the database server should allocate memory. This configuration parameter also defines an alarm event security-code that is activated if the server cannot allocate the new memory segment, thus ensuring that the database server never runs out of memory.

When you set the SHMVIRT_ALLOCSEG configuration parameter, you must:

- Specify the percentage of memory used or the whole number of kilobytes remaining on the server. You cannot use negative values and values between `0` and `.39`.
- Specify the alarm event-security code, which is a value ranging from `1` (not noteworthy) to `5` (fatal). If you do not specify an event-security code, the server sets the value to `3`, which is the default value.

Example 1:

```
SHMVIRT_ALLOCSEG 3000, 4
```

This specifies that if the database serve has 3000 kilobytes remaining in virtual memory and additional kilobytes of memory cannot be allocated, the server raises an alarm level of 4.
Example 2:

```
SHMVIRT_ALLOCSEG .8, 4
```

This specifies that if the database server has twenty percent remaining in virtual memory and additional kilobytes of memory cannot be allocated, the server raises an alarm level of 4.
Related information:
Event Alarm Parameters
SHMVIRT_ALLOCSEG configuration parameter

The STACKSIZE configuration parameter and memory utilization

The STACKSIZE configuration parameter indicates the initial stack size for each thread. The database server assigns the amount of space that this parameter indicates to each active thread. This space comes from the virtual portion of database server shared memory. You can reduce the amount of shared memory that the database server adds dynamically.

To reduce the amount of shared memory that the database server adds dynamically, estimate the amount of the stack space required for the average number of threads that your system runs and include that amount in the value that you set for the SHMVIRTSIZE configuration parameter.

To estimate the amount of stack space that you require, use the following formula:

```
stacktotal = STACKSIZE * avg_no_of_threads
```

avg_no_of_threads
is the average number of threads. You can monitor the number of active threads at regular intervals to determine this amount. Use **onstat -g sts** to check the stack use of threads. A general estimate is between 60 and 70 percent of the total number of connections (specified in the NETTYPE parameters in your ONCONFIG file), depending on your workload.

The database server also executes user-defined routines (UDRs) with user threads that use this stack. Programmers who write user-defined routines should take the following measures to avoid stack overflow:

- Do not use large automatic arrays.
- Avoid excessively deep calling sequences.
- *For DB-Access only:* Use **mi_call** to manage recursive calls.

If you cannot avoid stack overflow with these measures, use the STACK modifier of the CREATE FUNCTION statement to increase the stack for a particular routine.

Related information:
STACKSIZE configuration parameter

Configure and monitor memory caches

The database server uses caches to store information in memory instead of performing a disk read or another operation to obtain the information. These memory caches improve performance for multiple queries that access the same tables. You can set some configuration parameters to increase the effectiveness of each cache. You can view information about memory caches by running **onstat** commands.

The following table lists the main memory caches that have the greatest effect on performance and how to configure and monitor those caches.

Table 1. Main memory caches

Cache Name	Cache Description	Configuration Parameters	onstat command
Data Dictionary	Stores information about the table definition (such as column names and data types).	DD_HASHSIZE: The maximum number of buckets in the cache. DD_HASHMAX: The number of tables in each bucket	**onstat -g dic**
Data Distribution	Stores distribution statistics for a column.	DS_POOLSIZE: The maximum number of entries in the cache. DS_HASHSIZE: The number of buckets in the cache.	**onstat -g dsc**

Cache Name	Cache Description	Configuration Parameters	onstat command
SQL Statement	Stores parsed and optimized SQL statements.	STMT_CACHE: Enable the SQL statement cache. STMT_CACHE_HITS: The number of times anSQL statement is run before it is cached. STMT_CACHE_NOLIMIT: Prohibit entries into the SQL statement cache when allocated memory exceeds the value of the STMT_CACHE_SIZE configuration parameter. STMT_CACHE_NUMPOOL: The number of memory pools for the SQL statement cache. STMT_CACHE_SIZE: The size of the SQL statement cache, in KB.	**onstat -g ssc**
UDR	Stores frequently used user-defined routines and SPL routines.	PC_POOLSIZE: The maximum number of user-defined routines and SPL routines in the cache. PC_HASHSIZE: The number of buckets in the UDR cache.	**onstat -g prc**

The following table lists more memory caches and how to configure and monitor those caches.

Table 2. Additional memory caches

Cache Name	Cache Description	Configuration Parameters	onstat command
Access method	Stores user-defined access methods.	None.	**onstat -g cac am**
Aggregate	Stores user-defined aggregates.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac agg**
AQT dictionary	Stores accelerated query tables that the database server uses to determine which queries can be processed by Informix® Warehouse Accelerator.	None.	**onstat -g cac aqt**
Cast	Stores user-defined casts.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac cast**
External directives	Stores external directives.	None.	**onstat -g cac ed**
LBAC security policy information	Stores LBAC security policies.	PLCY_POOLSIZE PLCY_HASHSIZE	**onstat -g cac lbacplcy**
LBAC credential memory	Stores LBAC credentials.	USRC_POOLSIZE USRC_HASHSIZE	**onstat -g cac lbacusrc**
Operator class instance	Stores user-defined operator classes.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac opci**
Procedure name	Stores user-defined routine and SPL routine names.	PC_POOLSIZE PC_HASHSIZE	**onstat -g cac prn**
Routine resolution	Stores user-defined routine resolution information.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac rr**
Secondary transient	Stores transient unnamed complex data types on secondary servers in a high-availability cluster.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac ttype**
Extended type ID	Stores the IDs of user-defined types.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac typei**
Extended type name	Stores the name of user-defined types.	DS_POOLSIZE DS_HASHSIZE	**onstat -g cac typen**

- Data-dictionary cache
  The first time that the database server accesses a table, it retrieves the information that it needs about the table (such as the column names and data types) from the system catalog tables on disk. After the database server has accessed the table, it places that information in the data-dictionary cache in shared memory.
- Data-distribution cache
  The query optimizer uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost. The first time that the optimizer accesses the distribution statistics for a column, the database server retrieves the statistics from the **sysdistrib** system catalog table on disk and places that information in the data-distribution cache in memory.
- Monitor and tune the SQL statement cache
  The SQL statement cache stores optimized SQL statements so that multiple users who run the same SQL statement can achieve some performance improvements.

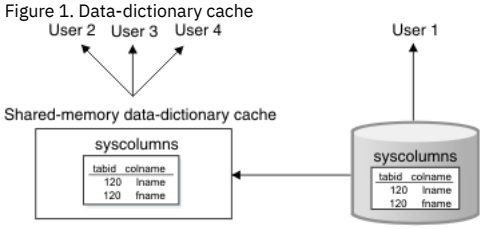
Related concepts:
SPL routine executable format stored in UDR cache
Related information:
onstat -g cac command: Print information about caches
onstat -g dsc command: Print distribution cache information
onstat -g prc command: Print sessions using UDR or SPL routines
onstat -g ssc command: Print SQL statement occurrences
Database configuration parameters

Data-dictionary cache

The first time that the database server accesses a table, it retrieves the information that it needs about the table (such as the column names and data types) from the system catalog tables on disk. After the database server has accessed the table, it places that information in the data-dictionary cache in shared memory.

Figure 1 shows how the database server uses this cache for multiple users. User 1 accesses the column information for **tabid 120** for the first time. The database server puts the column information in the data-dictionary cache. When user 2, user 3 and user 4 access the same table, the database server does not have to read from disk to access the data-dictionary information for the table. Instead, it reads the dictionary information from the data-dictionary cache in memory.

Figure 1. Data-dictionary cache



The database server still places pages for system catalog tables in the buffer pool, as it does all other data and index pages. However, the data-dictionary cache offers an additional performance advantage, because the data-dictionary information is organized in a more efficient format and organized to allow fast retrieval.

- Data-dictionary configuration
  The database server uses a hashing algorithm to store and locate information within the data-dictionary cache. The DD_HASHSIZE and DD_HASHMAX configuration parameters control the size of the data-dictionary cache.

Data-dictionary configuration

The database server uses a hashing algorithm to store and locate information within the data-dictionary cache. The DD_HASHSIZE and DD_HASHMAX configuration parameters control the size of the data-dictionary cache.

To modify the number of buckets in the data-dictionary cache, use DD_HASHSIZE (must be a prime number). To modify the number of tables that can be stored in one bucket, use DD_HASHMAX.

For medium to large systems, you can start with the following values for these configuration parameters:

- DD_HASHSIZE: `503`
- DD_HASHMAX: `4`

With these values, you can potentially store information about 2012 tables in the data-dictionary cache, and each hash bucket can have a maximum of 4 tables.

If the bucket reaches the maximum size, the database server uses a least recently used mechanism to clear entries from the data dictionary.

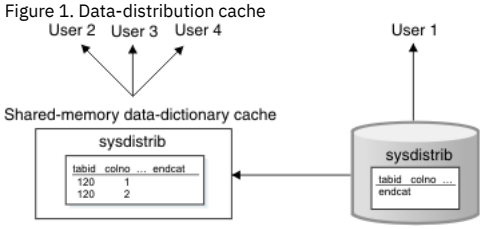
Related information:
DD_HASHSIZE configuration parameter
DD_HASHMAX configuration parameter

Data-distribution cache

The query optimizer uses distribution statistics generated by the UPDATE STATISTICS statement in the MEDIUM or HIGH mode to determine the query plan with the lowest cost. The first time that the optimizer accesses the distribution statistics for a column, the database server retrieves the statistics from the **sysdistrib** system catalog table on disk and places that information in the data-distribution cache in memory.

Figure 1 shows how the database server accesses the data-distribution cache for multiple users. When the optimizer accesses the column distribution statistics for User 1 for the first time, the database server puts the distribution statistics in the data-distribution cache. When the optimizer determines the query plan for user 2, user 3 and user 4 who access the same column, the database server does not have to read from disk to access the data-distribution information for the table. Instead, it reads the distribution statistics from the data-distribution cache in shared memory.

Figure 1. Data-distribution cache



The database server initially places pages for the **sysdistrib** system catalog table in the buffer pool as it does all other data and index pages. However, the data-distribution cache offers additional performance advantages. It:

- Is organized in a more efficient format
- Is organized to allow fast retrieval
- Bypasses the overhead of the buffer pool management

- Frees more pages in the buffer pool for actual data pages rather than system catalog pages
- Reduces I/O operations to the system catalog table

- Data-distribution configuration
  The database server uses a hashing algorithm to store and locate information within the data-distribution cache. The DS_POOLSIZE configuration parameter controls the size of the data-distribution cache and controls the total number of column distributions that can be stored in the data-distribution cache. The value of the DS_POOLSIZE configuration parameter represents half of the maximum number of distributions in the data distribution cache.

Data-distribution configuration

The database server uses a hashing algorithm to store and locate information within the data-distribution cache. The DS_POOLSIZE configuration parameter controls the size of the data-distribution cache and controls the total number of column distributions that can be stored in the data-distribution cache. The value of the DS_POOLSIZE configuration parameter represents half of the maximum number of distributions in the data distribution cache.

To modify the number of buckets in the data-distribution cache, use the DS_HASHSIZE configuration parameter.

For example, with the default values of `127` for DS_POOLSIZE and `31` for DS_HASHSIZE, you can potentially store distributions for about 254 columns in the data-distribution cache. When the cache is full, the database server automatically increases the size of the cache by 10%.

The values that you set for DS_HASHSIZE and DS_POOLSIZE, depend on the following factors:

- The number of columns for which you run the UPDATE STATISTICS statement in HIGH or MEDIUM mode and you expect to be used most often in frequently run queries.
  If you do not specify columns when you run UPDATE STATISTICS for a table, the database server generates distributions for all columns in the table.

  You can use the values of DD_HASHSIZE and DD_HASHMAX as guidelines for DS_HASHSIZE and DS_POOLSIZE. The DD_HASHSIZE and DD_HASHMAX specify the size for the data-dictionary cache, which stores information and statistics about tables that queries access.

  For medium to large systems, you can start with the following values:

  - DD_HASHSIZE 503
  - DD_HASHMAX 4
  - DS_HASHSIZE 503
  - DS_POOLSIZE 1000

  Monitor these caches by running the **onstat -g dsc** command to see the actual usage, and you can adjust these parameters accordingly.

- The amount of memory available
  The amount of memory that is required to store distributions for a column depends on the level at which you run UPDATE STATISTICS. Distributions for a single column might require between 1 KB and 2 MB, depending on whether you specify medium or high mode or enter a finer resolution percentage when you run UPDATE STATISTICS.

If the size of the data-distribution cache is too small, the following performance problems can occur:

- The database server uses the DS_POOLSIZE value to determine when to remove entries from the data-distribution cache. However, if the optimizer needs the dropped distributions for another query, the database server must reaccess them from the **sysdistrib** system catalog table on disk. The additional I/O and buffer pool operations to access **sysdistrib** on disk adds to the total response time of the query.
  The database server tries to maintain the number of entries in data-distribution cache at the DS_POOLSIZE value. If the total number of entries reaches within an internal threshold of DS_POOLSIZE, the database server uses a least recently used mechanism to remove entries from the data-distribution cache. The number of entries in a hash bucket can go past this DS_POOLSIZE value, but the database server eventually reduces the number of entries when memory requirements drop.

- If DS_HASHSIZE is small and DS_POOLSIZE is large, overflow lists can be long and require more search time in the cache.
  Overflow occurs when a hash bucket already contains an entry. When multiple distributions hash to the same bucket, the database server maintains an overflow list to store and retrieve the distributions after the first one.

  If DS_HASHSIZE and DS_POOLSIZE are approximately the same size, the overflow lists might be smaller or even nonexistent, which might waste memory. However, the amount of unused memory is insignificant overall.

You might want to change the values of the DS_HASHSIZE and DS_POOLSIZE configuration parameters if you see the following situations:

- If the data-distribution cache is full most of the time and commonly used columns are not listed in the **distribution name** field, try increasing the values of the DS_HASHSIZE and DS_POOLSIZE configuration parameters.
- If the total number of entries is much lower than the value of the DS_POOLSIZE configuration parameter, you can reduce the values of the DS_HASHSIZE and DS_POOLSIZE configuration parameters.
- If the number of hits are not evenly distributed among hash lists, increase the number of hash lists by increasing the value of the DS_HASHSIZE configuration parameter. Adjust the number of hash lists to have the least number of high hit entries per hash list.

Related information:
DD_HASHSIZE configuration parameter
DD_HASHMAX configuration parameter
DS_POOLSIZE configuration parameter
onstat -g dsc command: Print distribution cache information

Monitor and tune the SQL statement cache

The SQL statement cache stores optimized SQL statements so that multiple users who run the same SQL statement can achieve some performance improvements.

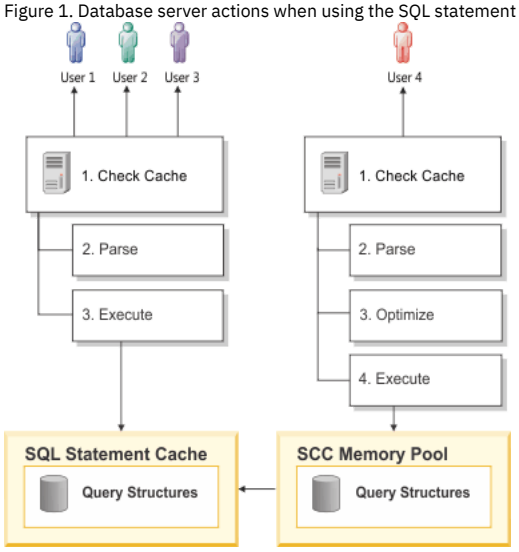
These performance improvements are:

- Reduced response times because they bypass the optimization step, as Figure 1 shows
- Reduced memory usage because the database server shares query data structures among users

For more information about the effect of the SQL statement cache on the performance of individual queries, see Optimize queries with the SQL statement cache.

Figure 1 shows how the database server accesses the SQL statement cache for multiple users.

- When the database server runs an SQL statement for User 1 for the first time, the database server checks whether the same exact SQL statement is in the SQL statement cache. If it is not in the cache, the database server parses the statement, determines the optimal query plan, and runs the statement.
- When User 2 runs the same SQL statement, the database server finds the statement in the SQL statement cache and does not optimize the statement.
- Similarly, if User 3 and User 4 run the same SQL statement, the database server does not optimize the statement. Instead, it uses the query plan in the SQL statement cache in memory.
  Figure 1. Database server actions when using the SQL statement cache



- Prepared statements and the statement cache
  Prepared statements are inherently cached for a single session. This means that if a prepared statement is executed many times or if a single cursor is opened many times, the session uses the same prepared query plan.
- SQL statement cache configuration
  The value of the STMT_CACHE configuration parameter enables or disables the SQL statement cache.
- Number of SQL statement executions
  When the SQL statement cache is enabled, the database server inserts a qualified SQL statement and its memory structures immediately in the SQL statement cache by default.
- Monitoring and tuning the size of the SQL statement cache
  If the size of the SQL statement cache is too small, performance problems can occur. You can monitor the effectiveness of the size of the SQL statement cache.
- Memory limit and size
  Although the database server tries to clean the SQL statement cache, sometimes entries cannot be removed because they are currently in use. In this case, the size of the SQL statement cache can exceed the value of the STMT_CACHE_SIZE configuration parameter.
- Multiple SQL statement cache pools
  Under some circumstances when the SQL statement cache is enabled, the database server allocates memory from one pool for query structures.
- SQL statement cache information in onstat -g ssc output
  The **onstat -g ssc** command displays summary information for the SQL statement cache.

Prepared statements and the statement cache

Prepared statements are inherently cached for a single session. This means that if a prepared statement is executed many times or if a single cursor is opened many times, the session uses the same prepared query plan.

If a session prepares a statement and then executes it many times, the SQL statement cache does not affect performance, because the statement is optimized just once during the PREPARE statement.

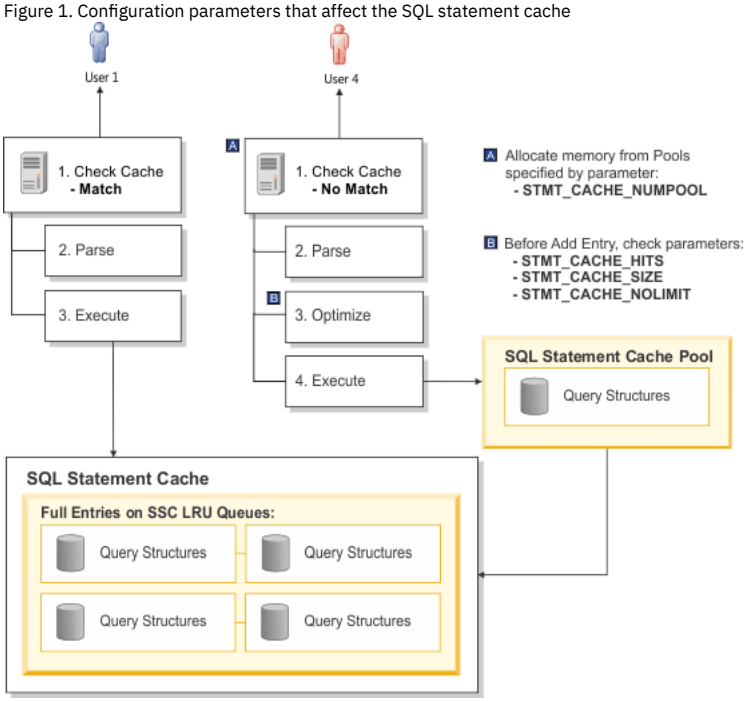
However, if other sessions also prepare that same statement, or if the first session prepares the statement several times, the statement cache usually provides a direct performance benefit, because the database server only calculates the query plan once. Of course, the original session might gain a small benefit from the statement cache, even if it only prepares the statement once, because other sessions use less memory, and the database server does less work for the other sessions

SQL statement cache configuration

The value of the STMT_CACHE configuration parameter enables or disables the SQL statement cache.

For more information about how the value of the STMT_CACHE configuration parameter enables the SQL statement cache, see Enabling the SQL statement cache describes.

Figure 1 shows how the database server uses the values of the pertinent configuration parameters for the SQL statement cache. Further explanation follows the figure.
Figure 1. Configuration parameters that affect the SQL statement cache



When the database server uses the SQL statement cache for a user, it means the database server takes the following actions:

- Checks the SQL statement cache first for a match of the SQL statement that the user is executing
- If the SQL statement matches an entry, executes the statement using the query memory structures in the SQL statement cache (User 2 in Figure 1)
- If the SQL statement does not match an entry, the database server checks if it qualifies for the cache.
  For information about what qualifies an SQL statement for the cache, see SQL statement cache qualifying criteria.
- If the SQL statement qualifies, inserts an entry into the cache for subsequent executions of the statement.

The following parameters affect whether or not the database server inserts the SQL statement into the cache (User 1 in Figure 1):

- STMT_CACHE_HITS specifies the number of times the statement executes with an entry in the cache (referred to as *hit count*). The database server inserts one of the following entries, depending on the hit count:
  - If the value of STMT_CACHE_HITS is 0, inserts a fully cached entry, which contains the text of the SQL statement plus the query memory structures
  - If the value of STMT_CACHE_HITS is not 0 and the statement does not exist in the cache, inserts a key-only entry that contains the text of the SQL statement. Subsequent executions of the SQL statement increment the hit count.
  - If the value of STMT_CACHE_HITS is equal to the number of hits for a key-only entry, adds the query memory structures to make a fully cached entry.
- STMT_CACHE_SIZE specifies the size of the SQL statement cache, and STMT_CACHE_NOLIMIT specifies whether or not to limit the memory of the cache to the value of STMT_CACHE_SIZE. If you do not specify the STMT_CACHE_SIZE parameter, it defaults to 524288 (512 * 1024) bytes.
  The default value for STMT_CACHE_NOLIMIT is `1`, which means the database server will insert entries into the SQL statement cache even though the total amount of memory might exceed the value of STMT_CACHE_SIZE.

  When STMT_CACHE_NOLIMIT is set to `0`, the database server inserts the SQL statement into the cache if the current size of the cache will not exceed the memory limit.

The following sections on STMT_CACHE_HITS, STMT_CACHE_SIZE, STMT_CACHE_NOLIMIT, STMT_CACHE_NUMPOOL and provide more details on how the following configuration parameters affect the SQL statement cache and reasons why you might want to change their default values.

Number of SQL statement executions

When the SQL statement cache is enabled, the database server inserts a qualified SQL statement and its memory structures immediately in the SQL statement cache by default.

If your workload has a disproportionate number of ad hoc queries, use the STMT_CACHE_HITS configuration parameter to specify the number of times an SQL statement is executed before the database server places a fully cached entry in the statement cache.

When the STMT_CACHE_HITS configuration parameter is greater than `0` and the number of times the SQL statement has been executed is less than STMT_CACHE_HITS, the database server inserts key-only entries in the cache. This specification minimizes unshared memory structures from occupying the statement cache, which leaves more memory for SQL statements that applications use often.

Monitor the number of hits on the SQL statement cache to determine if your workload is using this cache effectively. The following sections describe ways to monitor the SQL statement cache hits.

- Monitoring the number of hits on the SQL statement cache
  To monitor the number of hits in the SQL statement cache, run the **onstat -g ssc** command.
- Determining the number of nonshared entries in the SQL statement cache
  To determine how many nonshared entries exist in the SQL statement cache, run **onstat -g ssc all**.

Related concepts:
Too many single-use queries in the SQL statement cache
Related information:
STMT_CACHE_HITS configuration parameter

Monitoring the number of hits on the SQL statement cache

To monitor the number of hits in the SQL statement cache, run the **onstat -g ssc** command.

The **onstat -g ssc** command displays fully cached entries in the SQL statement cache. Figure 1 shows sample output for **onstat -g ssc**.
Figure 1. onstat -g ssc output

```
onstat -g ssc

Statement Cache Summary:
#lrus   currsize  maxsize   Poolsize   #hits    nolimit
4       49456     524288    57344      0        1

Statement Cache Entries:

lru hash ref_cnt  hits  flag heap_ptr      database           user
---------------- ---- ------------------------------------------------------------
  0  153       0     0    -F  a7e4690      vjp_stores         virginia
  SELECT * FROM customer, orders
    WHERE customer.customer_num = orders.customer_num
     AND order_date > "01/01/07"

  1  259       0     0     -F  aa58c20     vjp_stores         virginia
  SELECT * FROM customer, orders
    WHERE customer.customer_num = orders.customer_num
     AND order_date > "01/01/2007"

  2  232       0     1     DF  aa3d020     vjp_stores         virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
     AND O.order_num = I.order_num

  3  232       1     1     -F  aa8b020     vjp_stores         virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num


    Total number of entries: 4.
```

To monitor the number of times that the database server reads the SQL statement within the cache, look at the following output columns:

- In the `Statement Cache Summary` portion of the **onstat -g ssc** output, the **#hits** column is the value of the SQL_STMT_HITS configuration parameter.
  In Figure 1, the **#hits** column in the Statement Cache Summary portion of the output has a value of `0`, which is the default value of the STMT_CACHE_HITS configuration parameter.

  Important: The database server uses entries in the SQL statement cache only if the statements are exactly the same. The first two entries in Figure 1 are not the same because each contains a different literal value in the **order_date** filter.
- In the `Statement Cache Entries` portion of the **onstat -g ssc** output, the hits column shows the number of times that the database server ran each individual SQL statement from the cache. In other words, the column shows the number of times that the database server uses the memory structures in the cache instead of optimizing the statements to generate them again.
  The first time that it inserts the statement in the cache, the **hits** value is `0`.
  - The first two SQL statements in Figure 1 have a **hits** column value of `0`, which indicates that each statement is inserted into the cache but not yet run from the cache.
  - The last two SQL statements in Figure 1 have a **hits** column value of `1`, which indicates that these statements ran once from the cache.

  The **hits** value for individual entries indicates how much sharing of memory structures is done. Higher values in the **hits** column indicate that the SQL statement cache is useful in improving performance and memory usage.

For a complete description of the output fields that **onstat -g ssc** displays, see SQL statement cache information in onstat -g ssc output.

Determining the number of nonshared entries in the SQL statement cache

To determine how many nonshared entries exist in the SQL statement cache, run **onstat -g ssc all**.

The **onstat -g ssc all** option displays the key-only entries in addition to the fully cached entries in the SQL statement cache.

To determine how many nonshared entries exist in the cache:

1. Compare the **onstat -g ssc all** output with the **onstat -g ssc** output.
2. If the difference between these two outputs shows that many nonshared entries exist in the SQL statement cache, increase the value of the STMT_CACHE_HITS configuration parameter to allow more shared statements to reside in the cache and reduce the management overhead of the SQL statement cache.

You can use one of the following methods to change the STMT_CACHE_HITS parameter value:

- Update the ONCONFIG file to specify the STMT_CACHE_HITS configuration parameter. You must restart the database server for the new value to take effect.
  You can use a text editor to edit the ONCONFIG file. Then bring down the database server with the **onmode -ky** command and restart with the **oninit** command.
- Increase the STMT_CACHE_HITS configuration parameter dynamically while the database server is running:
  You can use any of the following methods to reset the STMT_CACHE_HITS value at run time:
  - Issue the **onmode -W** command. The following example specifies that three (3) instances are required before a new query is added to the statement cache:

    ```
    onmode -W STMT_CACHE_HITS 2
    ```

  - Call the **ADMIN** or **TASK** function of the SQL administration API. The following example is equivalent to the **onmode** command in the previous example:

    ```
    EXECUTE FUNCTION TASK("ONMODE", "W", "STMT_CACHE_HITS", "2");
    ```

  If you increase STMT_CACHE_HITS dynamically without updating the configuration file, and the database server is subsequently restarted, the STMT_CACHE_HITS setting reverts the value in the ONCONFIG file. Therefore, if you want the setting to persist after subsequent restarts, modify the ONCONFIG file.

Monitoring and tuning the size of the SQL statement cache

If the size of the SQL statement cache is too small, performance problems can occur. You can monitor the effectiveness of the size of the SQL statement cache.

The following performance problems can occur:

- Frequently executed SQL statements are not in the cache
  The statements used most often should remain in the SQL statement cache. If the SQL statement cache is not large enough, the database server might not have enough room to keep these statements when other statements come into the cache. For subsequent executions, the database server must reparse, reoptimize, and reinsert the SQL statement into the cache. Try increasing STMT_CACHE_SIZE.
- The database server spends a lot of time cleaning the SQL statement cache
  The database server tries to prevent the SQL statement cache from allocating large amounts of memory by using a threshold (70 percent of the STMT_CACHE_SIZE parameter) to determine when to remove entries from the SQL statement cache. If the new entry causes the size of the SQL statement cache to exceed the threshold, the database server removes least recently used entries (that are not currently in use) before inserting the new entry.

  However, if a subsequent query needs the removed memory structures, the database server must reparse and reoptimize the SQL statement. The additional processing time to regenerate these memory structures adds to the total response time of the query.

You can set the size of the SQL statement cache in memory with the STMT_CACHE_SIZE configuration parameter. The value of the parameter is the size in kilobytes. If STMT_CACHE_SIZE is not set, the default value is 512 kilobytes.

The **onstat -g ssc** output shows the value of STMT_CACHE_SIZE in the **maxsize** column. In Figure 1, this **maxsize** column has a value of `524288`, which is the default value (512 * 1024 = 524288).

Use the **onstat -g ssc** and **onstat -g ssc all** options to monitor the effectiveness of size of the SQL statement cache. If you do not see cache entries for the SQL statements that applications use most, the SQL statement cache might be too small or too many unshared SQL statement occupy the cache. The following sections describe how to determine these situations.

- Changing the size of the SQL statement cache
  You can analyze **onstat -g ssc all** output to determine if the SQL statement cache is too small. If the size of the cache is too small, you can change it.
- Too many single-use queries in the SQL statement cache
  When the database server places many queries that are only used once in the cache, they might replace statements that other applications use often. You can view **onstat -g ssc all** output to determine if too many unshared SQL statements occupy the cache. If so, you can prevent unshared SQL statements from being fully cached.

Related information:
STMT_CACHE_SIZE configuration parameter

Changing the size of the SQL statement cache

You can analyze **onstat -g ssc all** output to determine if the SQL statement cache is too small. If the size of the cache is too small, you can change it.

To determine if the size of the SQL statement cache is too small:

1. Run **onstat -g ssc all** to determine if the cache is too small.
2. Look at the values in the following output columns in the Statement Cache Entries portion of the **onstat -g ssc all** output:
   - The **flags** column shows the current status of an SQL statement in the cache.
     A value of `F` in the second position indicates that the statement is currently fully cached.

A value of – in the second position indicates that only the statement text (key-only entry) is in the cache. Entries with this – value in the second position appear in the **onstat -g ssc all** output, but not in the **onstat -g ssc** output.

- The **hits** column shows the number of times the SQL statement has been executed, excluding the first time it is inserted into the cache.

If you do not see fully cached entries for statements that applications use most and the value in the **hits** column is large for the entries that do occupy the cache, then the SQL statement cache is too small.

To change the size of the SQL statement cache:

1. Update the value of the STMT_CACHE_SIZE configuration parameter.
2. Restart the database server for the new value to take effect.

Related information:
STMT_CACHE_SIZE configuration parameter

Too many single-use queries in the SQL statement cache

When the database server places many queries that are only used once in the cache, they might replace statements that other applications use often. You can view **onstat -g ssc all** output to determine if too many unshared SQL statements occupy the cache. If so, you can prevent unshared SQL statements from being fully cached.

Look at the values in the following output columns in the `Statement Cache Entries` portion of the **onstat -g ssc all** output. If you see a lot of entries that have both of the following values, too many unshared SQL statements occupy the cache:

- **flags** column value of `F` in the second position
  A value of `F` in the second position indicates that the statement is currently fully cached.
- **hits** column value of `0` or `1`
  The **hits** column shows the number of times the SQL statement has been executed, excluding the first time it is inserted into the cache.

Increase the value of the STMT_CACHE_HITS configuration parameter to prevent unshared SQL statements from being fully cached.

Related concepts:
Number of SQL statement executions
Related information:
STMT_CACHE_HITS configuration parameter

Memory limit and size

Although the database server tries to clean the SQL statement cache, sometimes entries cannot be removed because they are currently in use. In this case, the size of the SQL statement cache can exceed the value of the STMT_CACHE_SIZE configuration parameter.

The default value of the STMT_CACHE_NOLIMIT configuration parameter is `1`, which means the database server inserts the statement even though the current size of the cache might be greater than the value of the STMT_CACHE_SIZE parameter.

If the value of the STMT_CACHE_NOLIMIT configuration parameter is `0`, the database server does not insert either a fully-qualified or key-only entry into the SQL statement cache if the size will exceed the value of STMT_CACHE_SIZE.

Use the **onstat -g ssc** option to monitor the current size of the SQL statement cache. Look at the values in the following output columns of the **onstat -g ssc** output:

- The **currsize** column shows the number of bytes currently allocated in the SQL statement cache.
  In Figure 1, the **currsize** column has a value of `11264`.
- The **maxsize** column shows the value of STMT_CACHE_SIZE.
  In Figure 1, the **maxsize** column has a value of `524288`, which is the default value (512 * 1024 = 524288).

When the SQL statement cache is full and users are currently executing all statements within it, any new SQL statements that a user executes can cause the SQL statement cache to grow beyond the size that STMT_CACHE_SIZE specifies. When the database server is no longer using an SQL statement within the SQL statement cache, it frees memory in the SQL statement cache until the size reaches a threshold of STMT_CACHE_SIZE. However, if thousands of concurrent users are executing several ad hoc queries, the SQL statement cache can grow very large before any statements are removed. In such cases, take one of the following actions:

- Set the STMT_CACHE_NOLIMIT configuration parameter to 0 to prevent insertions when the cache size exceeds the value of the STMT_CACHE_SIZE parameter.
- Set the STMT_CACHE_HITS parameter to a value greater than 0 to prevent caching unshared SQL statements.

You can use one of the following methods to change the STMT_CACHE_NOLIMIT configuration parameter value:

- Update the ONCONFIG file to specify the STMT_CACHE_NOLIMIT configuration parameter. You must restart the database server for the new value to take effect.
- Use the **onmode -W** command to override the STMT_CACHE_NOLIMIT configuration parameter dynamically while the database server is running.

  ```
  onmode -W STMT_CACHE_NOLIMIT 0
  ```

  If you restart the database server, the value reverts the value in the ONCONFIG file. Therefore, if you want the setting to remain for subsequent restarts, modify the ONCONFIG file.

Related information:
STMT_CACHE_HITS configuration parameter

Multiple SQL statement cache pools

Under some circumstances when the SQL statement cache is enabled, the database server allocates memory from one pool for query structures.

These circumstances are:

- When the database server does not find a matching entry in the cache
- When the database server finds a matching key-only entry in the cache and the hit count reaches the value of the STMT_CACHE_HITS configuration parameter

This one pool can become a bottleneck as the number of users increases. The STMT_CACHE_NUMPOOL configuration parameter allows you to configure multiple **sscpools**.

You can monitor the pools in the SQL statement cache to determine the following situations:

- The number of SQL statement cache pools is sufficient for your workload.
- The size or limit of the SQL statement cache is not causing excessive memory management.

- Number of SQL statement cache pools
  When the SQL statement cache (SSC) is enabled, the database server allocates memory from the SSC pool for unlinked SQL statements. The default value for the STMT_CACHE_NUMPOOL configuration parameter is 1. As the number of users increases, this one SSC pool might become a bottleneck.
- Size of SQL statement cache pools and the current cache
  Use the **onstat -g ssc pool** option to monitor the usage of each SQL statement cache (SSC) pool.

Related information:
STMT_CACHE_NUMPOOL configuration parameter

Number of SQL statement cache pools

When the SQL statement cache (SSC) is enabled, the database server allocates memory from the SSC pool for unlinked SQL statements. The default value for the STMT_CACHE_NUMPOOL configuration parameter is 1. As the number of users increases, this one SSC pool might become a bottleneck.

The number of longspins on the SSC pool indicates whether or not the SSC pool is a bottleneck.

Use the **onstat -g spi** option to monitor the number of longspins on an SSC pool. The **onstat -g spi** command displays a list of the resources in the system for which a wait was required before a latch on the resource could be obtained. During the wait, the thread spins (or loops), trying to acquire the resource. The **onstat -g spi** output displays the number of times a wait (**Num Waits** column) was required for the resource and the number of total loops (**Num Loops** column). The **onstat -g spi** output displays only resources that have at least one wait.

Figure 1 shows an excerpt of sample output for **onstat -g spi**. Figure 1 indicates that no waits occurred for any SSC pool (the **Name** column does not list any SSC pools).
Figure 1. onstat -g spi output

```
Spin locks with waits:
Num Waits   Num Loops   Avg Loop/Wait    Name
34477       387761      11.25            mtcb sleeping_lock
312         10205       32.71            mtcb vproc_list_lock
```

If you see an excessive number of longspins (**Num Loops** column) on an SSC pool, increase the number of SSC pools in the STMT_CACHE_NUMPOOL configuration parameter to improve performance.

Related information:
STMT_CACHE_NUMPOOL configuration parameter

Size of SQL statement cache pools and the current cache

Use the **onstat -g ssc pool** option to monitor the usage of each SQL statement cache (SSC) pool.

The **onstat -g ssc pool** command displays the size of each pool. The **onstat -g ssc** option displays the cumulative size of the SQL statement cache in the **currsize** column. This current size is the size of memory allocated from the SSC pools by the statements that are inserted into the cache. Because not all statements that allocate memory from the SSC pools are inserted into the cache, the current cache size could be smaller than the total size of the SSC pools. Normally, the total size of all SSC pools does not exceed the STMT_CACHE_SIZE value.

Figure 1 shows sample output for **onstat -g ssc pool**.
Figure 1. onstat -g ssc pool output

```
onstat -g ssc pool


Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
sscpool0     V     a7e4020  57344      2352     52          7
```

```
Blkpool Summary:
name         class addr      size      #blks
```

The `Pool Summary` section of the **onstat -g ssc pool** output lists the following information for each pool in the cache.

Column	Description
name	The name of the SQL statement cache (SSC) pool
class	The shared-memory segment type in which the pool has been created. For SSC pools, this value is always “V” for the virtual portion of shared-memory.
addr	The shared-memory address of the SSC pool structure
totalsize	The total size, in bytes, of this SSC pool
freesize	the number of free bytes in this SSC pool
#allocfrag	The number of contiguous areas of memory in this SSC pool that are allocated
#freefrag	The number of contiguous areas of memory that are not used in this SSC pool

The `Blkpool Summary` section of the **onstat -g ssc poo**l output lists the following information for all pools in the cache**.**

Column	Description
name	The name of the SSC pool
class	The shared-memory segment type in which the pool has been created. For SSC pools, this value is always “V” for the virtual portion of shared-memory.
addr	The shared-memory address of the SSC pool structure
totalsize	The total size, in bytes, of this SSC pool
#blks	The number of 8-kilobyte blocks that make up all the SSC pools

SQL statement cache information in onstat -g ssc output

The **onstat -g ssc** command displays summary information for the SQL statement cache.

The **onstat -g ssc** command displays the following information for the SQL statement cache.

Table 1. SQL statement cache information in **onstat -g ssc** output

Column	Description
#lrus	The number of LRU queues. Multiple LRU queues facilitate concurrent lookup and insertion of cache entries.
currsize	The number of bytes currently allocated to entries in the SQL statement cache
maxsize	The number of bytes specified in the STMT_CACHE_SIZE configuration parameter
poolsize	The cumulative number of bytes for all pools in the SQL statement cache. Use the **onstat -g ssc pool** option to monitor individual pool usage.
#hits	Setting of the STMT_CACHE_HITS configuration parameter, which specifies the number of times that a query is executed before it is inserted into the cache
nolimit	Setting of STMT_CACHE_NOLIMIT configuration parameter

The **onstat -g ssc** command lists the following information for each fully cached entry in the cache. The **onstat -g ssc all** option lists the following information for both the fully cached entries and key-only entries.

Column	Description
lru	The LRU identifier
hash	The hash-bucket identifier
ref_cnt	The number of sessions currently using this statement
hits	The number of times that users read the query from the cache, excluding the first time the statement entered the cache

Column	Description
flags	Shows flag codes. The flag codes for position 1 are: D Indicates that the statement has been dropped A statement in the cache can be dropped (not used any more) when one of its dependencies has changed. For example, when you run UPDATE STATISTICS for the table, the optimizer statistics might change, making the query plan for the SQL statement in the cache obsolete. In this case, the database server marks the statement as dropped the next time that it tries to use it. - Indicates that the statement has not been dropped The flag codes for position 2 are: F Indicates that the cache entry is fully cached and contains the memory structures for the query I Indicates that the statement is in the process of being moved to a fully cached state - Indicates that the statement is not fully cached A statement is not fully cached when the number of times the statement has been executed is less than the value of the STMT_CACHE_HITS configuration parameter. Entries with this - value in the second position appear in the **onstat -g ssc all** but not in the **onstat -g ssc** output.
heap_ptr	Pointer to the associated heap for the statement
database	Database against which the SQL statement is executed
user	User executing the SQL statement
statement	Statement text as it would be used to test for a match

Session memory

The database server uses the virtual portion of shared memory mainly for user sessions. Most of the memory that each user session allocates is for SQL statements. You can determine which session and which statements are using large amounts of memory. If necessary, you can set the SESSION_LIMIT_MEMORY configuration parameter to limit the amount of memory available to a session.

Use the following utility options to determine which session and prepared SQL statements are using large amounts of memory:

- **onstat -g mem**
- **onstat -g stm**

The **onstat -g mem** option displays memory usage of all sessions. You can find the session that is using the most memory by looking at the **totalsize** and **freesize** output columns. The following figure shows sample output for **onstat -g mem**. This sample output shows the memory use for three user sessions with the values `14`, `16`, `17` in the **names** output column.
Figure 1. **onstat -g mem** output

```
onstat -g mem

Pool Summary:
name          class addr     totalsize freesize #allocfrag #freefrag
...
14            V     a974020  45056     11960    99         10
16            V     a9ea020  90112     10608    159        5
17            V     a973020  45056     11304    97         13
...
Blkpool Summary:
name          class addr     size      #blks
mt            V     a235688  798720    19
global        V     a232800  0         0
```

To display the memory that is allocated by each prepared statement, use the **onstat -g stm** option. The following figure shows sample output for **onstat -g stm**.
Figure 2. **onstat -g stm** output

```
onstat -g stm


session   25 ------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 d36b018    9216  select sum(i) from t where i between -1 and ?
 d378018    6240  *select tabname from systables where tabid=7
 d36b114    8400  <SPL statement>
```

The **heapsz** column in the output in Figure 2 shows the amount of memory that is used by the statement. An asterisk (*) precedes the statement text if a cursor is open on the statement. The output does not show the individual SQL statements in an SPL routine.

To display the memory for only one session, specify the session ID in the **onstat -g stm** option. For an example, see Monitor session memory with onstat -g mem and onstat -g stm output.

Set the SESSION_LIMIT_MEMORY configuration parameter to limit how much memory a session can allocate, and can prevent individual sessions from monopolizing system resources. This limit does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

For example, to limit each session to 10 MB of memory, set `SESSION_LIMIT_MEMORY 102400` in the ONCONFIG file.

Related tasks:
Estimating the size of the virtual portion of shared memory
Related information:
SESSION_LIMIT_MEMORY configuration parameter

Data-replication buffers and memory utilization

Data replication requires two instances of the database server, a primary one and a secondary one, running on two computers. If you implement data replication for your database server, the database server holds logical-log records in the data-replication buffer before it sends them to the secondary database server.

The data-replication buffer is always the same size as the logical-log buffer.

Memory latches

The database server uses latches to control access to shared memory structures such as the buffer pool or the memory pools for the SQL statement cache. You can obtain statistics on latch use and information about specific latches. These statistics provide a measure of the system activity.

The statistics include the number of times that threads waited to obtain a latch. A large number of latch waits typically results from a high volume of processing activity in which the database server is logging most of the transactions.

Information about specific latches includes a listing of all the latches that are held by a thread and any threads that are waiting for latches. This information allows you to locate any specific resource contentions that exist.

You, as the database administrator, cannot configure or tune the number of latches. However, you can increase the number of memory structures on which the database server places latches to reduce the number of latch waits. For example, you can tune the number of SQL statement cache memory pools or the number of SQL statement cache LRU queues. For more information, see Multiple SQL statement cache pools.
Warning: Never stop a database server process that is holding a latch. If you do, the database server immediately initiates an abort.

- Monitoring latches with command-line utilities
  You can obtain information about latches by running **onstat -p** or **onstat -s**.
- Monitoring latches with SMI tables
  You can query the **sysprofile** SMI table to obtain the number of times a thread waited for a latch.

Monitoring latches with command-line utilities

You can obtain information about latches by running **onstat -p** or **onstat -s**.

- Monitoring latches with onstat -p
  Run **onstat -p** to obtain the values in the **lchwaits** field. This field stores the number of times that a thread was required to wait for a shared-memory latch.
- Monitoring latches with onstat -s
  Run **onstat -s** to obtain general latch information. The output includes the **userthread** column, which lists the address of any user thread that is waiting for a latch.

Monitoring latches with onstat -p

Run **onstat -p** to obtain the values in the **lchwaits** field. This field stores the number of times that a thread was required to wait for a shared-memory latch.

Figure 1 shows an excerpt of sample **onstat -p** output that shows the **lchwaits** field.
Figure 1. Partial onstat -p output showing the lchwaits field

```
...
ixda-RA  idx-RA   da-RA   logrec-RA    RA-pgsused  lchwaits
5        0        204      0           148         12
```

Related information:
onstat -p command: Print profile counts

Monitoring latches with onstat -s

Run **onstat -s** to obtain general latch information. The output includes the **userthread** column, which lists the address of any user thread that is waiting for a latch.

You can compare this address with the user addresses in the **onstat -u** output to obtain the user-process identification number.

Figure 1 shows sample **onstat -s** output.
Figure 1. onstat -s output

```
...
Latches with lock or userthread set
name     address  lock wait userthread
LRU1     402e90   0    0         6b29d8
bf[34]   4467c0   0    0         6b29d8
...
```


Monitoring latches with SMI tables

You can query the **sysprofile** SMI table to obtain the number of times a thread waited for a latch.

The **latchwts** column of the **sysprofile** table contains the number of times that a thread waited for a latch.

Encrypted values

An encrypted value uses more storage space than the corresponding plain text value because all of the information needed to decrypt the value except the encryption key is stored with the value.

Omitting the hint used with the password can reduce encryption overhead by up to 50 bytes. If you are using encrypted values, you must make sure that you have sufficient space available for the values.
Note: Embedding zero bytes in the encrypted result is not recommended.
Related information:
Column-level encryption
Calculating storage requirements for encrypted data

Effect of configuration on I/O activity

The configuration of your database server affects I/O activity.

The following factors affect I/O activity:

- The assignment of chunks and dbspaces can create I/O *hot spots*, or disk partitions with a disproportionate amount of I/O activity.
- Your allocation of critical data, sort areas, and areas for temporary files and index builds can place intermittent loads on various disks.
- How you configure read-ahead can increase the effectiveness of individual I/O operations.
- How you configure the background I/O tasks, such as logging and page cleaning, can affect I/O throughput.

- Chunk and dbspace configuration
  The number of disks that you use and the configuration of your chunks, dbspaces, and blobspaces affect the performance of your database server. You can improve performance by planning disk use and the configuration of chunks, dbspaces, and blobspaces.
- I/O for cooked files for dbspace chunks
  On UNIX, you can control the use of direct I/O for cooked files used for dbspace chunks.
- Placement of critical data
  The disk or disks that contain the system reserved pages, the physical log, and the dbspaces that contain the logical-log files are critical to the operation of the database server. The database server cannot operate if any of these elements becomes unavailable. By default, the database server places all three critical elements in the root dbspace.
- Configuration parameters that affect critical data
  The configuration parameters that configure the root dbspace and the logical and physical logs affect critical data.
- Configure dbspaces for temporary tables and sort files
  Applications that use temporary tables or large sort operations require a large amount of temporary space. To improve performance of these applications, use the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable to designate one or more dbspaces for temporary tables and sort files.
- Configure sbspaces for temporary smart large objects
  Applications can use temporary smart large objects for text, image, or other user-defined data types that are only required during the life of the user session. These applications do not require logging of the temporary smart large objects. Logging adds I/O activity to the logical log and increases memory utilization.
- Placement of simple large objects
  You can store simple large objects in either the same dbspace in which the table resides or in a blobspace.
- Factors that affect I/O for smart large objects
  An sbspace is a logical storage unit, composed of one or more chunks, in which you can store smart large objects (such as BLOB, CLOB, or multi representational data). Disk layout for sbspaces, the settings of certain configuration parameters, and some **onspaces** utility options affect I/O for smart large objects.
- Table I/O
  One of the most frequent functions that the database server performs is to bring data and index pages from disk into memory. Pages can be read individually for

brief transactions and sequentially for some queries. You can configure the number of pages that the database server brings into memory, and you can configure the timing of I/O requests for sequential scans.

- Configuration parameters that affect table I/O
  The AUTO_READAHEAD configuration parameter changes the automatic read-ahead mode or disables automatic read-ahead for a query. In addition, the DATASKIP configuration parameter enables or disables data skipping.
- Background I/O activities
  Background I/O activities do not service SQL requests directly. Many of these activities are essential to maintain database consistency and other aspects of database server operation. However, they create overhead in the CPU and take up I/O bandwidth.

Chunk and dbspace configuration

The number of disks that you use and the configuration of your chunks, dbspaces, and blobspaces affect the performance of your database server. You can improve performance by planning disk use and the configuration of chunks, dbspaces, and blobspaces.

All the data that resides in a database is stored on disk. The speed at which the database server can copy the appropriate data pages to and from disk determines how well your application performs.

Disks are typically the slowest component in the I/O path for a transaction or query that runs entirely on one host computer. Network communication can also introduce delays in client/server applications, but these delays are typically outside the control of the database server administrator. For information about actions that the database server administrator can take to improve network communications, see Network buffer pools and Connections and CPU utilization.

Disks can become overused or saturated when users request pages too often. Saturation can occur in the following situations:

- You use a disk for multiple purposes, such as for both logging and active database tables.
- Disparate data resides on the same disk.
- Table extents become interleaved.

The various functions that your application requires, as well as the consistency-control functions that the database server performs, determine the optimal disk, chunk, and dbspace layout for your application. The more disks that you make available to the database server, the easier it is to balance I/O across them. For more information about these factors, see Table performance considerations.

This section outlines important issues for the initial configuration of your chunks, dbspaces, and blobspaces. Consider the following issues when you decide how to lay out chunks and dbspaces on disks:

- Placement and mirroring of critical data
- Load balancing
- Reduction of contention
- Ease of backup and restore

Together with round-robin fragmentation, you can balance chunks over disks and controllers, saving time and handling errors. Placing multiple chunks on a single disk can improve throughput.

- Associate disk partitions with chunks
  You should assign chunks to entire disk partitions. When a chunk coincides with a disk partition (or device), it is easy to track disk-space use, and you avoid errors caused by miscalculated offsets.
- Associate dbspaces with chunks
  You should associate a single chunk with a dbspace, especially when that dbspace is to be used for a table fragment.
- Placing system catalog tables with database tables
  When a disk that contains the system catalog for a particular database fails, the entire database remains inaccessible until the system catalog is restored. Because of this potential inaccessibility, do not cluster the system catalog tables for all databases in a single dbspace. Instead place the system catalog tables with the database tables that they describe.

Associate disk partitions with chunks

You should assign chunks to entire disk partitions. When a chunk coincides with a disk partition (or device), it is easy to track disk-space use, and you avoid errors caused by miscalculated offsets.

The maximum size for a chunk is 4 terabytes.

Associate dbspaces with chunks

You should associate a single chunk with a dbspace, especially when that dbspace is to be used for a table fragment.

For more information about table placement and layout, see Table performance considerations.

Placing system catalog tables with database tables

When a disk that contains the system catalog for a particular database fails, the entire database remains inaccessible until the system catalog is restored. Because of this potential inaccessibility, do not cluster the system catalog tables for all databases in a single dbspace. Instead place the system catalog tables with the database tables that they describe.

To create a system catalog table in the table dbspace:

1. Create a database in the dbspace in which the table is to reside.
2. Use the SQL statements DATABASE or CONNECT to make that database the current database.
3. Enter the CREATE TABLE statement to create the table.

I/O for cooked files for dbspace chunks

On UNIX, you can control the use of direct I/O for cooked files used for dbspace chunks.

On UNIX, you can allocate disk space in two ways:

- Use files that are buffered through the operating system. These files are often called *cooked files*.
- Use unbuffered disk access, also called *raw* disk space.

When dbspaces reside on raw disk devices (also called *character-special devices*), the database server uses unbuffered disk access. A raw disk directly transfers data between the database server memory and disk without also copying data.

While you should generally use raw disk devices on UNIX systems to achieve better performance, you might prefer to use cooked files, which are easier to allocate and manage than raw devices. If you use cooked files, you might be able to get better performance by enabling the Informix® direct I/O option.

In addition, Informix supports a separate concurrent I/O option on AIX® operating systems. If you enable concurrent I/O on AIX, you get both unbuffered I/O and concurrent I/O. With concurrent I/O, writing to two parts of a file can occur concurrently. (On some other operating systems and file systems, enabling direct I/O also enables concurrent I/O as part of the same file system direct I/O feature.)

To determine the best performance, perform benchmark testing for the dbspace and table layout on your system.

- Direct I/O (UNIX)
  On UNIX, you can use direct I/O to improve the performance of cooked files. Direct I/O can be beneficial because it avoids file system buffering. Because direct I/O uses unbuffered I/O, it is more efficient for reads and writes that go to disk (as opposed to those reads and writes that merely access the file system buffers).
- Direct I/O (Windows)
  Direct I/O is used for dbspace chunks on Windows platforms regardless of the value of the DIRECT_IO configuration parameter.
- Concurrent I/O (AIX only)
  On AIX operating systems, you can use concurrent I/O in addition to direct I/O for chunks that use cooked files. Concurrent I/O can improve performance, because it allows multiple reads and writes to a file to occur concurrently, without the usual serialization of noncompeting read and write operations.
- Enabling the direct I/O or concurrent I/O option (UNIX)
  Use the DIRECT_IO configuration parameter to enable the direct I/O option on UNIX or the concurrent I/O option on AIX.
- Confirming the use of the direct or concurrent I/O option (UNIX)
  You can confirm and monitor the use of direct I/O or concurrent I/O (on AIX) for cooked file chunks.

Direct I/O (UNIX)

On UNIX, you can use direct I/O to improve the performance of cooked files. Direct I/O can be beneficial because it avoids file system buffering. Because direct I/O uses unbuffered I/O, it is more efficient for reads and writes that go to disk (as opposed to those reads and writes that merely access the file system buffers).

Direct I/O generally requires data to be aligned on disk sector boundaries.

Direct I/O also allows the use of kernel asynchronous I/O (KAIO), which can further improve performance. By using direct I/O and KAIO where available, the performance of cooked files used for dbspace chunks can approach the performance of raw devices.

If your file system supports direct I/O for the page size used for the dbspace chunk, the database server operates as follows:

- Does not use direct I/O by default.
- Uses direct I/O if the DIRECT_IO configuration parameter is set to `1`.
- Uses KAIO (if the file system supports it) with direct I/O by default.
- Does not use KAIO with direct I/O if the environment variable KAIOOFF is set.

If Informix® uses direct I/O for a chunk, and another program tries to open the chunk file without using direct I/O, the open will normally succeed, but there can be a performance penalty. The penalty can occur because the file system attempts to ensure that each open sees the same file data, either by switching to buffered I/O and not using direct I/O for the duration of the conflicting open, or by flushing the file system cache before each direct I/O operation and invalidating the file system cache after each direct write.

Informix does not use direct I/O for temporary dbspaces.

Related information:
DIRECT_IO configuration parameter (UNIX)

Direct I/O (Windows)

Direct I/O is used for dbspace chunks on Windows platforms regardless of the value of the DIRECT_IO configuration parameter.

Concurrent I/O (AIX only)

On AIX® operating systems, you can use concurrent I/O in addition to direct I/O for chunks that use cooked files. Concurrent I/O can improve performance, because it allows multiple reads and writes to a file to occur concurrently, without the usual serialization of noncompeting read and write operations.

Concurrent I/O can be especially beneficial when you have data in a single chunk file striped across multiple disks.

Concurrent I/O, which you enable by setting the DIRECT_IO configuration parameter to `2`, includes the benefit of avoiding file system buffering and is subject to the same limitations and use of KAIO as occurs if you use direct I/O without concurrent I/O. Thus, when concurrent I/O is enabled, you get both unbuffered I/O and concurrent I/O.

If Informix® uses concurrent I/O for a chunk, and another program (such as an external backup program) tries to open the same chunk file without using concurrent I/O, the open operation will fail.

Informix does not use direct or concurrent I/O for cooked files used for temporary dbspace chunks.

Related information:
DIRECT_IO configuration parameter (UNIX)

Enabling the direct I/O or concurrent I/O option (UNIX)

Use the DIRECT_IO configuration parameter to enable the direct I/O option on UNIX or the concurrent I/O option on AIX®.

Prerequisites:

- You must log on as user **root** or **informix**.
- Direct I/O or concurrent I/O must be available and the file system must support direct I/O for the page size used for the dbspace chunk.

To enable direct I/O, set the DIRECT_IO configuration parameter to `1`.

To enable concurrent I/O with direct I/O on AIX operating systems, set the DIRECT_IO configuration parameter to `2`.

If you do not want to enable direct I/O or concurrent I/O, set the DIRECT_IO configuration parameter to `0`.

Related information:
DIRECT_IO configuration parameter (UNIX)

Confirming the use of the direct or concurrent I/O option (UNIX)

You can confirm and monitor the use of direct I/O or concurrent I/O (on AIX®) for cooked file chunks.

You can confirm the use of direct I/O or concurrent I/O by:

- Displaying **onstat -d** information.
  The **onstat -d** command displays information that includes a flag that identifies whether direct I/O, concurrent I/O (on AIX), or neither is used for cooked file chunks.
- Verifying that the DIRECT_IO configuration parameter is set to `1` (for direct I/O) or `2` (for concurrent I/O).

Related information:
DIRECT_IO configuration parameter (UNIX)
onstat -d command: Print chunk information

Placement of critical data

The disk or disks that contain the system reserved pages, the physical log, and the dbspaces that contain the logical-log files are critical to the operation of the database server. The database server cannot operate if any of these elements becomes unavailable. By default, the database server places all three critical elements in the root

dbspace.

To arrive at an appropriate placement strategy for critical data, you must make a trade-off between the availability of data and maximum logging performance.

The database server also places temporary table and sort files in the root dbspace by default. You should use the DBSPACETEMP configuration parameter and the **DBSPACETEMP** environment variable to assign these tables and files to other dbspaces. For details, see Configure dbspaces for temporary tables and sort files.

- Consider separate disks for critical data components
  If you place the root dbspace, logical log, and physical log in separate dbspaces on separate disks, you can obtain some distinct performance advantages. The disks that you use for each critical data component should be on separate controllers.
- Consider mirroring for critical data components
  Consider mirroring for the dbspaces that contain critical data. Mirroring these dbspaces ensures that the database server can continue to operate even when a single disk fails.

Consider separate disks for critical data components

If you place the root dbspace, logical log, and physical log in separate dbspaces on separate disks, you can obtain some distinct performance advantages. The disks that you use for each critical data component should be on separate controllers.

This approach has the following advantages:

- Isolates logging activity from database I/O and allows physical-log I/O requests to be serviced in parallel with logical-log I/O requests
- Reduces the time that you need to recover from a failure
  However, unless the disks are mirrored, there is an increased risk that a disk that contains critical data might be affected in the event of a failure, which will bring the database server to a halt and require the complete restoration of all data from a level-0 backup.
- Allows for a relatively small root dbspace that contains only reserved pages, the database partition, and the **sysmaster** database
  In many cases, 10,000 kilobytes is sufficient.

The database server uses different methods to configure various portions of critical data. To assign an appropriate dbspace for the root dbspace and physical log, set the appropriate database server configuration parameters. To assign the logical-log files to an appropriate dbspace, use the **onparams** utility.

For more information about the configuration parameters that affect each portion of critical data, see Configuration parameters that affect critical data.

Consider mirroring for critical data components

Consider mirroring for the dbspaces that contain critical data. Mirroring these dbspaces ensures that the database server can continue to operate even when a single disk fails.

However, depending on the mix of I/O requests for a given dbspace, a trade-off exists between the fault tolerance of mirroring and I/O performance. You obtain a marked performance advantage when you mirror dbspaces that have a read-intensive usage pattern and a slight performance disadvantage when you mirror write-intensive dbspaces.

Most modern storage devices have excellent mirroring capabilities, and you can use those devices instead of the mirroring capabilities of the database server.

When mirroring is in effect, two disks are available to handle read requests, and the database server can process a higher volume of those requests. However, each write request requires two physical write operations and does not complete until both physical operations are performed. The write operations are performed in parallel, but the request does not complete until the slower of the two disks performs the update. Thus, you experience a slight performance penalty when you mirror write-intensive dbspaces.

- Consider mirroring the root dbspace
  You can achieve a certain degree of fault tolerance with a minimum performance penalty if you mirror the root dbspace and restrict its contents to read-only or seldom-accessed tables.
- Consider mirroring smart-large-object chunks
  You can achieve higher availability and faster access if you mirror chunks that contain metadata pages.
- Mirroring and its effect on the logical log
  The logical log is write intensive. If the dbspace that contains the logical-log files is mirrored, you encounter a slight double-write performance penalty. However, you can adjust the rate at which logging generates I/O requests to a certain extent by choosing an appropriate log buffer size and logging mode.
- Mirroring and its effect on the physical log
  The physical log is write intensive, with activity occurring at checkpoints and when buffered data pages are flushed. I/O to the physical log also occurs when a page-cleaner thread is activated. If the dbspace that contains the physical log is mirrored, you encounter a slight double-write performance penalty.

Consider mirroring the root dbspace

You can achieve a certain degree of fault tolerance with a minimum performance penalty if you mirror the root dbspace and restrict its contents to read-only or seldom-accessed tables.

When you place update-intensive tables in other, nonmirrored dbspaces, you can use the database server backup-and-restore facilities to perform warm restores of those tables in the event of a disk failure. When the root dbspace is mirrored, the database server remains online to service other transactions while the failed disk is being repaired.

When you mirror the root dbspace, always place the first chunk on a different device than that of the mirror. The MIRRORPATH configuration parameter should have a different value than ROOTPATH.

Related information:
MIRRORPATH configuration parameter
ROOTPATH configuration parameter

Consider mirroring smart-large-object chunks

You can achieve higher availability and faster access if you mirror chunks that contain metadata pages.

An sbspace is a logical storage unit composed of one or more chunks that store *smart large objects*, which consist of CLOB (character large object) or BLOB (binary large object) data.

The first chunk of an sbspace contains a special set of pages, called *metadata*, which is used to locate smart large objects in the sbspace. Additional chunks that are added to the sbspace can also have metadata pages if you specify them on the **onspaces** command when you create the chunk.

Consider mirroring chunks that contain metadata pages for the following reasons:

- Higher availability
  Without access to the metadata pages, users cannot access any smart large objects in the sbspace. If the first chunk of the sbspace contains all of the metadata pages and the disk that contains that chunk becomes unavailable, you cannot access a smart large object in the sbspace, even if it resides on a chunk on another disk. For high availability, mirror at least the first chunk of the sbspace and any other chunk that contains metadata pages.
- Faster access
  By mirroring the chunk that contains the metadata pages, you can spread read activity across the disks that contain the primary chunk and mirror chunk.

Related information:
Sbspaces

Mirroring and its effect on the logical log

The logical log is write intensive. If the dbspace that contains the logical-log files is mirrored, you encounter a slight double-write performance penalty. However, you can adjust the rate at which logging generates I/O requests to a certain extent by choosing an appropriate log buffer size and logging mode.

For details on the slight double-write performance penalty, see Consider mirroring for critical data components.

With unbuffered and ANSI-compliant logging, the database server requests a flush of the log buffer to disk for every committed transaction (two when the dbspace is mirrored). Buffered logging generates far fewer I/O requests than unbuffered or ANSI-compliant logging.

With buffered logging, the log buffer is written to disk only when it fills and all the transactions that it contains are completed. You can reduce the frequency of logical-log I/O even more if you increase the size of your logical-log buffers. However, buffered logging leaves transactions in any partially filled buffers vulnerable to loss in the event of a system failure.

Although database consistency is guaranteed under buffered logging, specific transactions are not guaranteed against a failure. The larger the logical-log buffers, the more transactions you might need to reenter when service is restored after a failure.

Unlike the physical log, you cannot specify an alternative dbspace for logical-log files in your initial database server configuration. Instead, use the **onparams** utility first to add logical-log files to an alternative dbspace and then drop logical-log files from the root dbspace.

Related information:
The onparams Utility

Mirroring and its effect on the physical log

The physical log is write intensive, with activity occurring at checkpoints and when buffered data pages are flushed. I/O to the physical log also occurs when a page-cleaner thread is activated. If the dbspace that contains the physical log is mirrored, you encounter a slight double-write performance penalty.

For details on the slight double-write performance penalty, see Consider mirroring for critical data components.

To keep I/O to the physical log at a minimum, you can adjust the checkpoint interval and the LRU minimum and maximum thresholds. (See CKPTINTVL and its effect on checkpoints and BUFFERPOOL and its effect on page cleaning.)

Configuration parameters that affect critical data

The configuration parameters that configure the root dbspace and the logical and physical logs affect critical data.

You can use the following configuration parameters to configure the root dbspace:

- ROOTNAME
- ROOTOFFSET
- ROOTPATH
- ROOTSIZE
- MIRROR
- MIRRORPATH
- MIRROROFFSET

These parameters determine the location and size of the initial chunk of the root dbspace and configure mirroring, if any, for that chunk. (If the initial chunk is mirrored, all other chunks in the root dbspace must also be mirrored). Otherwise, these parameters have no major impact on performance.

The following configuration parameters affect the logical logs:

- LOGSIZE
- LOGBUFF

The LOGSIZE configuration parameter determines the size of each logical-log files. The LOGBUFF configuration parameter determines the size of the three logical-log buffers that are in shared memory.

The PHYSFILE configuration parameter determines the initial size of the physical log in rootdbs. This configuration parameter is used only when the instance is created.

Related concepts:
The LOGBUFF configuration parameter and memory utilization
Checkpoints and the physical log

Configure dbspaces for temporary tables and sort files

Applications that use temporary tables or large sort operations require a large amount of temporary space. To improve performance of these applications, use the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable to designate one or more dbspaces for temporary tables and sort files.

Depending on how the temporary space is created, the database server uses the following default locations for temporary table and sort files when you do not set DBSPACETEMP:

- The dbspace of the current database, when you create an explicit temporary table with the TEMP TABLE clause of the CREATE TABLE statement and do not specify a dbspace for the table either in the IN dbspace clause or in the FRAGMENT BY clause
  This action can severely affect I/O to that dbspace. If the root dbspace is mirrored, you encounter a slight double-write performance penalty for I/O to the temporary tables and sort files.
- The root dbspace when you create an explicit temporary table with the INTO TEMP option of the SELECT statement
  This action can severely affect I/O to the root dbspace. If the root dbspace is mirrored, you encounter a slight double-write performance penalty for I/O to the temporary tables and sort files.
- The operating-system directory or file that you specify in one of the following variables:
  - In UNIX, the operating-system directory or directories that the **PSORT_DBTEMP** environment variable specifies, if it is set
    If **PSORT_DBTEMP** is not set, the database server writes sort files to the operating-system file space in the **/tmp** directory.
  - In Windows, the directory specified in **TEMP** or **TMP** in the User Environment Variables window on **Control Panel > System.**

  The database server uses the operating-system directory or files to direct any overflow that results from the following database operations:
  - SELECT statement with GROUP BY clause
  - SELECT statement with ORDER BY clause
  - Hash-join operation
  - Nested-loop join operation
  - Index builds

Warning: If you do not specify a value for the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable, the database server uses this operating-system file for implicit temporary tables. If this file system has insufficient space to hold a sort file, the query that performs the sort returns an error. Meanwhile, the operating system might be severely impacted until you remove the sort file.
You can improve performance with the use of temporary dbspaces that you create exclusively to store temporary tables and sort files. Use the DBSPACETEMP configuration parameter and the **DBSPACETEMP** environment variable to assign these tables and files to temporary dbspaces.

When you specify dbspaces in either the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable, you gain the following performance advantages:

- Reduced I/O impact on the root dbspace, production dbspaces, or operating-system files
- Use of parallel sorts into the temporary files (to process query clauses such as ORDER BY or GROUP BY, or to sort index keys when you execute CREATE INDEX) when you specify more than one dbspace for temporary tables and PDQ priority is set to greater than 0.
- Improved speed with which the database server creates temporary tables when you assign two or more temporary dbspaces on separate disks
- Automatic fragmentation of the temporary tables across dbspaces when SELECT....INTO TEMP statements are run

The following table shows statements that create temporary tables and information about where the temporary tables are created.

Statement That Creates Temporary Table	Database Logged	WITH NO LOG clause	FRAGMENT BY clause	Where Temp Table Created
CREATE TEMP TABLE	Yes	No	No	Root dbspace
CREATE TEMP TABLE	Yes	Yes	No	One of dbspaces that are specified in DBSPACETEMP
CREATE TEMP TABLE	Yes	No	Yes	Cannot create temp table. Error 229/196
SELECT ..INTO TEMP	Yes	Yes	No	Fragmented by round-robin only in the non-logged dbspaces that are specified in DBSPACETEMP

Important: Use the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable for better performance of sort operations and to prevent the database server from unexpectedly filling file systems. The dbspaces that you list must be composed of chunks that are allocated as unbuffered devices.

- Creating temporary dbspaces
  You can create a dbspace for the exclusive use of temporary tables and sort files. The database server does not perform logical or physical logging of temporary dbspaces, and temporary dbspaces are never backed up as part of a full-system backup.
- Specify temporary tables in the DBSPACETEMP configuration parameter
  The DBSPACETEMP configuration parameter specifies a list of dbspaces in which the database server places temporary tables and sort files by default. Some or all of the dbspaces that you list in this configuration parameter can be temporary dbspaces, which are reserved exclusively to store temporary tables and sort files.
- Override the DBSPACETEMP configuration parameter for a session
  To override the DBSPACETEMP configuration parameter, you can use the **DBSPACETEMP** environment variable for both temporary tables and sort files. This environment variable specifies a comma- or colon-separated list of dbspaces in which to place temporary tables for the current session.
- Estimating temporary space for dbspaces and hash joins
  You can estimate and increase the amount of temporary space for dbspaces and for hash joins. If you do this, you can prevent the possible overflow of memory to temporary space on disk.
- PSORT_NPROCS environment variable
  The **PSORT_NPROCS** environment variable specifies the maximum number of threads that the database server can use to sort a query. When a query involves a large sort operation, multiple sort threads can execute in parallel to improve the performance of the query.

Related concepts:
Specify temporary tables in the DBSPACETEMP configuration parameter
Related information:
DBSPACETEMP configuration parameter
CREATE TEMP TABLE statement
INTO TEMP clause

Creating temporary dbspaces

You can create a dbspace for the exclusive use of temporary tables and sort files. The database server does not perform logical or physical logging of temporary dbspaces, and temporary dbspaces are never backed up as part of a full-system backup.

To create a dbspace for the exclusive use of temporary tables and sort files, use **onspaces -t**. For best performance, use the following guidelines:

- If you create more than one temporary dbspace, create each dbspace on a separate disk to balance the I/O impact.
- Place no more than one temporary dbspace on a single disk.

You cannot mirror a temporary dbspace that you create with **onspaces -t**.
Important: In the case of a database with logging, you must include the WITH NO LOG clause in the SELECT... INTO TEMP statement to place the explicit temporary tables in the dbspaces listed in the DBSPACETEMP configuration parameter and the **DBSPACETEMP** environment variable. Otherwise, the database server stores the explicit temporary tables in the root dbspace.
Related information:
DBSPACETEMP configuration parameter
create tempdbspace argument: Create a temporary dbspace (SQL administration API)
onspaces -c -d: Create a dbspace

Specify temporary tables in the DBSPACETEMP configuration parameter

The DBSPACETEMP configuration parameter specifies a list of dbspaces in which the database server places temporary tables and sort files by default. Some or all of the dbspaces that you list in this configuration parameter can be temporary dbspaces, which are reserved exclusively to store temporary tables and sort files.

If the database server inserts data into a temporary table through a SELECT INTO TEMP operation that creates the TEMP table, that temporary table uses round-robin distributed storage. Its fragments are created in the temporary dbspaces that are listed in the DBSPACETEMP configuration parameter or in the **DBSPACETEMP** environment variable. For example, the following query uses round-robin distributed storage:

```
SELECT col1 FROM tab1
   INTO TEMP temptab1 WITH NO LOG;
```

The DBSPACETEMP configuration parameter lets the database administrator restrict which dbspaces the database server uses for temporary storage.
Important: The DBSPACETEMP configuration parameter is not set in the **onconfig.std** file. For best performance with temporary tables and sort files, use DBSPACETEMP to specify two or more dbspaces on separate disks.

Tips:

- If you work on a small system with a limited number of disks and cannot place temporary dbspaces on different disk drives, you might consider using 1 (or possibly 2) temporary dbspaces. This can reduce the logging that is associated with the temporary dbspaces.
- If you have many disk drives, you can parallelize many operations (such as sorts, joins, and temporary tables) without having multiple temporary dbspaces. The number of temporary dbspaces that you have relates to how much you want to spread the I/O out. A good starting place is 4 temporary dbspaces. If you create too many small temporary dbspaces, you will not have enough space for nonparallel creation of large objects.

Related concepts:
Configure dbspaces for temporary tables and sort files
Distribution schemes
Related information:
DBSPACETEMP configuration parameter
CREATE TEMP TABLE statement

Override the DBSPACETEMP configuration parameter for a session

To override the DBSPACETEMP configuration parameter, you can use the **DBSPACETEMP** environment variable for both temporary tables and sort files. This environment variable specifies a comma- or colon-separated list of dbspaces in which to place temporary tables for the current session.

Important: Use the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable for better performance of sort operations and to prevent the database server from unexpectedly filling file systems.
You should use DBSPACETEMP rather than the **PSORT_DBTEMP** environment variable to specify sort files for the following reasons:

- **DBSPACETEMP** typically yields better performance.
  When dbspaces reside on character-special devices (also known as raw disk devices), the database server uses unbuffered disk access. I/O is faster to unbuffered devices than to regular (buffered) operating-system files because the database server manages the I/O operation directly.
- **PSORT_DBTEMP** specifies one or more operating-system directories in which to place sort files.
  These operating-system files can unexpectedly fill on your computer because the database server does not manage them.

Estimating temporary space for dbspaces and hash joins

You can estimate and increase the amount of temporary space for dbspaces and for hash joins. If you do this, you can prevent the possible overflow of memory to temporary space on disk.

You can use the following guidelines to estimate the amount of temporary space to allocate:

- For OLTP applications, allocate temporary dbspaces that equal at least 10 percent of the table.
- For DSS applications, allocate temporary dbspaces that equal at least 50 percent of the table.

A hash join, which works by building a table (the hash table) from the rows in one of the tables in a join, and then probing it with rows from the other table, can use a significant amount of memory and can potentially overflow to temporary space on disk. The hash table size is governed by the size of the table used to build the hash table (which is often the smaller of the two tables in the join), after applying any filters, which can reduce the number of rows and possibly reduce the number of columns.

Hash-join partitions are organized into pages. Each page has a header. The header and tuples are larger in databases on 64-bit platforms than in builds on 32-bit platforms. The size of each page is the base page size (2K or 4K depending on system) unless a single row needs more space. If you need more space, you can add bytes to the length of your rows.

You can use the following formula to estimate the amount of memory that is required for the hash table in a hash join:

```
hash_table_size = (32 bytes + row_size_smalltab) * num_rows_smalltab
```

where `row_size_smalltab` and `num_rows_smalltab` refer to the row size and the number of rows, respectively, in the smaller of the two tables participating in the hash join.
For example, suppose you have a page head that is 80 bytes in length and a row header that is 48 bytes in length. Because each row must be aligned to 8 bytes, you might need to add up to 7 bytes to the row length, as shown in these formulas:

```
per_row_size = 48 bytes + rowsize + mod(rowsize, 8)
page_size = base_page_size (2K or 4K)
rows_per_page = round_down_to_integer((page_size - 80 bytes) / per_row_size)
```

If the value of `rows_per_page` is less than one, increase the `page_size` value to the smallest multiple of the `base_page_size`, as shown in this formula:

```
size = (numrows_smalltab / rows_per_page) * page_size
```

You can use the DS_NONPDQ_QUERY_MEM configuration parameter to configure sort memory for all queries except PDQ queries. Its setting has no effect, however, if the PDQ priority setting is greater than zero.

For more information, see Hash join and Configuring memory for queries with hash joins, aggregates, and other memory-intensive elements.

Related information:
DS_NONPDQ_QUERY_MEM configuration parameter

PSORT_NPROCS environment variable

The **PSORT_NPROCS** environment variable specifies the maximum number of threads that the database server can use to sort a query. When a query involves a large sort operation, multiple sort threads can execute in parallel to improve the performance of the query.

When the value of PDQ priority is `0` and **PSORT_NPROCS** is greater than 1, the database server uses parallel sorts. The management of PDQ does not limit these sorts. In other words, although the sort is executed in parallel, the database server does not regard sorting as a PDQ activity. When PDQ priority is `0`, the database server does not control sorting by any of the PDQ configuration parameters.

When PDQ priority is greater than 0 and **PSORT_NPROCS** is greater than 1, the query benefits both from parallel sorts and from PDQ features such as parallel scans and additional memory. Users can use the **PDQPRIORITY** environment variable to request a specific proportion of PDQ resources for a query. You can use the MAX_PDQPRIORITY parameter to limit the number of such user requests. For more information about MAX_PDQPRIORITY, see Limiting PDQ resources in queries.

The database server allocates a relatively small amount of memory for sorting, and that memory is divided among the **PSORT_NPROCS** sort threads. Sort processes use temporary space on disk when not enough memory is allocated. For more information about memory allocated for sorting, see Estimating memory needed for sorting.
Important: For better performance for a sort operation, set PSORT_NPROCS initially to 2 if your computer has multiple CPUs. If the subsequent CPU activity is lower than I/O activity, you can increase the value of PSORT_NPROCS.
For more information about sorts during index builds, see Improving performance for index builds.

Configure sbspaces for temporary smart large objects

Applications can use temporary smart large objects for text, image, or other user-defined data types that are only required during the life of the user session. These applications do not require logging of the temporary smart large objects. Logging adds I/O activity to the logical log and increases memory utilization.

You can store temporary smart large objects in a permanent sbspace or a temporary sbspace.

- Permanent sbspaces
  If you store the temporary smart large objects in a regular sbspace and keep the default no logging attribute, changes to the objects are not logged, but the metadata is always logged.
- Temporary sbspaces
  Applications that update temporary smart large objects stored in temporary sbspaces are significantly faster because the database server does not log the metadata or the user data in a temporary sbspace.

To improve performance of applications that update temporary smart large objects, specify the **LOTEMP** flag in the **mi_lo_specset_flags** or **ifx_lo_specset_flags** API function and specify a temporary sbspace for the temporary smart large objects. The database server uses the following order of precedence for locations to place temporary smart large objects:

- The sbspace you specify in the **mi_lo_specset_sbspace** or **ifx_lo_specset_sbspace** API function when you create the smart large object
  Specify a temporary sbspace in the API function so that changes to the objects and the metadata are not logged. The sbspace you specify in the API function overrides any default sbspaces that the SBSPACETEMP or SBSPACENAME configuration parameters might specify.
- The sbspace you specify in the IN Sbspace clause when you create an explicit temporary table with the TEMP TABLE clause of the CREATE TABLE statement
  Specify a temporary sbspace in the IN Sbspace clause so that changes to the objects and the metadata are not logged.
- The permanent sbspace you specify in the SBSPACENAME configuration parameter, if you do not specify an sbspace in the SBSPACETEMP configuration parameter

If no temporary sbspace is specified in any of the above methods, then the database server issues the following error message when you try to create a temporary smart large object:

```
-12053  Smart Large Objects: No sbspace number specified.
```

- Creating temporary sbspaces
  To create an sbspace for the exclusive use of temporary smart large objects, use **onspaces -c -S** with the **-t** option.
- Specify which sbspaces to use for temporary storage
  The SBSPACETEMP configuration parameter specifies a list of sbspaces in which the database server places temporary smart large objects by default. Some or all of the sbspaces that you list in this configuration parameter can be temporary sbspaces, which are reserved exclusively to store temporary smart large objects.

Creating temporary sbspaces

To create an sbspace for the exclusive use of temporary smart large objects, use **onspaces -c -S** with the **-t** option.

For best performance, use the following guidelines:

- If you create more than one temporary sbspace, create each sbspace on a separate disk to balance the I/O impact.
- Place no more than one temporary sbspace on a single disk.

The database server does not perform logical or physical logging of temporary sbspaces, and temporary sbspaces are never backed up as part of a full-system backup. You cannot mirror a temporary sbspace that you create with **onspaces -t**.
Important: In the case of a database with logging, you must include the WITH NO LOG clause in the SELECT... INTO TEMP statement to place the temporary smart large objects in the sbspaces listed in the SBSPACETEMP configuration parameter. Otherwise, the database server stores the temporary smart large objects in the sbspace listed in the SBSPACENAME configuration parameter.

Related information:
onspaces -c -S: Create an sbspace
Creating a temporary sbspace

Specify which sbspaces to use for temporary storage

The SBSPACETEMP configuration parameter specifies a list of sbspaces in which the database server places temporary smart large objects by default. Some or all of the sbspaces that you list in this configuration parameter can be temporary sbspaces, which are reserved exclusively to store temporary smart large objects.

Important: The SBSPACETEMP configuration parameter is not set in the **onconfig.std** file. For best performance with temporary smart large objects, use SBSPACETEMP to specify two or more sbspaces on separate disks.
Related information:
SBSPACETEMP configuration parameter

Placement of simple large objects

You can store simple large objects in either the same dbspace in which the table resides or in a blobspace.

A blobspace is a logical storage unit composed of one or more chunks that store only simple large objects (TEXT or BYTE data). For information about sbspaces, which store smart large objects (such as BLOB, CLOB, or multirepresentational data), see Factors that affect I/O for smart large objects.

If you use a blobspace, you can store simple large objects on a separate disk from the table with which the data is associated. You can store simple large objects associated with different tables in the same blobspace.

You can create a blobspace with the **onspaces** utility or with an SQL administration API command that uses the create blobspace argument with the **admin()** or **task()** function.

You assign simple large objects to a blobspace when you create the tables with which simple large objects are associated, using the CREATE TABLE statement.

Simple large objects are not logged and do not pass through the buffer pool. However, frequency of checkpoints can affect applications that access TEXT or BYTE data. For more information, see LOGSIZE and LOGFILES and their effect on checkpoints.

- Advantage of blobspaces over dbspaces
  If you store simple large objects in a blobspace on a separate disk from the table with which it is associated, instead of storing the objects in a dbspace, you can obtain some performance advantages.
- Blobpage size considerations
  Blobspaces are divided into units called *blobpages*. The database server retrieves simple large objects from a blobspace in blobpage-sized units. You specify the size of a blobpage in multiples of a disk page when you create the blobspace.

Related information:
CREATE TABLE statement
create blobspace argument: Create a blobspace (SQL administration API)
onspaces -c -b: Create a blobspace

Advantage of blobspaces over dbspaces

If you store simple large objects in a blobspace on a separate disk from the table with which it is associated, instead of storing the objects in a dbspace, you can obtain some performance advantages.

The performance advantages of storing simple large objects in a blobspace are:

- You have parallel access to the table and simple large objects.
- Unlike simple large objects stored in a dbspace, blobspace data is written directly to disk. Simple large objects do not pass through resident shared memory, which leaves memory pages free for other uses.
- Simple large objects are not logged, which reduces logging I/O activity for logged databases.

For more information, see Storing simple large objects in the tblspace or a separate blobspace.

Blobpage size considerations

Blobspaces are divided into units called *blobpages*. The database server retrieves simple large objects from a blobspace in blobpage-sized units. You specify the size of a blobpage in multiples of a disk page when you create the blobspace.

The optimal blobpage size for your configuration depends on the following factors:

- The size distribution of the simple large objects
- The trade-off between retrieval speed for your largest simple large object and the amount of disk space that is wasted by storing simple large objects in large blobpages

To retrieve simple large objects as quickly as possible, use the size of your largest simple large object rounded up to the nearest disk-page-sized increment. This scheme guarantees that the database server can retrieve even the largest simple large object in a single I/O request. Although this scheme guarantees the fastest retrieval, it has the potential to waste disk space. Because simple large objects are stored in their own blobpage (or set of blobpages), the database server reserves the same amount of disk space for every blobpage even if the simple large object takes up a fraction of that page. Using a smaller blobpage allows you to make better use of your disk, especially when large differences exist in the sizes of your simple large objects.

To achieve the greatest theoretical utilization of space on your disk, you can make your blobpage the same size as a standard disk page. Then many, if not most, simple large objects would require several blobpages. Because the database server acquires a lock and issues a separate I/O request for each blobpage, this scheme performs poorly.

In practice, a balanced scheme for sizing uses the most frequently occurring simple-large-object size as the size of a blobpage. For example, suppose that you have 160 simple-large-object values in a table with the following size distribution:

- Of these values, 120 are 12 kilobytes each.
- The other 40 values are 16 kilobytes each.

You can choose one of the following blobpage sizes:

- The 12-kilobyte blobpage size provides greater storage efficiency than a 16-kilobyte blobpage size, as the following two calculations show:
  - 12 kilobytes
    This configuration allows the majority of simple-large-object values to require a single blobpage and the other 40 values to require two blobpages. In this configuration, 8 kilobytes is wasted in the second blobpage for each of the larger values. The total wasted space is as follows:

    ```
    wasted-space = 8 kilobtyes * 40
                 = 329 kilobytes
    ```

  - 16 kilobytes
    In this configuration, 4 kilobytes is wasted in the extents of 120 simple large objects. The total wasted space is as follows:

    ```
    wasted-space = 4 kilobtyes * 120
                 = 640 kilobytes
    ```

- If your applications access the 16-kilobyte simple-large-object values more frequently, the database server must perform a separate I/O operation for each blobpage. In this case, the 16-kilobyte blobpage size provides better retrieval speed than a 12-kilobyte blobpage size.

The maximum number of pages that a blobspace can contain is 2147483647. Therefore, the size of the blobspace is limited to the blobpage size x 2147483647. This includes blobpages in all chunks that make up the blobspace.

Tip: If a table has more than one *simple-large-object column and the data values are not close in size, store t*he data in different blobspaces, each with an appropriately sized blobpage.

- Optimize blobspace blobpage size
  When you are evaluating blobspace storage strategy, you can measure efficiency by two criteria: blobpage fullness and the blobpages required per simple large object.
- Obtain blobspace storage statistics
  To help you determine the optimal blobpage size for each blobspace, use the **oncheck -pB** command.
- Determine blobpage fullness with oncheck -pB output
  The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. These statistics provide a measure of storage efficiency for individual simple large objects in a database or table.

Optimize blobspace blobpage size

When you are evaluating blobspace storage strategy, you can measure efficiency by two criteria: blobpage fullness and the blobpages required per simple large object.

Blobpage fullness refers to the amount of data within each blobpage. TEXT and BYTE data stored in a blobspace cannot share blobpages. Therefore, if a single simple large object requires only 20 percent of a blobpage, the remaining 80 percent of the page is unavailable for use.

However, avoid making the blobpages too small. When several blobpages are needed to store each simple large object, you increase the overhead cost of storage. For example, more locks are required for updates, because a lock must be acquired for each blobpage.

Obtain blobspace storage statistics

To help you determine the optimal blobpage size for each blobspace, use the **oncheck -pB** command.

The command lists the following statistics for each table (or database):

- The number of blobpages used by the table (or database) in each blobspace
- The average fullness of the blobpages used by each simple large object stored as part of the table (or database)

Determine blobpage fullness with oncheck -pB output

The **oncheck -pB** command displays statistics that describe the average fullness of blobpages. These statistics provide a measure of storage efficiency for individual simple large objects in a database or table.

If you find that the statistics for a significant number of simple large objects show a low percentage of fullness, the database server might benefit from changing the size of the blobpage in the blobspace.

Both the **oncheck -pB** and **onstat -d update** commands display the same information about the number of free blobpages. The **onstat -d update** command displays the same information as **onstat -d** and an accurate number of free blobpages for each blobspace chunk.

Execute **oncheck -pB** with either a database name or a table name as a parameter. The following example retrieves storage information for all simple large objects stored in the table **sriram.catalog** in the **stores_demo** database:

```
oncheck -pB stores_demo:sriram.catalog
```

oncheck -pB Output

Figure 1 shows the output of this command.
Figure 1. Output of **oncheck -pB**

```
              BLOBSpace Report for stores_demo:sriram.catalog

Total pages used by table            7

BLOBSpace usage:
Space    Page              Percent Full
Name     Number     Pages  0-25%  26-50%  51-75  76-100%
------------------------------------------------------------
blobPIC 0x300080  1       x
   blobPIC 0x300082  2       x
         ------
Page Size is 6144       3

bspc1    0x2000b2  2              x
bspc1    0x2000b6  2                      x
              ------
Page Size is 2048       4
```

Space Name is the name of the blobspace that contains one or more simple large objects stored as part of the table (or database).

Page Number is the starting address in the blobspace of a specific simple large object.

Pages is the number of the database server pages required to store this simple large object.

Percent Full is a measure of the average blobpage fullness, by blobspace, for each blobspace in this table or database.

Page Size is the size in bytes of the blobpage for this blobspace. Blobpage size is always a multiple of the database server page size.

The example output indicates that four simple large objects are stored as part of the table **sriram.catalog**. Two objects are stored in the blobspace **blobPIC** in 6144-byte blobpages. Two more objects are stored in the blobspace **bspc1** in 2048-byte blobpages.

The summary information that appears at the top of the display, **Total pages used by table** is a simple total of the blobpages needed to store simple large objects. The total says nothing about the size of the blobpages used, the number of simple large objects stored, or the total number of bytes stored.

The efficiency information displayed under the **Percent Full** heading is imprecise, but it can alert an administrator to trends in the storage of TEXT and BYTE data.

- Interpreting blobpage average fullness
  You can analyze the output of the **oncheck -pB** command to calculate average fullness.
- Analyzing efficiency criteria with oncheck -pB output
  You can analyze the output of the **oncheck -pB** command to determine if there is a more efficient storage strategy.

Interpreting blobpage average fullness

You can analyze the output of the **oncheck -pB** command to calculate average fullness.

The first simple large object listed in Determine blobpage fullness with oncheck -pB output is stored in the blobspace **blobPIC** and requires one 6144-byte blobpage. The blobpage is 51 to 75 percent full, meaning that the size is between 0.51 * 6144 = 3133 bytes and 0.75 * 6144 = 4608. The maximum size of this simple large object must be less than or equal to 75 percent of 6144 bytes, or 4608 bytes.

The second object listed under blobspace **blobPIC** requires two 6144-byte blobpages for storage, or a total of 12,288 bytes. The average fullness of all allocated blobpages is 51 to 75 percent. Therefore, the minimum size of the object must be greater than 50 percent of 12,288 bytes, or 6144 bytes. The maximum size of the simple large object must be less than or equal to 75 percent of 12,288 bytes, or 9216 bytes. The average fullness does not mean that each page is 51 to 75 percent full. A calculation would yield 51 to 75 percent average fullness for two blobpages where the first blobpage is 100 percent full and the second blobpage is 2 to 50 percent full.

Now consider the two simple large objects in blobspace **bspc1**. These two objects appear to be nearly the same size. Both objects require two 2048-byte blobpages, and the average fullness for each is 76 to 100 percent. The minimum size for these simple large objects must be greater than 75 percent of the allocated blobpages, or 3072 bytes. The maximum size for each object is slightly less than 4096 bytes (allowing for overhead).

Analyzing efficiency criteria with oncheck -pB output

You can analyze the output of the **oncheck -pB** command to determine if there is a more efficient storage strategy.

Looking at the efficiency information for that is shown for blobspace **bspc1** in Figure 1, a database server administrator might decide that a better storage strategy for TEXT and BYTE data would be to double the blobpage size from 2048 bytes to 4096 bytes. (Blobpage size is always a multiple of the database server page size.) If the database server administrator made this change, the measure of page fullness would remain the same, but the number of locks needed during an update of a simple large object would be reduced by half.

The efficiency information for blobspace **blobPIC** reveals no obvious suggestion for improvement. The two simple large objects in **blobPIC** differ considerably in size, and there is no optimal storage strategy. In general, simple large objects of similar size can be stored more efficiently than simple large objects of different sizes.

Factors that affect I/O for smart large objects

An sbspace is a logical storage unit, composed of one or more chunks, in which you can store smart large objects (such as BLOB, CLOB, or multi representational data). Disk layout for sbspaces, the settings of certain configuration parameters, and some **onspaces** utility options affect I/O for smart large objects.

The DataBlade API and the Informix® ESQL/C application programming interface also provide functions that affect I/O operations for smart large objects.
Important: For most applications, you should use the values that the database server calculates for the disk-storage information.

- Disk layout for sbspaces
  You create sbspaces on separate disks from the table with which the data is associated. You can store smart large objects associated with different tables within the same sbspace. When you store smart large objects in an sbspace on a separate disk from the table with which it is associated, the database server provides some performance advantages.
- Configuration parameters that affect sbspace I/O
  The SBSPACENAME, BUFFERPOOL, and LOGBUFF configuration parameters affect the I/O performance of sbspaces.
- onspaces options that affect sbspace I/O
  When you create an sbspace with the **onspaces** utility, you specify information that affects I/O performance. This information includes the size of extents, the buffering mode (and whether you want the server to use lightweight I/O), and logging.

Related information:
Sbspaces
What is Informix ESQL/C?
DataBlade API overview

Disk layout for sbspaces

You create sbspaces on separate disks from the table with which the data is associated. You can store smart large objects associated with different tables within the same sbspace. When you store smart large objects in an sbspace on a separate disk from the table with which it is associated, the database server provides some performance advantages.

These performance advantages are:

- You have parallel access to the table and smart large objects.
- When you choose not to log the data in an sbspace, you reduce logging I/O activity for logged databases.

To create an sbspace, use the **onspaces** utility. You assign smart large objects to an sbspace when you use the CREATE TABLE statement to create the tables with which the smart large objects are associated.

Related information:
onspaces -c -S: Create an sbspace
CREATE TABLE statement

Configuration parameters that affect sbspace I/O

The SBSPACENAME, BUFFERPOOL, and LOGBUFF configuration parameters affect the I/O performance of sbspaces.

The SBSPACENAME configuration parameter indicates the default sbspace name if you do not specify the sbspace name when you define a column of data type CLOB or BLOB. To reduce disk contention and provide better load balancing, place the default sbspace on a separate disk from the table data.

The BUFFERPOOL configuration parameter specifies the default values for buffers and LRU queues in a buffer pool for both the default page size buffer pool and for any non-default pages size buffer pools. The size of your memory buffer pool affects I/O operations for smart large objects because the buffer pool is the default area of shared memory for these objects. If your applications frequently access smart large objects, it is advantageous to have these objects in the buffer pool. Smart large

objects only use the default page size buffer pool. For information about estimating the amount to increase your buffer pool for smart large objects, see The BUFFERPOOL configuration parameter and memory utilization.

By default, the database server reads smart large objects into the buffers in the resident portion of shared memory. For more information on using lightweight I/O buffers, see Lightweight I/O for smart large objects.

The LOGBUFF configuration parameter affects logging I/O activity because it specifies the size of the logical-log buffers that are in shared memory. The size of these buffers determines how quickly they fill and therefore how often they need to be flushed to disk.

If you log smart-large-object user data, increase the size of your logical-log buffer to prevent frequent flushing to these log files on disk.

Related information:
SBSPACENAME configuration parameter
BUFFERPOOL configuration parameter
LOGBUFF configuration parameter

onspaces options that affect sbspace I/O

When you create an sbspace with the **onspaces** utility, you specify information that affects I/O performance. This information includes the size of extents, the buffering mode (and whether you want the server to use lightweight I/O), and logging.

- Sbspace extents
  As you add smart large objects to a table, the database server allocates disk space to the sbspace in units called *extents*. Each extent is a block of physically contiguous pages from the sbspace.
- Lightweight I/O for smart large objects
  Instead of using the buffer pool, the administrator and programmer have the option to use *lightweight I/O*. Lightweight I/O operations use private buffers in the session pool of the virtual portion of shared memory.
- Logging
  If you decide to log all write operations on data stored in sbspaces, logical-log I/O activity and memory utilization increases.

Sbspace extents

As you add smart large objects to a table, the database server allocates disk space to the sbspace in units called *extents*. Each extent is a block of physically contiguous pages from the sbspace.

Even when the sbspace includes more than one chunk, each extent is allocated entirely within a single chunk so that it remains contiguous. Contiguity is important to I/O performance.

When the pages of data are contiguous, disk-arm motion is minimized when the database server reads the rows sequentially. The mechanism of extents is a compromise between the following competing requirements:

- The size of some smart large objects is not known in advance.
- The number of smart large objects in different tables can grow at different times and different rates.
- All the pages of a single smart large object should ideally be adjacent for best performance when you retrieve the entire object.

Because you might not be able to predict the number and size of smart large objects, you cannot specify the extent length of smart large objects. Therefore, the database server adds extents only as they are needed, but all the pages in any one extent are contiguous for better performance. In addition, when the database server creates a new extent that is adjacent to the previous extent, it treats both extents as a single extent.

The number of pages in an sbspace extent is determined by one of the following methods:

- The database server calculates the extent size for a smart large object from a set of heuristics, such as the number of bytes in a write operation. For example, if an operation asks to write 30 kilobytes, the database server tries to allocate an extent the size of 30 kilobytes.
- The final size of the smart large object as indicated by one of the following functions when you open the sbspace in an application program:
  - *For DB-Access:* the DataBlade API **mi_lo_specset_estbytes** function. For more information about the DataBlade API functions to open a smart large object and set the estimated number of bytes, see the *IBM® Informix DataBlade API Programmer's Guide*.
  - *For ESQL/C:* the Informix® ESQL/C **ifx_lo_specset_estbytes** function. For more information about the Informix ESQL/C functions to open a smart large object and set the estimated number of bytes, see the *IBM Informix ESQL/C Programmer's Manual*.

These functions are the best way to set the extent size because they reduce the number of extents in a smart large object. The database server tries to allocate the entire smart large object as one extent (if an extent of that size is available in the chunk).

- The EXTENT_SIZE flag in the **-Df** option of the **onspaces** command when you create or alter the sbspace
  Most administrators do not use the **onspaces** EXTENT_SIZE flag because the database server calculates the extent size from heuristics. However, you might consider using the **onspaces** EXTENT_SIZE flag in the following situations:
  - Many one-page extents are scattered throughout the sbspace.
  - Almost all smart large objects are the same length.
- The EXTENT SIZE keyword of the CREATE TABLE statement when you define the CLOB or BLOB column
  Most administrators do not use the EXTENT SIZE keyword when they create or alter a table because the database server calculates the extent size from heuristics. However, you might consider using this EXTENT SIZE keyword if almost all smart large objects are the same length.

Important: For most applications, you should use the values that the database server calculates for the extent size. Do not use the DataBlade API **mi_lo_specset_extsz** function or the Informix ESQL/C **ifx_lo_specset_extsz** function to set the extent size of the smart large object.

If you know the size of the smart large object, it is recommended that you specify the size in the **DataBlade API mi_lo_specset_estbytes()** function or Informix ESQL/C **ifx_lo_specset_estbytes()** function instead of in the **onspaces** utility or the CREATE TABLE or the ALTER TABLE statement. These functions are the best way to set the extent size because the database server allocates the entire smart large object as one extent (if it has contiguous storage in the chunk).

Extent sizes over one megabyte do not provide much I/O benefit because the database server performs read and write operations in multiples of 60 kilobytes at the most. However, the database server registers each extent for a smart large object in the metadata area; therefore, especially large smart large objects might have many extent entries. Performance of the database server might degrade when it accesses these extent entries. In this case, you can reduce the number of extent entries in the metadata area if you specify the eventual size of the smart large object in the **mi_lo_specset_estbytes()** function or **ifx_lo_specset_estbytes()** function.

For more information, see Improving metadata I/O for smart large objects.

Lightweight I/O for smart large objects

Instead of using the buffer pool, the administrator and programmer have the option to use *lightweight I/O*. Lightweight I/O operations use private buffers in the session pool of the virtual portion of shared memory.

By default, smart large objects pass through the buffer pool in the resident portion of shared memory. Although smart large objects have lower priority than other data, the buffer pool can become full when an application accesses many smart large objects. A single application can fill the buffer pool with smart large objects and leave little room for data that other applications might need. In addition, when the database server performs scans of many pages into the buffer pool, the overhead and contention associated with checking individual pages in and out might become a bottleneck.

Important: Use private buffers only when you read or write smart large objects in read or write operations greater than 8080 bytes and you seldom access them. That is, if you have infrequent read or write function calls that read large amounts of data in a single function invocation, lightweight I/O can improve I/O performance.

- Advantages of lightweight I/O for smart large objects
  Lightweight I/O provides some performance advantages, because the database server is not using the buffer pool.
- Specifying lightweight I/O for smart large objects
  To specify the use of lightweight I/O when creating the sbspace, use the BUFFERING tag of the **-Df** option in the **onspaces -c -S** command.

Related concepts:
The BUFFERPOOL configuration parameter and memory utilization

Advantages of lightweight I/O for smart large objects

Lightweight I/O provides some performance advantages, because the database server is not using the buffer pool.

Lightweight I/O provides the following advantages:

- Transfers larger blocks of data in one I/O operation
  These I/O blocks can be as large as 60 kilobytes. But the bytes must be adjacent for the database server to transfer them in a single I/O operation.
- Bypasses the overhead of the buffer pool when many pages are read
- Prevents frequently accessed pages from being forced out of the buffer pool when many sequential pages are read for smart large objects

When you use lightweight I/O buffers for smart large objects, the database server might read several pages with one I/O operation. A single I/O operation reads in several smart-large-object pages, up to the size of an extent. For information about when to specify extent size, see Sbspace extents.

Specifying lightweight I/O for smart large objects

To specify the use of lightweight I/O when creating the sbspace, use the BUFFERING tag of the **-Df** option in the **onspaces -c -S** command.

The default value for BUFFERING is ON, which means to use the buffer pool. The buffering mode that you specify (or the default, if you do not specify) in the **onspaces** command is the default buffering mode for all smart large objects stored within the sbspace.
Important: In general, if read and write operations to the smart large objects are less than 8080 bytes, do not specify a buffering mode when you create the sbspace. If you are reading or writing short blocks of data, such as 2 kilobytes or 4 kilobytes, leave the default of "buffering=ON" to obtain better performance.
Programmers can override the default buffering mode when they create, open, or alter a smart-large-object instance with DataBlade API and the Informix® ESQL/C functions. The DataBlade API and the Informix ESQL/C application programming interface provide the LO_NOBUFFER flag to allow lightweight I/O for smart large objects.
Important: Use the LO_NOBUFFER flag only when you read or write smart large objects in operations greater than 8080 bytes and you seldom access them. That is, if you have infrequent read or write function calls that read large amounts of data in a single function invocation, lightweight I/O can improve I/O performance.
Related information:
onspaces -c -S: Create an sbspace
What is Informix ESQL/C?
DataBlade API overview

Logging

If you decide to log all write operations on data stored in sbspaces, logical-log I/O activity and memory utilization increases.

For more information, see Configuration parameters that affect sbspace I/O.

Table I/O

One of the most frequent functions that the database server performs is to bring data and index pages from disk into memory. Pages can be read individually for brief transactions and sequentially for some queries. You can configure the number of pages that the database server brings into memory, and you can configure the timing of I/O requests for sequential scans.

You can also indicate how the database server is to respond when a query requests data from a dbspace that is temporarily unavailable.

The following sections describe these methods of reading pages.

For information about I/O for smart large objects, see Factors that affect I/O for smart large objects.

- Sequential scans
  When the database server performs a sequential scan of data or index pages, most of the I/O wait time is caused by seeking the appropriate starting page. To dramatically improve performance for sequential scans, you can bring in a number of contiguous pages with each I/O operation.
- Light scans
  Some sequential scans of tables can use *light scans* to read the data. A light scan bypasses the buffer pool by utilizing session memory to read directly from disk.
- Unavailable data
  Another aspect of table I/O pertains to situations in which a query requests access to a table or fragment in a dbspace that is temporarily unavailable. When the database server determines that a dbspace is unavailable as the result of a disk failure, queries directed to that dbspace fail by default. The database server allows you to specify dbspaces that, when unavailable, can be skipped by queries,

Sequential scans

When the database server performs a sequential scan of data or index pages, most of the I/O wait time is caused by seeking the appropriate starting page. To dramatically improve performance for sequential scans, you can bring in a number of contiguous pages with each I/O operation.

The action of bringing additional pages along with the first page in a sequential scan is called *read ahead*.

The timing of I/O operations that are needed for a sequential scan is also important. If the scan thread must wait for the next set of pages to be brought in after working its way through each batch, a delay occurs. Timing second and subsequent read requests to bring in pages before they are needed provides the greatest efficiency for sequential scans. The number of pages to bring in and the frequency of read-ahead I/O requests depends on the availability of space in the memory buffers. Read-ahead operations can increase page cleaning to unacceptable levels if too many pages are brought in with each batch or if batches are brought in too often.

Related information:
Read-ahead operations

Light scans

Some sequential scans of tables can use *light scans* to read the data. A light scan bypasses the buffer pool by utilizing session memory to read directly from disk.

Light scans can provide performance advantages over use of the buffer pool for sequential scans and skip scans of large tables. These advantages include:

- Bypassing the overhead of the buffer pool when many data pages are read
- Preventing frequently accessed pages from being forced out of the buffer pool when many sequential pages are read for a single query.

Light scans occur under these conditions:

- The optimizer chooses a sequential scan or a skip-scan of the table.
- The amount of data in the table exceeds one MB.
- The query meets one of the following locking conditions:
  - The isolation level is Dirty Read (or the database has no transaction logging).
  - The table has at least a shared lock on the entire table and the isolation level is *not* Cursor Stability.
    Note: A sequential scan in Repeatable Read isolation automatically acquires a share lock on the table.

Tables that cannot be accessed by light scans

Light scans are only performed on user tables whose data rows are stored in tblspaces. Light scans are not used to access indexes, or to access data stored in blobspaces, smart blob spaces, or partition blobs. Similarly, light scans are not used to access data in the system catalog tables, nor in the tables and pseudotables of system

databases like **sysadmin**, **sysmaster**, **sysuser**, and **sysutils**.

Configuration settings that affect light scans

If the BATCHEDREAD_TABLE configuration parameter or the IFX_BATCHEDREAD_TABLE session environment option to the SET ENVIRONMENT statement is set to `0`, light scans are not used to access tables that have variable length rows, or tables where the row length is greater than the pagesize of the dbspace in which the table is contained. A *variable length* row includes tables that have a variable length column, such as VARCHAR, LVARCHAR or NVARCHAR, as well as tables that are compressed.

You can use the IFX_BATCHEDREAD_TABLE session environment option of the SET ENVIRONMENT statement, or the **onmode -wm** command, to override the setting of the BATCHEDREAD_TABLE configuration parameter for the current session. You can use the **onmode -wf** command to change the value of BATCHEDREAD_TABLE in the ONCONFIG file.

Example of onstat output during a light scan

If you have a long-running scan, you can view output from the **onstat -g scn** command to check the progress of the scan, to determine how long the scan will take before it completes, and to see whether the scan is a light scan or a bufferpool scan.

The following example shows some of the output from **onstat -g scn** for a light scan. The word `Light` in the `Scan Type` field identifies the scan as a light scan.

```
SesID  Thread Partnum  Rowid  Rows Scan'd  Scan Type  Lock Mode  Notes
17     48     300002   207    15           Light                 Forward row lookup
```

Related information:
BATCHEDREAD_TABLE configuration parameter
onstat -g scn command: Print scan information

Unavailable data

Another aspect of table I/O pertains to situations in which a query requests access to a table or fragment in a dbspace that is temporarily unavailable. When the database server determines that a dbspace is unavailable as the result of a disk failure, queries directed to that dbspace fail by default. The database server allows you to specify dbspaces that, when unavailable, can be skipped by queries,

For information about specifying dbspaces that, when unavailable, can be skipped by queries, see How DATASKIP affects table I/O.
Warning: If a dbspace containing data that a query requests is listed in the DATASKIP configuration parameter and is currently unavailable because of a disk failure, the data that the database server returns to the query can be inconsistent with the actual contents of the database.

Configuration parameters that affect table I/O

The AUTO_READAHEAD configuration parameter changes the automatic read-ahead mode or disables automatic read-ahead for a query. In addition, the DATASKIP configuration parameter enables or disables data skipping.

Automatic read-ahead processing helps improve query performance by issuing asynchronous page requests when Informix® detects that the query is encountering I/O. Asynchronous page requests can improve query performance by overlapping query processing with the processing necessary to retrieve data from disk and put it in the buffer pool. You can also use the AUTO_READAHEAD environment option of the SET ENVIRONMENT statement of SQL to enable or disable the value of the AUTO_READAHEAD configuration parameter for a session.

- How DATASKIP affects table I/O
  The DATASKIP configuration parameter allows you to specify which dbspaces, if any, queries can skip when those dbspaces are unavailable as the result of a disk failure. You can list specific dbspaces and turn data skipping on or off for all dbspaces.

Related information:
AUTO_READAHEAD configuration parameter

How DATASKIP affects table I/O

The DATASKIP configuration parameter allows you to specify which dbspaces, if any, queries can skip when those dbspaces are unavailable as the result of a disk failure. You can list specific dbspaces and turn data skipping on or off for all dbspaces.

When data skipping is enabled, the database server sets the sixth character in the SQLWARN array to `W`..
Warning: The database server cannot determine whether the results of a query are consistent when a dbspace is skipped. If the dbspace contains a table fragment, the user who executes the query must ensure that the rows within that fragment are not needed for an accurate query result. Turning DATASKIP on allows queries with incomplete data to return results that can be inconsistent with the actual state of the database. Without proper care, that data can yield incorrect or misleading query results.
Related information:
DATASKIP Configuration Parameter
SQLWARN array

Background I/O activities

Background I/O activities do not service SQL requests directly. Many of these activities are essential to maintain database consistency and other aspects of database server operation. However, they create overhead in the CPU and take up I/O bandwidth.

These overhead activities take time away from queries and transactions. If you do not configure background I/O activities properly, too much overhead for these activities can limit the transaction throughput of your application.

The following list shows some background I/O activities:

- Checkpoints
- Logging
- Page cleaning
- Backup and restore
- Rollback and recovery
- Data replication
- Auditing

Checkpoints occur regardless of whether much database activity occurs; however, they can occur with greater frequency as activity increases. Other background activities, such as logging and page cleaning, occur more frequently as database use increases. Activities such as backups, restores, or fast recoveries occur only as scheduled or under exceptional circumstances.

For the most part, tuning your background I/O activities involves striking a balance between appropriate checkpoint intervals, logging modes and log sizes, and page-cleaning thresholds. The thresholds and intervals that trigger background I/O activity often interact; adjustments to one threshold might shift the performance bottleneck to another.

The following sections describe the performance effects and considerations that are associated with the configuration parameters that affect these background I/O activities.

- Configuration parameters that affect checkpoints
  The RTO_SERVER_RESTART, CKPTINTVL, LOGSIZE, LOGFILES, PHYSFILE, and ONDBSPACEDOWN configuration parameters affect checkpoints.
- Configuration parameters that affect logging
  The LOGBUFF, PHYSBUFF, LOGFILES, LOGSIZE, DYNAMIC_LOGS, AUTO_LLOG, LTXHWM, LTXEHWM, SESSION_LIMIT_LOGSPACE, SESSION_LIMIT_TXN_TIME, and TEMPTAB_NOLOG configuration parameters affect logging.
- Configuration parameters that affect page cleaning
  Several configuration parameters, including the CLEANERS and RTO_SERVER_RESTART configuration parameters, affect page cleaning. If pages are not cleaned often enough, an **sqlexec** thread that performs a query might be unable to find the available pages that it needs.
- Configuration parameters that affect backup and restore
  Four configuration parameters that affect backup and restore on all operating systems also affect background I/O. Additional configuration parameters affect backup and restore on UNIX.
- Configuration parameters that affect rollback and recovery
  The OFF_RECVRY_THREADS, ON_RECVRY_THREADS, PLOG_OVERFLOW_PATH, and RTO_SERVER_RESTART configuration parameters affect recovery. The LOW_MEMORY_RESERVE configuration parameter reserves a specific amount of memory, in kilobytes, for the database server to use when critical activities, such as rollback activities, are needed.
- Configuration parameters that affect data replication and auditing
  Data replication and auditing are optional. If you use these features, you can set configuration parameters that affect data-replication performance and auditing performance.
- LRU tuning
  The LRU settings for flushing each buffer pool between checkpoints are not critical to checkpoint performance. The LRU settings are necessary only for maintaining enough clean pages for page replacement.

Configuration parameters that affect checkpoints

The RTO_SERVER_RESTART, CKPTINTVL, LOGSIZE, LOGFILES, PHYSFILE, and ONDBSPACEDOWN configuration parameters affect checkpoints.

- RTO_SERVER_RESTART and its effect on checkpoints
  The RTO_SERVER_RESTART configuration parameter specifies the amount of time, in seconds, that Informix® has to recover from an unplanned outage.
- CKPTINTVL and its effect on checkpoints
  If the RTO_SERVER_RESTART configuration parameter is not on, the CKPTINTVL configuration parameter specifies the frequency, in seconds, at which the database server checks to determine whether a checkpoint is needed.
- LOGSIZE and LOGFILES and their effect on checkpoints
  The LOGSIZE and LOGFILES configuration parameters indirectly affect checkpoints because they specify the size and number of logical-log files. A checkpoint can occur when the database server detects that the next logical-log file to become current contains the most-recent checkpoint record.
- Checkpoints and the physical log
  The PHYSFILE configuration parameter specifies the size of the initial physical log. A checkpoint occurs when either the physical log becomes 75 percent full or a high number of dirty partitions exist.
- ONDBSPACEDOWN and its effect on checkpoints
  The ONDBSPACEDOWN configuration parameter specifies the response that the database server makes when an I/O error indicates that a dbspace is down. By default, the database server identifies any dbspace that contains no critical data as `down` and continues processing. Critical data includes the root dbspace, the logical log, or the physical log.

RTO_SERVER_RESTART and its effect on checkpoints

The RTO_SERVER_RESTART configuration parameter specifies the amount of time, in seconds, that Informix® has to recover from an unplanned outage.

The performance advantage of enabling this configuration parameter is:

- Enabling fast recovery to meet the RTO_SERVER_RESTART policy by seeding the buffer pool with the data pages required by log replay.

The performance disadvantages of enabling this configuration parameter are:

- Increased physical log activity which might slightly impact transaction performance
- Increased checkpoint frequency, because the physical log space is depleted more quickly (You can increase the size of the physical log to avoid the increase in checkpoint frequency.)

When RTO_SERVER_RESTART is enabled, the database server:

- Attempts to make sure nonblocking checkpoints do not run out of critical resources during checkpoint processing by triggering more frequent checkpoints if transactions might run out of physical or logical log resources, which would cause transaction blocking.
- Ignores the CKPTINTVL configuration parameter.
- Automatically controls checkpoint frequency to meet the RTO policy and to prevent the server from running out of log resources.
- Automatically adjusts the number of AIO virtual processors and cleaner threads and automatically tunes LRU flushing.

The database server prints warning messages in the message log if the server cannot meet the RTO_SERVER_RESTART policy.

- Automatic checkpoints, LRU tuning, and AIO virtual processor tuning
  The database server automatically adjusts checkpoint frequency to avoid transaction blocking. The server monitors physical and logical log consumption along with information about past checkpoint performance. Then, if necessary, the server triggers checkpoints more frequently to avoid transaction blocking.

Related information:
RTO_SERVER_RESTART configuration parameter

Automatic checkpoints, LRU tuning, and AIO virtual processor tuning

The database server automatically adjusts checkpoint frequency to avoid transaction blocking. The server monitors physical and logical log consumption along with information about past checkpoint performance. Then, if necessary, the server triggers checkpoints more frequently to avoid transaction blocking.

You can turn off automatic checkpoint tuning by setting **onmode -wf AUTO_CKPTS** to 0, or setting the AUTO_CKPTS configuration parameter to 0.

Because the database server does not block transactions during checkpoint processing, LRU flushing is relaxed. If the server is not able to complete checkpoint processing before the physical log is full (which causes transaction blocking), and if you cannot increase the size of the physical log, you can configure the server for more aggressive LRU flushing. The increase in LRU flushing impacts transaction performance, but reduces transaction blocking. If you do not configure the server for more aggressive flushing, the server automatically adjusts LRU flushing to be more aggressive only when the server is unable to find a low priority buffer for page replacement.

If the VPCLASS configuration parameter setting for AIO virtual processors is set to **autotune=1**, the database server automatically increases the number of AIO virtual processors and page-cleaner threads when the server detects that AIO virtual processors are not keeping up with the I/O workload.

Automatic LRU tuning affects all buffer pools and adjusts **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

Related concepts:
LRU tuning
Related information:
AUTO_CKPTS configuration parameter
BUFFERPOOL configuration parameter
VPCLASS configuration parameter

CKPTINTVL and its effect on checkpoints

If the RTO_SERVER_RESTART configuration parameter is not on, the CKPTINTVL configuration parameter specifies the frequency, in seconds, at which the database server checks to determine whether a checkpoint is needed.

When the RTO_SERVER_RESTART configuration parameter is on, the database server ignores the CKPTINTVL configuration parameter. Instead, the server automatically triggers checkpoints in order to maintain the RTO_SERVER_RESTART policy.

The database server can skip a checkpoint if all data is physically consistent when the checkpoint interval expires.

Checkpoints also occur in either of these circumstances:

- Whenever the physical log becomes 75 percent full
- If a high number of dirty partitions exist, even if the physical log is not 75 percent full.
  This occurs because when the database server checks if the physical log is 75 percent full, the server also checks if the following condition is true:

```
(Physical Log Pages Used + Number of Dirty Partitions) >=
(Physical Log Size * 9) /10)
```

A partition, which represents one page going into the physical log during checkpoint processing and has a page that maintains information (such as the number of rows and number of data pages) about the partition, becomes dirty when the partition is updated.

If you set CKPTINTVL to a long interval, you can use physical-log capacity to trigger checkpoints based on actual database activity instead of an arbitrary time unit. However, a long checkpoint interval can increase the time needed for recovery in the event of a failure. Depending on your throughput and data-availability requirements, you can choose an initial checkpoint interval of 5, 10, or 15 minutes, with the understanding that checkpoints might occur more often, depending on physical-logging activity.

The database server writes a message to the message log to note the time that it completes a checkpoint. To read these messages, use **onstat -m**.

Related information:
CKPTINTVL configuration parameter

LOGSIZE and LOGFILES and their effect on checkpoints

The LOGSIZE and LOGFILES configuration parameters indirectly affect checkpoints because they specify the size and number of logical-log files. A checkpoint can occur when the database server detects that the next logical-log file to become current contains the most-recent checkpoint record.

If you need to free the logical-log file that contains the last checkpoint, the database server must write a new checkpoint record to the current logical-log file. If the frequency with which logical-log files are backed up and freed increases, the frequency at which checkpoints occur increases. Although checkpoints block user processing, they no longer last as long. Because other factors (such as the physical-log size) also determine the checkpoint frequency, this effect might not be significant.

When the dynamic log allocation feature is enabled, the size of the logical log does not affect the thresholds for long transactions as much as it did in previous versions of the database server. For details, see LTXHWM and LTXEHWM and their effect on logging.

The LOGSIZE, LOGFILES, and LOGBUFF configuration parameters also affect logging I/O activity and logical backups. For more information, see Configuration parameters that affect logging.

Related information:
LOGFILES configuration parameter
LOGSIZE configuration parameter
Estimate the number of logical-log files

Checkpoints and the physical log

The PHYSFILE configuration parameter specifies the size of the initial physical log. A checkpoint occurs when either the physical log becomes 75 percent full or a high number of dirty partitions exist.

The rate at which transactions generate physical log activity can affect checkpoint performance. To avoid transaction blocking during checkpoint processing, consider the size of the physical log and how quickly it fills.

You can enable the database server to expand the size of the physical log as needed to improve performance by creating an extendable plogspace for the physical log.

For example, operations that do not perform updates do not generate before-images. If the size of the database is growing, but applications rarely update the data, little physical logging occurs. In this situation, you might not need a large physical log.

Similarly, you can define a smaller physical log if your application updates the same pages. The database server writes the before-image of only the first update that is made to a page for the following operations:

- Inserts, updates, and deletes for rows that contain user-defined data types (UDTs), smart large objects, and simple large objects
- ALTER statements
- Operations that create or modify indexes (B-tree, R-tree, or user-defined indexes)

Because the physical log is recycled after each checkpoint, the physical log must be large enough to hold before-images from changes between checkpoints. If the database server frequently triggers checkpoints because it runs out of physical log space, consider increasing the size of the physical log.

If you increase the checkpoint interval or if you anticipate increased update activity, you might want to increase the size of the physical log.

The physical log is an important part of maintaining RTO_SERVER_RESTART policy. To ensure that you have an abundance of space, set the size of the physical log to at least 110 percent of the size of all buffer pools.

You can use the **onparams** utility to change the physical log location and size. You can change the physical log while transactions are active and without restarting the database server.

Related reference:
Configuration parameters that affect critical data
Related information:
PHYSFILE configuration parameter
Strategy for estimating the size of the physical log
Change the physical-log location and size
Plogspace

ONDBSPACEDOWN and its effect on checkpoints

The ONDBSPACEDOWN configuration parameter specifies the response that the database server makes when an I/O error indicates that a dbspace is down. By default, the database server identifies any dbspace that contains no critical data as `down` and continues processing. Critical data includes the root dbspace, the logical log, or the physical log.

To restore access to that database, you must back up all logical logs and then perform a warm restore on the down dbspace.

The database server halts operation whenever a disabling I/O error occurs on a nonmirrored dbspace that contains critical data, regardless of the setting for ONDBSPACEDOWN. In such an event, you must perform a cold restore of the database server to resume normal database operations.

The value of ONDBSPACEDOWN has no affect on temporary dbspaces. For temporary dbspaces, the database server continues processing regardless of the ONDBSPACEDOWN setting. If a temporary dbspace requires fixing, you can drop and recreate it.

When ONDBSPACEDOWN is set to `2`, the database server continues processing to the next checkpoint and then suspends processing of all update requests. The database server repeatedly retries the I/O request that produced the error until the dbspace is repaired and the request completes or the database server administrator intervenes. The administrator can use **onmode -O** to mark the dbspace `down` and continue processing while the dbspace remains unavailable or use **onmode -k** to halt the database server.
Important: This *2* setting for ONDBSPACEDOWN can affect the performance for update requests severely because they are suspended due to a down dbspace. When you use this setting for ONDBSPACEDOWN, be sure to monitor the status of the dbspaces.
When you set ONDBSPACEDOWN to `1`, the database server treats all dbspaces as though they were critical. Any nonmirrored dbspace that becomes disabled halts normal processing and requires a cold restore. The performance impact of halting and performing a cold restore when any dbspace goes down can be severe.
Important: If you decide to set ONDBSPACEDOWN to *1*, consider mirroring all your dbspaces.
Related information:
ONDBSPACEDOWN configuration parameter

Configuration parameters that affect logging

The LOGBUFF, PHYSBUFF, LOGFILES, LOGSIZE, DYNAMIC_LOGS, AUTO_LLOG, LTXHWM, LTXEHWM, SESSION_LIMIT_LOGSPACE, SESSION_LIMIT_TXN_TIME, and TEMPTAB_NOLOG configuration parameters affect logging.

Logging, checkpoints, and page cleaning are necessary to maintain database consistency. A direct trade-off exists between the frequency of checkpoints or the size of the logical logs and the time that it takes to recover the database in the event of a failure. Therefore, a major consideration when you attempt to reduce the overhead for these activities is the delay that you can accept during recovery.

- LOGBUFF and PHYSBUFF and their effect on logging
  The LOGBUFF and PHYSBUFF configuration parameters affect logging I/O activity because they specify the respective sizes of the logical-log and physical-log buffers that are in shared memory. The size of these buffers determines how quickly the buffers fill and therefore how often they need to be flushed to disk.
- LOGFILES and its effect on logging
  The LOGFILES configuration parameter, which specifies the number of logical-log files, affects logging.
- LOGSIZE and its effect on logging
  The LOGSIZE configuration parameter specifies the size of each logical log file. It is difficult to predict how much logical-log space your database server system requires until the system is fully in use.
- DYNAMIC_LOGS and its effect on logging
  The dynamic log file allocation feature prevents hanging problems that are caused by rollbacks of a long transaction because the database server does not run out of log space. The DYNAMIC_LOGS configuration parameter specifies whether the dynamic log file allocation feature is off, on, or causes the server to pause to allow the manual addition of a logical log file.
- AUTO_LLOG and its effect on logging
  Insufficient logical logs can affect performance by triggering frequent checkpoints, blocking checkpoints, or long checkpoints. The AUTO_LLOG configuration parameter controls whether the database server automatically adds logical logs to improve performance.
- LTXHWM and LTXEHWM and their effect on logging
  The LTXHWM and LTXEHWM configuration parameters define long transaction watermarks.
- TEMPTAB_NOLOG and its effect on logging
  The TEMPTAB_NOLOG configuration parameter allows you to disable logging on temporary tables. You can do this to improve performance and to prevent Informix® from transferring temporary tables when using High-Availability Data Replication (HDR).
- SESSION_LIMIT_LOGSPACE and its effect on logging
  The SESSION_LIMIT_LOGSPACE configuration parameter specifies the maximum amount of log space that a session can use for individual transactions, and can prevent individual sessions from monopolizing the logical log.
- SESSION_LIMIT_TXN_TIME and its effect on logging
  The SESSION_LIMIT_TXN_TIME configuration parameter limits how much time a transaction can run in a session, and can prevent individual session transactions from monopolizing the logical log.

LOGBUFF and PHYSBUFF and their effect on logging

The LOGBUFF and PHYSBUFF configuration parameters affect logging I/O activity because they specify the respective sizes of the logical-log and physical-log buffers that are in shared memory. The size of these buffers determines how quickly the buffers fill and therefore how often they need to be flushed to disk.

Related information:
LOGBUFF configuration parameter
PHYSBUFF configuration parameter

LOGFILES and its effect on logging

The LOGFILES configuration parameter, which specifies the number of logical-log files, affects logging.

When you initialize or restart the database server, it creates the number of logical-log files that you specify in the LOGFILES configuration parameter.

You might add logical-log files for the following reasons:

- To increase the disk space allocated to the logical log
- To change the size of your logical-log files
- To enable an open transaction to roll back
- As part of moving logical-log files to a different dbspace

- Calculating the space allocated to logical log files
  If all of your logical log files are the same size, you can calculate the total space allocated to the logical log files.

Related information:
LOGFILES configuration parameter
Estimate the number of logical-log files

Calculating the space allocated to logical log files

If all of your logical log files are the same size, you can calculate the total space allocated to the logical log files.

To calculate the space allocated to these files, use the following formula:

```
total logical log space = LOGFILES * LOGSIZE
```

If you add logical-log files that are not the size specified by the LOGSIZE configuration parameter, you cannot use the `LOGFILES * LOGSIZE` expression to calculate the size of the logical log. Instead, you need to add the sizes for each individual log file on disk.

Use the **onstat -l** utility to monitor logical-log files.

LOGSIZE and its effect on logging

The LOGSIZE configuration parameter specifies the size of each logical log file. It is difficult to predict how much logical-log space your database server system requires until the system is fully in use.

The size of the logical log space (LOGFILES * LOGSIZE) is determined by these policies:

Recovery time objective (RTO)
: This is the length of time you can afford to be without your systems. If your only objective is failure recovery, the total log space only needs to be large enough to contain all the transactions for two checkpoint cycles. When the RTO_SERVER_RESTART configuration parameter is enabled and the server has a combined buffer pool size of less that four gigabytes, you can configure the total log space to 110% of the combined buffer pool sizes. Too much log space does not impact performance; however, too little log space can cause more frequent checkpoints and transaction blocking.

Recovery point objective (RPO)
: This describes the age of the data you want to restore in the event of a disaster. If the objective is to make sure transactional work is protected, the optimum LOGSIZE should be a multiple of how much work gets done per RPO unit. Because the database server supports partial log backup, an optimal log size is not critical and a non-optimal log size simply means more frequent log file changes. RPO is measured in units of time. If the business rule is that the system cannot lose more than ten minutes of transactional data if a complete site disaster occurs, then a log backup should occur every ten minutes.
  You can use the Scheduler, which manages and executes scheduled administrative tasks, to set up automatic log backup.

Long Transactions
: If you have long transactions that require a large amount of log space, you should allocate that space for the logs. Inadequate log space impacts transaction performance.

Choose a log size based on how much logging activity occurs and the amount of risk in case of catastrophic failure. If you cannot afford to lose more than an hour's worth of data, create many small log files that each hold an hour's worth of transactions. Turn on continuous-log backup. Small logical-log files fill sooner, which means more frequent logical-log backups.

If your system is stable with high logging activity, choose larger logs to improve performance. Continuous-log backups occur less frequently with large log files. Also consider the maximum transaction rates and speed of the backup devices. Do not let the whole logical log fill. Turn on continuous-log backup and leave enough room in the logical logs to handle the longest transactions.

The backup process can hinder transaction processing that involves data located on the same disk as the logical-log files. If enough logical-log disk space is available, however, you can wait for periods of low user activity before you back up the logical-log files.

- Estimating logical-log size when logging dbspaces
  To estimate the size of logical logs, use a formula or **onstat -u** information.

- Estimating the logical-log size when logging simple large objects
  To obtain better overall performance for applications that perform frequent updates of TEXT or BYTE data in blobspaces, reduce the size of the logical log.
- Estimating the logical-log size when logging smart large objects
  If you plan to log smart-large-object user data, you must ensure that the log size is considerably larger than the amount of data being written. Smart-large-object metadata is always logged even if the smart large objects are not logged.

Related information:
LOGSIZE configuration parameter
The Scheduler

Estimating logical-log size when logging dbspaces

To estimate the size of logical logs, use a formula or **onstat -u** information.

Use the following formula to obtain an initial estimate for LOGSIZE in kilobytes:

```
LOGSIZE = (connections * maxrows * rowsize) / 1024) / LOGFILES
```

In this formula:

- *connections* is the maximum number of connections for all network types specified in the sqlhosts information by one or more NETTYPE parameters. If you configured more than one connection by setting multiple NETTYPE configuration parameters in your configuration file, sum the **users** fields for each NETTYPE parameter, and substitute this total for *connections* in the preceding formula.
- *maxrows* is the largest number of rows to be updated in a single transaction.
- *rowsize* is the average size of a row in bytes. You can calculate *rowsize* by adding up the length (from the **syscolumns** system catalog table) of the columns in a row.
- *1024* is a necessary divisor because you specify LOGSIZE in kilobytes.

To obtain a better estimate during peak activity periods, execute the **onstat -u** command. The last line of the **onstat -u** output contains the maximum number of concurrent connections.

You need to adjust the size of the logical log when your transactions include simple large objects or smart large objects, as the following sections describe.

You also can increase the amount of space devoted to the logical log by adding another logical-log file.

Related information:
Adding logical-log files manually

Estimating the logical-log size when logging simple large objects

To obtain better overall performance for applications that perform frequent updates of TEXT or BYTE data in blobspaces, reduce the size of the logical log.

Blobpages cannot be reused until the logical log to which they are allocated is backed up. When TEXT or BYTE data activity is high, the performance impact of more frequent checkpoints is balanced by the higher availability of free blobpages.

When you use volatile blobpages in blobspaces, smaller logs can improve access to simple large objects that must be reused. Simple large objects cannot be reused until the log in which they are allocated is flushed to disk. In this case, you can justify the cost in performance because those smaller log files are backed up more frequently.

Estimating the logical-log size when logging smart large objects

If you plan to log smart-large-object user data, you must ensure that the log size is considerably larger than the amount of data being written. Smart-large-object metadata is always logged even if the smart large objects are not logged.

Use the following guidelines when you log smart large objects:

- If you are appending data to a smart large object, the increased logging activity is roughly equal to the amount of data written to the smart large object.
- If you are updating a smart large object (overwriting data), the increased logging activity is roughly twice the amount of data written to the smart large object. The database server logs both the before-image and after-image of a smart large object for update transactions. When updating the smart large objects, the database server logs only the updated parts of the before and after image.
- Metadata updates affect logging less. Even though metadata is always logged, the number of bytes logged is usually much smaller than the smart large objects.

DYNAMIC_LOGS and its effect on logging

The dynamic log file allocation feature prevents hanging problems that are caused by rollbacks of a long transaction because the database server does not run out of log space. The DYNAMIC_LOGS configuration parameter specifies whether the dynamic log file allocation feature is off, on, or causes the server to pause to allow the manual

addition of a logical log file.

Dynamic log allocation allows you to do the following actions:

- Add a logical log file while the system is active, even during fast recover.
- Insert a logical log file immediately after the current log file, instead of appending it to the end.
- Immediately access the logical log file even if the root dbspace is not backed up.

The default value for the DYNAMIC_LOGS configuration parameter is `2`, which means that the database server automatically allocates a new logical log file after the current log file when it detects that the next log file contains an open transaction. The database server automatically checks if the log after the current log still contains an open transaction at the following times:

- Immediately after it switches to a new log file while writing log records (not while reading and applying log records)
- At the beginning of the transaction cleanup phase which occurs as the last phase of logical recovery
  Logical recovery happens at the end of fast recovery and at the end of a cold restore or roll forward.
- During transaction cleanup (rollback of open transactions), a switch to a new log file log might occur
  The database server also checks after this switch because it is writing log records for the rollback.

When you use the default value of `2` for DYNAMIC_LOGS, the database server determines the location and size of the new logical log for you:

- The database server uses the following criteria to determine on which disk to allocate the new log file:
  - Favor mirrored dbspaces
  - Avoid root dbspace until no other critical dbspace is available
  - Least favored space is unmirrored and noncritical dbspaces
- The database server uses the average size of the largest log file and the smallest log file for the size of the new logical log file. If not enough contiguous disk space is available for this average size, the database server searches for space for the next smallest average size. The database server allocates a minimum of 200 kilobytes for the new log file.

If you want to control the location and size of the additional log file, set DYNAMIC_LOGS to `1`. When the database server switches log files, it still checks if the next active log contains an open transaction. If it does find an open transaction in the next log to be active, it does the following actions:

- Issues alarm event 27 (log required)
- Writes a warning message to the online log
- Pauses to wait for the administrator to manually add a log with the **onparams -a -i** command-line option

You can write a script that will execute when alarm event 27 occurs to execute **onparams -a -i** with the location you want to use for the new log. Your script can also execute the **onstat -d** command to check for adequate space and execute the **onparams -a -i** command with the location that has enough space. You must use the **-i** option to add the new log right after the current log file.

If you set DYNAMIC_LOGS to 0, the database server still checks whether the next active log contains an open transaction when it switches log files. If it does find an open transaction in the next log to be active, it issues the following warning:

```
WARNING: The oldest logical log file (%d) contains records
from an open transaction (0x%p), but the Dynamic Log
Files feature is turned off.
```

Related information:
DYNAMIC_LOGS configuration parameter
Fast recovery

AUTO_LLOG and its effect on logging

Insufficient logical logs can affect performance by triggering frequent checkpoints, blocking checkpoints, or long checkpoints. The AUTO_LLOG configuration parameter controls whether the database server automatically adds logical logs to improve performance.

If you created a server during installation, the AUTO_LLOG configuration parameter is enabled automatically. Otherwise, you can edit the value of the AUTO_LLOG configuration parameter.

If the AUTO_LLOG configuration parameter is enabled, the database server automatically adds logical log files under the following circumstances:

- When a substantial portion of the last 20 checkpoints were caused by logical logs filling up
- When inadequate logical log space causes a blocking checkpoint
- When inadequate logical log space causes a long checkpoint

The AUTO_LLOG configuration parameter also specifies the dbspace for new logical log files and the maximum size of all logical log files before the server stops adding logical logs for performance. The following guidelines show estimates of the maximum amount of space for logical logs that you might need, depending on the number of concurrent users who access your database server:

- 1 - 100 users: 200 MB
- 101 - 500 users: 5 MB
- 501 - 1000 users: 1 GB
- More than 1000 users: 2 GB

The settings of the AUTO_LLOG configuration parameter and the DYNAMIC_LOGS configuration parameters do not interact.

Related information:
AUTO_LLOG configuration parameter

LTXHWM and LTXEHWM and their effect on logging

The LTXHWM and LTXEHWM configuration parameters define long transaction watermarks.

After the release of the dynamic log file feature, long transaction high watermarks are no longer as critical, because the server does not run out of log space unless you use up the physical disk space available on the system. The LTXHWM parameter still indicates how full the logical log is when the database server starts to check for a possible long transaction and to roll it back. LTXEHWM still indicates the point at which the database server suspends new transaction activity to locate and roll back a long transaction. These events are usually rare, but if they occur, they can indicate a serious problem within an application.

Under normal operations, use the default values for LTXHWM and LTXEHWM. However, you might want to change these default values for one of the following reasons:

- To allow other transactions to continue update activity (which requires access to the log) during the rollback of a long transaction
  In this case, you increase the value of LTXEHWM to raise the point at which the long transaction rollback has exclusive access to the log.
- To run scheduled transactions of unknown length, such as large loads that are logged
  In this case, you increase the value of LTXHWM so that the transaction has a chance to complete before it reaches the high watermark.

Related information:
LTXEHWM configuration parameter
LTXHWM configuration parameter

TEMPTAB_NOLOG and its effect on logging

The TEMPTAB_NOLOG configuration parameter allows you to disable logging on temporary tables. You can do this to improve performance and to prevent Informix® from transferring temporary tables when using High-Availability Data Replication (HDR).

To disable logging on temporary tables, set the TEMPTAB_NOLOG configuration parameter to `1`.

To enable logging on temporary tables for primary server and to disable logging on temporary tables for secondary servers(HDR, RSS and SDS), set the TEMPTAB_NOLOG configuration parameter to `2`.

Related information:
TEMPTAB_NOLOG configuration parameter

SESSION_LIMIT_LOGSPACE and its effect on logging

The SESSION_LIMIT_LOGSPACE configuration parameter specifies the maximum amount of log space that a session can use for individual transactions, and can prevent individual sessions from monopolizing the logical log.

SESSION_LIMIT_LOGSPACE does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Related information:
SESSION_LIMIT_LOGSPACE configuration parameter
SESSION_LIMIT_TXN_TIME configuration parameter

SESSION_LIMIT_TXN_TIME and its effect on logging

The SESSION_LIMIT_TXN_TIME configuration parameter limits how much time a transaction can run in a session, and can prevent individual session transactions from monopolizing the logical log.

The database server terminates a transaction that exceeds the SESSION_LIMIT_TXN_TIME limit, and produces an error in the database server message log.

SESSION_LIMIT_TXN_TIME does not apply to a user who holds administrative privileges, such as user **informix** or a DBSA user.

Configuration parameters that affect page cleaning

Several configuration parameters, including the CLEANERS and RTO_SERVER_RESTART configuration parameters, affect page cleaning. If pages are not cleaned often enough, an **sqlexec** thread that performs a query might be unable to find the available pages that it needs.

If the **sqlexec** thread cannot find the available pages that it needs, the thread initiates a *foreground write* and waits for pages to be freed. Foreground writes impair performance, so you should avoid them. To reduce the frequency of foreground writes, increase the number of page cleaners or decrease the threshold for triggering a page cleaning.

Use **onstat -F** to monitor the frequency of foreground writes.

The following configuration parameters affect page cleaning:

- BUFFERPOOL, which contains **lrus**, **lru_max_dirty**, and **lru_min_dirty** values
  Information that was specified with the BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY configuration parameters before Version 10.0 is now specified using the BUFFERPOOL configuration parameter.
- CLEANERS
- RTO_SERVER_RESTART

- CLEANERS and its effect on page cleaning
  The CLEANERS configuration parameter indicates the number of page-cleaner threads to run. For installations that support fewer than 20 disks, one page-cleaner thread is recommended for each disk that contains database server data. For installations that support between 20 and 100 disks, one page-cleaner thread is recommended for every two disks. For larger installations, one page-cleaner thread is recommended for every four disks.
- BUFFERPOOL and its effect on page cleaning
  The BUFFERPOOL configuration parameter specifies the number of least recently used (LRU) queues to set up within the shared-memory buffer pool. The buffer pool is distributed among LRU queues. Configuring more LRU queues allows more page cleaners to operate and reduces the size of each LRU queue.
- RTO_SERVER_RESTART and its effect on page cleaning
  The RTO_SERVER_RESTART configuration parameter allows you to use recovery time objective (RTO) standards to set the amount of time, in seconds, that Informix® has to recover from a problem after you restart Informix and bring it into online or quiescent mode.

CLEANERS and its effect on page cleaning

The CLEANERS configuration parameter indicates the number of page-cleaner threads to run. For installations that support fewer than 20 disks, one page-cleaner thread is recommended for each disk that contains database server data. For installations that support between 20 and 100 disks, one page-cleaner thread is recommended for every two disks. For larger installations, one page-cleaner thread is recommended for every four disks.

If you increase the number of LRU queues, you must increase the number of page-cleaner threads proportionally.

Related information:
CLEANERS configuration parameter

BUFFERPOOL and its effect on page cleaning

The BUFFERPOOL configuration parameter specifies the number of least recently used (LRU) queues to set up within the shared-memory buffer pool. The buffer pool is distributed among LRU queues. Configuring more LRU queues allows more page cleaners to operate and reduces the size of each LRU queue.

For a single-processor system, set the **lrus** field of the BUFFERPOOL configuration parameter to a minimum of 8. For multiprocessor systems, set the **lrus** field to a minimum of 8 or to the number of CPU VPs, whichever is greater.

The **lrus**, **lru_max_dirty**, and **lru_min_dirty** values control how often pages are flushed to disk between checkpoints. Automatic LRU tuning, as set by the AUTO_LRU configuration parameter, affects all buffer pools and adjusts the **lru_min_dirty** and **lru_max_dirty** values in the BUFFERPOOL configuration parameter.

If you increase the **lru_max_dirty** and **lru_min_dirty** values to improve transaction throughput, do not change the gap between the **lru_max_dirty** and **lru_min_dirty**.

When the buffer pool is very large and transaction blocking is occurring during checkpoint processing, look in the message log to determine which resource is triggering transaction blocking. If the physical or logical log is critically low and triggers transaction blocking, increase the size of the resource that is causing the transaction blocking. If you cannot increase the size of the resource, consider making LRU flushing more aggressive by decreasing the **lru_min_dirty** and **lru_max_dirty** settings so that the server has fewer pages to flush to disk during checkpoint processing.

To monitor the percentage of dirty pages in LRU queues, use the **onstat -R** command. When the number of dirty pages consistently exceeds the **lru_max_dirty** limit, you have too few LRU queues or too few page cleaners. First, use the BUFFERPOOL configuration parameter to increase the number of LRU queues. If the percentage of dirty pages still exceeds the **lru_max_dirty** limit, update the CLEANERS configuration parameter to increase the number of page cleaners.

Related concepts:
The BUFFERPOOL configuration parameter and memory utilization
Related information:
BUFFERPOOL configuration parameter
Number of LRU queues to configure

RTO_SERVER_RESTART and its effect on page cleaning

The RTO_SERVER_RESTART configuration parameter allows you to use recovery time objective (RTO) standards to set the amount of time, in seconds, that Informix® has to recover from a problem after you restart Informix and bring it into online or quiescent mode.

When this configuration parameter is enabled, the database server automatically adjusts the number of AIO virtual processors and cleaner threads and automatically tunes LRU flushing.

Use the AUTO_LRU_TUNING configuration parameter to specify whether automatic LRU tuning is enabled or disabled when the server starts.

Related information:
RTO_SERVER_RESTART configuration parameter
AUTO_LRU_TUNING configuration parameter

Configuration parameters that affect backup and restore

Four configuration parameters that affect backup and restore on all operating systems also affect background I/O. Additional configuration parameters affect backup and restore on UNIX.

The following configuration parameters affect backup and restore on all operating systems:

- BAR_MAX_BACKUP
- BAR_NB_XPORT_COUNT
- BAR_PROGRESS_FREQ
- BAR_XFER_BUF_SIZE

In addition, the following configuration parameters affect backup and restore on UNIX:

- LTAPEBLK
- LTAPEDEV
- LTAPESIZE
- TAPEBLK
- TAPEDEV
- TAPESIZE

- ON-Bar configuration parameters
  BAR_MAX_BACKUP, BAR_NB_XPORT_COUNT, BAR_PROGRESS_FREQ, and BAR_XFER_BUF_SIZE are some ON-Bar configuration parameters that affect background I/O.
- ontape configuration parameters (UNIX)
  On UNIX, LTAPEBLK, LTAPEDEV, LTAPESIZE, TAPEBLK, TAPEDEV, and TAPESIZE are configuration parameters that affect the **ontape** utility.

ON-Bar configuration parameters

BAR_MAX_BACKUP, BAR_NB_XPORT_COUNT, BAR_PROGRESS_FREQ, and BAR_XFER_BUF_SIZE are some ON-Bar configuration parameters that affect background I/O.

The BAR_MAX_BACKUP configuration parameter specifies the maximum number of backup processes per ON-Bar command. This configuration parameter also defines the degree of parallelism, determining how many processes start to run concurrently, including processes for backing up and restoring a whole system. When the number of running processes is reached, further processes start only when a running process completes its operation.

BAR_NB_XPORT_COUNT specifies the number of shared-memory data buffers for each backup or restore process.

BAR_PROGRESS_FREQ specifies, in minutes, how frequently the backup or restore progress messages display in the activity log.

BAR_XFER_BUF_SIZE specifies the size, in pages, of the buffers.

Related information:
BAR_MAX_BACKUP configuration parameter
BAR_NB_XPORT_COUNT configuration parameter
BAR_PROGRESS_FREQ configuration parameter
BAR_XFER_BUF_SIZE configuration parameter

ontape configuration parameters (UNIX)

On UNIX, LTAPEBLK, LTAPEDEV, LTAPESIZE, TAPEBLK, TAPEDEV, and TAPESIZE are configuration parameters that affect the **ontape** utility.

On UNIX, the LTAPEBLK, LTAPEDEV, and TAPESIZE configuration parameters specify the block size, device, and tape size for logical-log backups made with **ontape**. The TAPEBLK configuration parameter specifies the block size for database backups made with **ontape**, **onload**, and **onunload**.

TAPEDEV specifies the tape device. TAPESIZE specifies the tape size for these backups.

Related information:
ON-Bar and ontape configuration parameters and environment variable

Configuration parameters that affect rollback and recovery

The OFF_RECVRY_THREADS, ON_RECVRY_THREADS, PLOG_OVERFLOW_PATH, and RTO_SERVER_RESTART configuration parameters affect recovery. The LOW_MEMORY_RESERVE configuration parameter reserves a specific amount of memory, in kilobytes, for the database server to use when critical activities, such as rollback activities, are needed.

- OFF_RECVRY_THREADS and ON_RECVRY_THREADS and their effect on fast recovery
  The OFF_RECVRY_THREADS configuration parameter specifies the number of recovery threads that operate when the database server performs a cold restore or fast recovery. The setting of ON_RECVRY_THREADS specifies the number of recovery threads that operate when the database server performs a warm restore.
- PLOG_OVERFLOW_PATH and its effect on fast recovery
  The PLOG_OVERFLOW_PATH configuration parameter specifies the location of a disk file (named **plog_extend.servernum**) that the database server uses if the physical log file overflows during fast recovery.
- RTO_SERVER_RESTART and its effect on fast recovery
  The RTO_SERVER_RESTART configuration parameter enables you to use recovery time objective (RTO) standards to set the amount of time, in seconds, that Informix® has to recover from a problem after you restart Informix and bring it into online or quiescent mode.
- The LOW_MEMORY_RESERVE configuration parameter and memory utilization
  The LOW_MEMORY_RESERVE configuration parameter reserves a specific amount of memory, in kilobytes, for the database server to use when critical activities are needed and the server has limited free memory.

OFF_RECVRY_THREADS and ON_RECVRY_THREADS and their effect on fast recovery

The OFF_RECVRY_THREADS configuration parameter specifies the number of recovery threads that operate when the database server performs a cold restore or fast recovery. The setting of ON_RECVRY_THREADS specifies the number of recovery threads that operate when the database server performs a warm restore.

To improve the performance of fast recovery, increase the number of recovery threads with the OFF_RECVRY_THREADS configuration parameter. When fast recovery begins, the database server creates an LGR memory pool and allocates approximately 100 KB from this pool for each recovery thread. The LGR pool and its memory are freed when fast recovery completes. Because secondary servers in a high-availability cluster are almost always in fast recovery mode, the LGR memory pool is almost always present on secondary servers.

Follow these guidelines when you set the OFF_RECVRY_THREADS configuration parameter:

- If you have enough shared memory, set the number of threads to the number of tables or fragments that are frequently updated. Balance the number of threads with the amount of shared memory.
- On a single-CPU computer, set the number of threads to 10 - 30 or 40. The cost of too many threads can outweigh the advantages of parallel operations.

A warm restore takes place concurrently with other database operations. To reduce the impact of the warm restore on other users, you can allocate fewer threads to it than you might allocate to a cold restore. However, to replay logical-log transactions in parallel during a warm restore, specify more threads with the ON_RECVRY_THREADS configuration parameter.

Related information:
OFF_RECVRY_THREADS configuration parameter
ON_RECVRY_THREADS configuration parameter

PLOG_OVERFLOW_PATH and its effect on fast recovery

The PLOG_OVERFLOW_PATH configuration parameter specifies the location of a disk file (named **plog_extend.servernum**) that the database server uses if the physical log file overflows during fast recovery.

The database server removes the **plog_extend.servernum** file when the first checkpoint is performed during a fast recovery.

Related information:
PLOG_OVERFLOW_PATH configuration parameter

RTO_SERVER_RESTART and its effect on fast recovery

The RTO_SERVER_RESTART configuration parameter enables you to use recovery time objective (RTO) standards to set the amount of time, in seconds, that Informix® has to recover from a problem after you restart Informix and bring it into online or quiescent mode.

Related information:
RTO_SERVER_RESTART configuration parameter

The LOW_MEMORY_RESERVE configuration parameter and memory utilization

The LOW_MEMORY_RESERVE configuration parameter reserves a specific amount of memory, in kilobytes, for the database server to use when critical activities are needed and the server has limited free memory.

If you enable the new LOW_MEMORY_RESERVE configuration parameter by setting it to a specified value in kilobytes, critical activities, such as rollback activities, can complete even when you receive out-of-memory errors.

Related information:
LOW_MEMORY_RESERVE configuration parameter
onstat -g seg command: Print shared memory segment statistics

Configuration parameters that affect data replication and auditing

Data replication and auditing are optional. If you use these features, you can set configuration parameters that affect data-replication performance and auditing performance.

To obtain immediate performance improvements, you can disable these features, provided that the operating requirements for your system allow you to do so.

- Configuration parameters that affect data replication
  Synchronized data replication can increase the amount of time it take longer to free the log buffer after a log flush. The DRINTERVAL, DRTIMEOUT, and HDR_TXN_SCOPE configuration parameters can adjust synchronization and system performance.
- Configuration parameters that affect auditing
  The ADTERR and ADTMODE configuration parameters affect auditing performance.

Configuration parameters that affect data replication

Synchronized data replication can increase the amount of time it take longer to free the log buffer after a log flush. The DRINTERVAL, DRTIMEOUT, and HDR_TXN_SCOPE configuration parameters can adjust synchronization and system performance.

The DRINTERVAL configuration parameter indicates whether the data-replication buffer is flushed synchronously or asynchronously to the secondary database server. If this parameter is set to flush asynchronously, it specifies the interval between flushes. Each flush impacts the CPU and sends data across the network to the secondary database server.

If the DRINTERVAL configuration parameter is set to 0, the synchronization mode that is specified by the HDR_TXN_SCOPE configuration parameter is used. The HDR_TXN_SCOPE configuration parameter specifies whether HDR replication is fully synchronous, nearly synchronous, or asynchronous.

- In fully synchronous mode, transactions require acknowledgement of completion on the HDR secondary server before they can complete.
- In asynchronous mode, transactions do not require acknowledgement of being received or completed on the HDR secondary server before they can complete.
- In nearly synchronous mode, transactions require acknowledgement of being received on the HDR secondary server before they can complete.

The DRTIMEOUT configuration parameter specifies the interval for which either database server waits for a transfer acknowledgment from the other. If the primary database server does not receive the expected acknowledgment, it adds the transaction information to the file named in the DRLOSTFOUND configuration parameter. If the secondary database server receives no acknowledgment, it changes the data-replication mode as the DRAUTO configuration parameter specifies.

Related information:
DRINTERVAL configuration parameter
DRTIMEOUT configuration parameter
DRLOSTFOUND configuration parameter
DRAUTO configuration parameter
HDR_TXN_SCOPE configuration parameter
onstat -g dri command: Print high-availability data replication information
Replication of primary-server data to secondary servers
Fully synchronous mode for HDR replication
Nearly synchronous mode for HDR replication
Asynchronous mode for HDR replication

Configuration parameters that affect auditing

The ADTERR and ADTMODE configuration parameters affect auditing performance.

The ADTERR configuration parameter specifies whether the database server is to halt processing for a user session for which an audit record encounters an error. When ADTERR is set to halt such a session, the response time for that session appears to degrade until one of the successive attempts to write the audit record succeeds.

The ADTMODE configuration parameter enables or disables auditing according to the audit records that you specify with the **onaudit** utility. Records are written to files in the directory that the AUDITPATH parameter specifies. The AUDITSIZE parameter specifies the size of each audit-record file.

The effect of auditing on performance is largely determined by the auditing events that you choose to record. Depending on which users and events are audited, the impact of these configuration parameters can vary widely.

Infrequent events, such as requests to connect to a database, have low performance impact. Frequent events, such as requests to read any row, can generate a large amount of auditing activity. The more users for whom such frequent events are audited, the greater the impact on performance.

Related information:
ADTERR configuration parameter
ADTMODE configuration parameter
Auditing data security

LRU tuning

The LRU settings for flushing each buffer pool between checkpoints are not critical to checkpoint performance. The LRU settings are necessary only for maintaining enough clean pages for page replacement.

The default settings for LRU flushing are 50 percent for **lru_min_dirty** and 60 percent for **lru_max_dirty**.

If your database server has been configured for more aggressive LRU flushing because of checkpoint performance, you can decrease the LRU flushing at least to the default values.

The database server automatically tunes LRU flushing when the AUTO_LRU_TUNING configuration parameter is on and in the following cases:

- A page replacement is forced to perform a foreground write in order to find an empty page. In this case, LRU flushing is adjusted to be 5 percent more aggressive for the specific bufferpool where the foreground write took place.
- A page replacement is forced to use a buffer that is marked as high priority, meaning it is frequently accessed. In this case, LRU flushing is adjusted to be one (1) percent more aggressive for the specific bufferpool where the page replacement using high priority buffer took place.
- If the RTO_SERVER_RESTART configuration parameter is on and the time it takes to flush the bufferpool is longer than the recovery time objective, LRU flushing is adjusted to be 10 percent more aggressive for all bufferpools.

After a checkpoint has occurred, if a page replacement performed a foreground write during the previous checkpoint interval, the database server increases the LRU settings by 5 percent and continues to increase the LRU flushing at each subsequent checkpoint until the foreground write stops or until the **lru_max_dirty** for a given buffer pool falls below 10 percent. For example, if a page replacement performs a foreground write and the LRU settings for a buffer pool are `80` and `90`, the database server adjusts these to `76` and `85.5`.

In addition to foreground writes, LRU flushing is tuned more aggressively whenever a page fault replaces high priority buffers and non-high priority buffers are on the modified LRU queue. Automatic LRU adjustments only make LRU flushing more aggressive; they do not decrease LRU flushing. Automatic LRU adjustments are not permanent and are not recorded in the ONCONFIG file.

LRU flushing is reset to the values contained in the ONCONFIG file on which the database server starts.

The AUTO_LRU_TUNING configuration parameter specifies whether automatic LRU tuning is enabled or disabled when the server starts.

Related concepts:
Automatic checkpoints, LRU tuning, and AIO virtual processor tuning
Related information:
AUTO_LRU_TUNING configuration parameter
RTO_SERVER_RESTART configuration parameter

Table performance considerations

Some performance issues are associated with unfragmented tables and table fragments.

Issues include:

- Table placement on disk to increase throughput and reduce contention
- Space estimates for tables, blobpages, sbspaces, and extents
- Changes to tables that add or delete historical data
- Denormalization of the database to reduce overhead

- Placing tables on disk
  Tables that the database server supports reside on one or more portions of one or more disks. You control the placement of a table on disk when you create it by assigning it to a dbspace.
- Estimating table size
  You can calculate the approximate sizes (in disk pages) of tables.
- Managing the size of first and next extents for the tblspace tblspace
  The tblspace **tblspace** is a collection of pages that describe the location and structure of all tblspaces in a dbspace. Each dbspace has one tblspace **tblspace**. When you create a dbspace, you can use the TBLTBLFIRST and TBLTBLNEXT configuration parameters to specify the first and next extent sizes for the tblspace **tblspace** in a root dbspace.
- Managing sbspaces
  An *sbspace* is a logical storage unit composed of one or more chunks that store smart large objects. You can estimate the amount of storage needed for smart large objects, improve metadata I/O, monitor sbspaces, and change storage characteristics.
- Managing extents
  As you add rows to a table, the database server allocates disk space in units called *extents*. Each extent is a block of physically contiguous pages from the dbspace. Even when the dbspace includes more than one chunk, each extent is allocated entirely within a single chunk, so that it remains contiguous.
- Storing multiple table fragments in a single dbspace
  You can store multiple fragments of the same table or index in a single dbspace, thus reducing the total number of dbspaces needed for a fragmented table. You

must specify a name for each fragment that you want to store in the same dbspace. Storing multiple table or index fragments in a single dbspace simplifies the management of dbspaces.

- Displaying a list of table and index partitions
  Use the **onstat -g opn** option to display a list of the table and index partitions, by thread ID, that are currently open in the system.
- Changing tables to improve performance
  You can change tables to improve performance by dropping indexes, attaching or detaching fragments, and altering table definitions. You can also create databases for decision-support applications by unloading and loading tables in OLTP databases.
- Denormalize the data model to improve performance
  You might need to denormalize the data model to reduce overhead and optimize performance.
- Reduce disk space in tables with variable length rows
  You can enable the database server to insert more rows per page into tables with variable-length rows, if you set the MAX_FILL_DATA_PAGES configuration parameter to `1`. Allowing more variable length rows per page has advantages and disadvantages.
- Reduce disk space by compressing tables and fragments
  You can reduce disk space by compressing data in tables and table fragments. After compressing data, you can repack the data to consolidate the free space in a table or fragment, and shrink the space for the data to return the free space to the dbspace.

Placing tables on disk

Tables that the database server supports reside on one or more portions of one or more disks. You control the placement of a table on disk when you create it by assigning it to a dbspace.

Tables that the database server supports reside on one or more portions of a disk or disks. You control the placement of a table on disk when you create it by assigning it to a dbspace. A dbspace consists of one or more chunks. Each chunk corresponds to all or part of a disk partition. When you assign chunks to dbspaces, you make the disk space in those chunks available for storing tables or table fragments.

When you configure chunks and allocate them to dbspaces, you must relate the size of the dbspaces to the tables or fragments that each dbspace is to contain. To estimate the size of a table, follow the instructions in Estimating table size.

The database administrator (DBA) who is responsible for creating a table assigns that table to a dbspace in one of the following ways:

- By using the IN DBSPACE clause of the CREATE TABLE statement
- By using the dbspace of the current database
  The most recent DATABASE or CONNECT statement that the DBA issues before issuing the CREATE TABLE statement sets the current database.

The DBA can fragment a table across multiple dbspaces, as described in Planning a fragmentation strategy, or use the ALTER FRAGMENT statement to move a table to another dbspace. The ALTER FRAGMENT statement provides the simplest method for altering the placement of a table. However, the table is unavailable while the database server processes the alteration. Schedule the movement of a table or fragment at a time that affects the fewest users.

Other methods exist for moving tables between dbspaces:

- You can unload the data from a table and then move that data to another dbspace with the SQL statements LOAD and UNLOAD, the **onload** and **onunload** utilities or the High-Performance Loader (HPL).
- You can load data into and unload data from external tables.

Moving tables between databases with LOAD and UNLOAD, **onload** and **onunload**, or HPL involves periods in which data from the table is copied to tape and then reloaded onto the system. These periods present windows of vulnerability during which a table can become inconsistent with the rest of the database. To prevent the table from becoming inconsistent, you must restrict access to the version that remains on disk while the data transfers occur.

Depending on the size, fragmentation strategy, and indexes that are associated with a table, it can be faster to unload a table and reload it than to alter fragmentation. For other tables, it can be faster to alter fragmentation. You can experiment to determine which method is faster for a table that you want to move or re-partition.

- Isolating high-use tables
  You can place a table with high I/O activity on a dedicated disk device. Doing this reduces contention for the data that is stored in that table.
- Placing high-use tables on middle partitions of disks
  To minimize disk-head movement, place the most frequently accessed data on partitions close to the middle band of the disk (not near the center and not near the edge). This approach minimizes disk-head movement to reach data in the high-demand table.
- Using multiple disks
  You can use multiple disks for dbspaces, logical logs, temporary tables, and sort files.
- Backup and restore considerations when placing tables on disks
  When you decide where to place your tables or fragments, remember that if a device that contains a dbspace fails, all tables or table fragments in that dbspace are rendered inaccessible, even though tables and fragments in other dbspaces are accessible. The need to limit data unavailability in the event of a disk failure might influence which tables you group together in a particular dbspace.
- Factors affecting the performance of nonfragmented tables and table fragments
  Numerous factors affect the performance of an individual table or table fragment. These include the placement of the table or fragment, the size of the table or fragment, the indexing strategy that was used, the size and placement of table extents with respect to one another, and the frequency of access to the table.

Related information:
ALTER FRAGMENT statement
LOAD statement
UNLOAD statement
The onunload and onload utilities
Moving data with external tables
CREATE EXTERNAL TABLE Statement

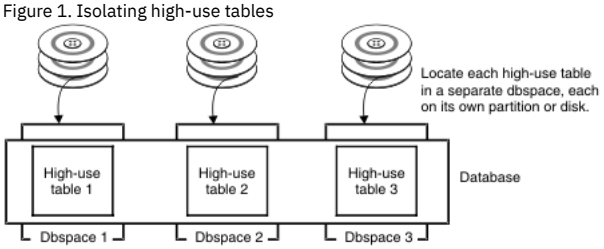
Isolating high-use tables

You can place a table with high I/O activity on a dedicated disk device. Doing this reduces contention for the data that is stored in that table.

When disk drives have different performance levels, you can put the tables with the highest use on the fastest drives. Placing two high-use tables on separate disk devices reduces competition for disk access when the two tables experience frequent, simultaneous I/O from multiple applications or when joins are formed between them.

To isolate a high-use table on its own disk device, assign the device to a chunk, assign that chunk to a dbspace, and then place the table in the dbspace that you created. Figure 1 shows three high-use tables, each in a separate dbspace, placed on three disks.

Figure 1. Isolating high-use tables

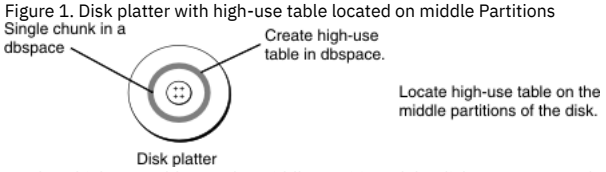


Placing high-use tables on middle partitions of disks

To minimize disk-head movement, place the most frequently accessed data on partitions close to the middle band of the disk (not near the center and not near the edge). This approach minimizes disk-head movement to reach data in the high-demand table.

The following figure shows the placement of the most frequently accessed data on partitions close to the middle band of the disk.

Figure 1. Disk platter with high-use table located on middle Partitions



To place high-use tables on the middle partition of the disk, create a raw device composed of cylinders that reside midway between the spindle and the outer edge of the disk. (For instructions on how to create a raw device, see the *IBM® Informix® Administrator's Guide* for your operating system.) Allocate a chunk, associating it with this raw device, as your *IBM Informix Administrator's Reference* describes. Then create a dbspace with this same chunk as the initial and only chunk. When you create a high-use table, place the table in this dbspace.

Using multiple disks

You can use multiple disks for dbspaces, logical logs, temporary tables, and sort files.

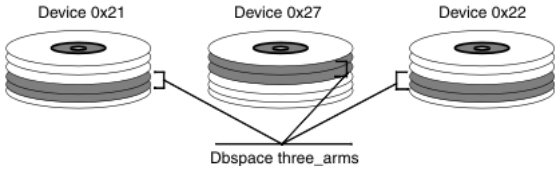
- Using multiple disks for a dbspace
  Using multiple disks for a dbspace helps to distribute I/O across dbspaces that contain several small tables.
- Using multiple disks for logical logs
  You can distribute logical logs in different dbspaces on multiple disks in round-robin fashion to improve logical backup performance. This scheme allows the database server to back up logs on one disk, while performing logging operations on the other disks.
- Spreading temporary tables and sort files across multiple disks
  You can spread the I/O associated with temporary tables and sort files across multiple disks, after defining dbspaces for temporary tables and sort files. This can improve performance for applications that require a large amount of temporary space for temporary tables or large sort operations.

Using multiple disks for a dbspace

Using multiple disks for a dbspace helps to distribute I/O across dbspaces that contain several small tables.

A dbspace can include multiple chunks, and each chunk can represent a different disk. The maximum size for a chunk is 4 terabytes. This arrangement allows you to distribute data in a dbspace over multiple disks. Figure 1 shows a dbspace distributed over three disks.

Figure 1. A dbspace distributed over three disks



Because you cannot use this type of distributed dbspace for parallel database queries (PDQ), you should use the table-fragmentation techniques described in Distribution schemes to partition large, high-use tables across multiple dbspaces.

Using multiple disks for logical logs

You can distribute logical logs in different dbspaces on multiple disks in round-robin fashion to improve logical backup performance. This scheme allows the database server to back up logs on one disk, while performing logging operations on the other disks.

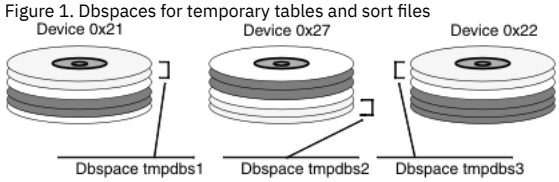
Keep your logical logs and the physical log on separate devices to improve performance by decreasing I/O contention on a single device. The logical and physical logs are created in the root dbspace when the database server is initialized. After initialization, you can move them to other dbspaces.

Spreading temporary tables and sort files across multiple disks

You can spread the I/O associated with temporary tables and sort files across multiple disks, after defining dbspaces for temporary tables and sort files. This can improve performance for applications that require a large amount of temporary space for temporary tables or large sort operations.

To define several dbspaces for temporary tables and sort files, use **onspaces -t**. When you place these dbspaces on different disks and list them in the DBSPACETEMP configuration parameter, you spread the I/O associated with temporary tables and sort files across multiple disks, as Figure 1 illustrates. You can list dbspaces that contain regular tables in DBSPACETEMP.

Figure 1. Dbspaces for temporary tables and sort files



DBSPACETEMP= tmpdbs1,tmpdbs2,tmpdbs3

Users can specify their own lists of dbspaces for temporary tables and sort files with the **DBSPACETEMP** environment variable. For details, see Configure dbspaces for temporary tables and sort files.

Backup and restore considerations when placing tables on disks

When you decide where to place your tables or fragments, remember that if a device that contains a dbspace fails, all tables or table fragments in that dbspace are rendered inaccessible, even though tables and fragments in other dbspaces are accessible. The need to limit data unavailability in the event of a disk failure might influence which tables you group together in a particular dbspace.

Although you must perform a cold restore if a dbspace that contains critical data fails, you need only perform a warm restore if a noncritical dbspace fails. The desire to minimize the impact of cold restores might influence the dbspace that you use to store critical data.

Factors affecting the performance of nonfragmented tables and table fragments

Numerous factors affect the performance of an individual table or table fragment. These include the placement of the table or fragment, the size of the table or fragment, the indexing strategy that was used, the size and placement of table extents with respect to one another, and the frequency of access to the table.

Estimating table size

You can calculate the approximate sizes (in disk pages) of tables.

For a description of size calculations for indexes, see Estimating index pages.

The disk pages allocated to a table are collectively referred to as a *tblspace*. The tblspace includes data pages. A separate tblspace includes index pages. If simple large objects (TEXT or BYTE data) are associated with a table that is not stored in an alternative dbspace, pages that hold simple large objects are also included in the tblspace.

The tblspace does not correspond to any fixed region within a dbspace. The data extents and indexes that make up a table can be scattered throughout the dbspace.

The size of a table includes all the pages within the tblspace: data pages and pages that store simple large objects. Blobpages that are stored in a separate blobspace are not included in the tblspace and are not counted as part of the table size.

The following sections describe how to estimate the page count for each type of page within the tblspace.

Tip: If an appropriate sample table exists, or if you can build a sample table of realistic size with simulated data, you do not need to make estimates. You can run **oncheck -pt** to obtain exact numbers.

- Estimating data pages
  How you estimate the data pages of a table depends on whether that table contains fixed-length or variable-length rows.
- Estimating pages that simple large objects occupy
  You can estimate the total number of pages for all simple large objects, or you can estimate the number of pages based on the median size of the simple large objects.

Estimating data pages

How you estimate the data pages of a table depends on whether that table contains fixed-length or variable-length rows.

- Estimating tables with fixed-length rows
  You can estimate the size (in pages) of a table with fixed-length rows. A table with fixed-length rows has no columns of the VARCHAR or NVARCHAR data type.
- Estimating tables with variable-length rows
  You can estimate the size of a table with variable-length rows with columns of the VARCHAR or NVARCHAR data type.
- Selecting an intermediate value for the size of the table
  The actual table size should fall somewhere between the projected number of data pages (*projsize*) and the maximum number of data pages (*maxsize*).

Estimating tables with fixed-length rows

You can estimate the size (in pages) of a table with fixed-length rows. A table with fixed-length rows has no columns of the VARCHAR or NVARCHAR data type.

Perform the following steps to estimate the size (in pages) of a table with fixed-length rows.

To estimate the page size, row size, number of rows, and number of data pages:

1. Use **onstat -b** to obtain the size of a page.
   The **buffer size** field in the last line of this output displays the page size.
2. Subtract 28 from this amount to account for the header that appears on each data page. The resulting amount is referred to as *pageuse*.
3. To calculate the size of a row, add the widths of all the columns in the table definition. TEXT and BYTE columns each use 56 bytes. If you have already created your table, you can use the following SQL statement to obtain the size of a row:

   ```
   SELECT rowsize FROM systables WHERE tabname =
      'table-name';
   ```

4. Estimate the number of rows that the table is expected to contain. This number is referred to as *rows*. The procedure for calculating the number of data pages that a table requires differs depending on whether the row size is less than or greater than *pageuse*.
5. If the size of the row is less than or equal to *pageuse*, use the following formula to calculate the number of data pages. The **trunc()** function notation indicates that you are to round down to the nearest integer.

   ```
   data_pages = rows / trunc(pageuse/(rowsize + 4))
   ```

   The maximum number of rows per page is 255, regardless of the size of the row.

   Important: Although the maximum size of a row that the database server accepts is approximately 32 kilobytes, performance degrades when a row exceeds the size of a page. For information about breaking up wide tables for improved performance, see Denormalize the data model to improve performance.
6. If the size of the row is greater than *pageuse*, the database server divides the row between pages. The page that contains the initial portion of a row is called the *home page*. Pages that contains subsequent portions of a row are called *remainder pages*. If a row spans more than two pages, some of the remainder pages are completely filled with data from that row. When the trailing portion of a row uses less than a page, it can be combined with the trailing portions of other rows to fill out the partial remainder page. The number of data pages is the sum of the home pages, the full remainder pages, and the partial remainder pages.
   a. Calculate the number of home pages.
      The number of home pages is the same as the number of rows:

      ```
      homepages = rows
      ```

   b. Calculate the number of full remainder pages.
      First calculate the size of the row remainder with the following formula:

      ```
      remsize = rowsize - (pageuse + 8)
      ```

      If *remsize* is less than *pageuse* – 4, you have no full remainder pages.

      If *remsize* is greater than *pageuse* – 4, use *remsize* in the following formula to obtain the number of full remainder pages:

```
fullrempages = rows * trunc(remsize/(pageuse - 8))
```

c. Calculate the number of partial remainder pages.
First calculate the size of a partial row remainder left after you have accounted for the home and full remainder pages for an individual row. In the following formula, the **remainder()** function notation indicates that you are to take the remainder after division:

```
partremsize = remainder(rowsize/(pageuse - 8)) + 4
```

The database server uses certain size thresholds with respect to the page size to determine how many partial remainder pages to use. Use the following formula to calculate the ratio of the partial remainder to the page:

```
partratio = partremsize/pageuse
```

Use the appropriate formula in the following table to calculate the number of partial remainder pages.

`partratio` Value	Formula to Calculate the Number of Partial Remainder Pages
Less than .1	partrempages = *rows*/(trunc((*pageuse*/10)/*remsize*) + 1)
Less than .33	partrempages = *rows*/(trunc((*pageuse*/3)/*remsize*) + 1)
.33 or larger	partrempages = *rows*

d. Add up the total number of pages with the following formula:

```
tablesize = homepages + fullrempages + partrempages
```


Estimating tables with variable-length rows

You can estimate the size of a table with variable-length rows with columns of the VARCHAR or NVARCHAR data type.

When a table contains one or more VARCHAR or NVARCHAR columns, its rows can have varying lengths. These varying lengths introduce uncertainty into the calculations. You must form an estimate of the typical size of each VARCHAR column, based on your understanding of the data, and use that value when you make the estimates.
Important: When the database server allocates space to rows of varying size, it considers a page to be full when no room exists for an additional row of the maximum size.
To estimate the size of a table with variable-length rows, you must make the following estimates and choose a value between them, based on your understanding of the data:

- The maximum size of the table, which you calculate based on the maximum width allowed for all VARCHAR or NVARCHAR columns
- The projected size of the table, which you calculate based on a typical width for each VARCHAR or NVARCHAR column

To estimate the maximum number of data pages:

1. To calculate *rowsize*, add together the maximum values for all column widths.
2. Use this value for *rowsize* and perform the calculations described in Estimating tables with fixed-length rows. The resulting value is called *maxsize*.

To estimate the projected number of data pages:

1. To calculate *rowsize*, add together typical values for each of your variable-width columns. It is suggested that you use the most frequently occurring width within a column as the typical width for that column. If you do not have access to the data or do not want to tabulate widths, you might choose to use some fractional portion of the maximum width, such as 2/3 (.67).
2. Use this value for *rowsize* and perform the calculations described in Estimating tables with fixed-length rows. The resulting value is called *projsize*.

Selecting an intermediate value for the size of the table

The actual table size should fall somewhere between the projected number of data pages (*projsize*) and the maximum number of data pages (*maxsize*).

Based on your knowledge of the data, choose a value within that range that seems most reasonable to you. The less familiar you are with the data, the more conservative (higher) your estimate should be.

Estimating pages that simple large objects occupy

You can estimate the total number of pages for all simple large objects, or you can estimate the number of pages based on the median size of the simple large objects.

The blobpages can reside in either the dbspace where the table resides or in a blobspace. For more information about when to use a blobspace, see Storing simple large objects in the tblspace or a separate blobspace.

The following methods for estimating blobpages yield a conservative (high) estimate because a single TEXT or BYTE column does not necessarily occupy the entire blobpage within a tblspace. In other words, a blobpage in a tblspace can contain multiple TEXT or BYTE columns.

To estimate the number of blobpages:

1. Obtain the page size with **onstat -b**.

2. Calculate the usable portion of the blobpage with the following formula:

   ```
   bpuse = pagesize - 32
   ```

3. For each byte of blobsize *n*, calculate the number of pages that the byte occupies (*bpages_n*) with the following formula:

   ```
   bpages1 = ceiling(bytesize1/bpuse)
   bpages2 = ceiling(bytesize2/bpuse)
   ...
   bpages_n = ceiling(bytesize_n/bpuse)
   ```

   The **ceiling()** function indicates that you should round up to the nearest integer value.

4. Add up the total number of pages for all simple large objects, as follows:

   ```
   blobpages = bpages1 + bpages2 + ... + bpagesn
   ```

Alternatively, you can base your estimate on the median size of simple large objects (TEXT or BYTE data); that is, the simple-large-object data size that occurs most frequently. This method is less precise, but it is easier to calculate.

To estimate the number of blobpages based on the median size of simple large objects:

1. Calculate the number of pages required for simple large objects of median size, as follows:

   ```
   mpages = ceiling(mblobsize/bpuse)
   ```

2. Multiply this amount by the total number of simple large objects, as follows:

   ```
   blobpages = blobcount * mpages
   ```

- Storing simple large objects in the tblspace or a separate blobspace
  When you create a simple-large-object column on magnetic disk, you have the option of storing the column data in the tblspace or in a separate blobspace. You can often improve performance by storing simple-large-object data in a separate blobspace, and by storing smart large objects and user-defined data in sbspaces.
- Estimating tblspace pages for simple large objects
  In your estimate of the space required for a table, include blobpages for any simple large objects that are to be stored in that tblspace. For a table that is both relatively small and nonvolatile, you can achieve the effect of a dedicated blobspace by separating row pages and blobpages.

Storing simple large objects in the tblspace or a separate blobspace

When you create a simple-large-object column on magnetic disk, you have the option of storing the column data in the tblspace or in a separate blobspace. You can often improve performance by storing simple-large-object data in a separate blobspace, and by storing smart large objects and user-defined data in sbspaces.

In the following example, a TEXT value is stored in the tblspace, and a BYTE value is stored in a blobspace named **rasters**:

```
CREATE TABLE examptab
   (
   pic_id SERIAL,
   pic_desc TEXT IN TABLE,
   pic_raster BYTE IN rasters
   )
```

For information about storing simple-large-object data in a separate blobspace, see Estimating pages that simple large objects occupy.
A TEXT or BYTE value is always stored apart from the rows of the table; only a 56-byte descriptor is stored with the row. However, a simple large object occupies at least one disk page. The simple large object to which the descriptor points can reside in the same set of extents on disk as the table rows (in the same tblspace) or in a separate blobspace.

When simple large objects are stored in the tblspace, the pages of their data are interspersed among the pages that contain rows, which can greatly increase the size of the table. When the database server reads only the rows and not the simple large objects, the disk arm must move farther than when the blobpages are stored apart. The database server scans only the row pages in the following situations:

- When it performs any SELECT operation that does not retrieve a simple-large-object column
- When it uses a filter expression to test rows

Another consideration is that disk I/O to and from a dbspace is buffered in shared memory of the database server. Pages are stored in case they are needed again soon, and when pages are written, the requesting program can continue before the actual disk write takes place. However, because blobspace data is expected to be voluminous, disk I/O to and from blobspaces is not buffered, and the requesting program is not allowed to proceed until all output has been written to the blobspace.

For best performance, store a simple-large-object column in a blobspace in either of the following circumstances:

- When single data items are larger than one or two pages each
- When the number of pages of TEXT or BYTE data is more than half the number of pages of row data

Estimating tblspace pages for simple large objects

In your estimate of the space required for a table, include blobpages for any simple large objects that are to be stored in that tblspace. For a table that is both relatively small and nonvolatile, you can achieve the effect of a dedicated blobspace by separating row pages and blobpages.

To separate row pages from blobpages within a dbspace:

1. Load the entire table with rows in which the simple-large-object columns are null.
2. Create all indexes. The row pages and the index pages are now contiguous.
3. Update all the rows to install the simple large objects. The blobpages now appear after the pages of row and index data within the tblspace.

Managing the size of first and next extents for the tblspace tblspace

The tblspace **tblspace** is a collection of pages that describe the location and structure of all tblspaces in a dbspace. Each dbspace has one tblspace **tblspace**. When you create a dbspace, you can use the TBLTBLFIRST and TBLTBLNEXT configuration parameters to specify the first and next extent sizes for the tblspace **tblspace** in a root dbspace.

You can use the **onspaces** utility to specify the initial and next extent sizes for the tblspace **tblspace** in non-root dbspaces.

Specify the initial and next extent sizes if you want to reduce the number of tblspace **tblspace** extents and reduce the frequency of situations when you need to place the tblspace **tblspace** extents in non-primary chunks.

The ability to specify a first extent size that is larger than the default provides flexibility for managing space. When you create an extent, you can reserve space during creation of the dbspace, thereby decreasing the risk of needing additional extents created in chunks that are not initial chunks.

You can only specify the first and next extent sizes when you create a dbspace. You cannot alter the specification of the first and next extents sizes after the creation of the dbspace. In addition, you cannot specify extent sizes for temporary dbspaces, sbspaces, blobspaces, or external spaces.

If you do not specify first and next extent sizes for the tblspace **tblspace**, Informix® uses the existing default extent sizes.

Related information:
TBLTBLFIRST configuration parameter
TBLTBLNEXT configuration parameter
Specifying the first and next extent sizes for the tblspace tblspace

Managing sbspaces

An *sbspace* is a logical storage unit composed of one or more chunks that store smart large objects. You can estimate the amount of storage needed for smart large objects, improve metadata I/O, monitor sbspaces, and change storage characteristics.

- Estimating pages that smart large objects occupy
  In your estimate of the space required for a table, you should also consider the amount of sbspace storage for any smart large objects (such as CLOB, BLOB, or multi-representative data types) that are part of the table. An sbspace contains user-data areas and metadata areas.
- Improving metadata I/O for smart large objects
  The metadata pages in an sbspace contain information about the location of the smart large objects in the sbspace. Typically, these pages are read intensive. You can improve metadata I/O by redistributing it.
- Monitoring sbspaces
  You can monitor the effectiveness of I/O operations on smart large objects. For better I/O performance, all smart large objects should be allocated in one extent to be contiguous.
- Changing storage characteristics of smart large objects
  When you create an sbspace, but do not specify values in the **-Df** option of the **onspaces -c -S** command, you use the defaults for the storage characteristics and attributes (such as logging and buffering). After you monitor sbspaces, you might want to change the storage characteristics, logging status, lock mode, or other attributes for new smart large objects.

Estimating pages that smart large objects occupy

In your estimate of the space required for a table, you should also consider the amount of sbspace storage for any smart large objects (such as CLOB, BLOB, or multi-representative data types) that are part of the table. An sbspace contains user-data areas and metadata areas.

CLOB and BLOB data is stored in sbpages that reside in the user-data area. The metadata area contains the smart-large-object attributes, such as average size and whether or not the smart large object is logged. For more information about sbspaces, see your *IBM® Informix® Administrator's Guide*.

- Estimating the size of the sbspace and metadata area
  The first chunk of an sbspace must have a metadata area. When you add smart large objects, the database server adds more control information to this metadata area.
- Sizing the metadata area manually for a new chunk
  Each chunk can contain metadata, but the sum total must accommodate enough room for all LO headers (average length 570 bytes each) and the chunk free list (which lists all the free extents in the chunk).

Estimating the size of the sbspace and metadata area

The first chunk of an sbspace must have a metadata area. When you add smart large objects, the database server adds more control information to this metadata area.

If you add a chunk to the sbspace after the initial allocation, you can take one of the following actions for metadata space:

- Allocate another metadata area on the new chunk by default.
  This action provides the following advantages:
  - It is easier because the database server automatically calculates and allocates a new metadata area on the added chunk based on the average smart large object size
  - Distributes I/O operations on the metadata area across multiple disks
- Use the existing metadata area
  If you specify the **onspaces -U** option, the database server does not allocate metadata space in the new chunk. Instead it must use a metadata area in one of the other chunks.

In addition, the database server reserves 40 percent of the user area to be used in case the metadata area runs out of space. Therefore, if the allocated metadata becomes full, the database server starts using this reserved space in the user area for additional control information.

You can let the database server calculate the size of the metadata area for you on the initial chunk and on each added chunks. However, you might want to specify the size of the metadata area explicitly, to ensure that the sbspace does not run out of metadata space and the 40 percent reserve area. You can use one of the following methods to explicitly specify the amount of metadata space to allocate:

- Specify the AVG_LO_SIZE tag on the **onspaces -Df** option.
  The database server uses this value to calculate the size of the metadata area to allocate when the **-Ms** option is not specified. If you do not specify AVG_LO_SIZE, the database server uses the default value of 8 kilobytes to calculate the size of the metadata area.
- Specify the metadata area size in the **-Ms** option of the **onspaces** utility.
  Use the procedure that Sizing the metadata area manually for a new chunk describes to estimate a value to specify in the **onspaces -Ms** option.

Sizing the metadata area manually for a new chunk

Each chunk can contain metadata, but the sum total must accommodate enough room for all LO headers (average length 570 bytes each) and the chunk free list (which lists all the free extents in the chunk).

The following procedure assumes that you know the sbspace size and need to allocate more metadata space.

To size the metadata area manually for a new chunk:

1. Use the **onstat -d** option to obtain the size of the current metadata area from the **Metadata size** field.
2. Estimate the number of smart large objects that you expect to reside in the sbspace and their average size.
3. Use the following formula to calculate the total size of the metadata area:

   ```
   Total metadata kilobytes = (LOcount*570)/1024 +
                              (numchunks*800) + 100
   ```

   LOcount
   is the number of smart large objects that you expect to have in all sbspace chunks, including the new one.
   numchunks
   is the total number of chunks in the sbspace.

4. To obtain the additional required area for metadata, subtract the current metadata size that you obtained in step 1 from the value that you obtained in step 3.
5. When you add another chunk, specify in the **-Ms** option of the **onspaces -a** command the value that you obtained in step 4.

- Example of calculating the metadata area for a new chunk
  This topic contains an example showing how to estimate the metadata size required for two sbspaces chunks.

Example of calculating the metadata area for a new chunk

This topic contains an example showing how to estimate the metadata size required for two sbspaces chunks.

Suppose the **Metadata size** field in the **onstat -d** option shows that the current metadata area is 1000 pages. If the system page size is 2048 bytes, the size of this metadata area is 2000 kilobytes, as the following calculation shows:

```
current metadata = (metadata_size * pagesize) / 1024
                 = (1000 * 2048) / 1024
               = 2000 kilobytes
```

Suppose you expect 31,000 smart large objects in the two sbspace chunks. The following formula calculates the total size of metadata area required for both chunks, rounding up fractions:

```
Total metadata = (LOcount*570)/1024 + (numchunks*800) + 100
               = (31,000 * 570)/1024 + (2*800) + 100
               = 17256 + 1600 + 100
             = 18956 kilobytes
```

To obtain the additional area that is required for metadata:

1. Subtract the current metadata size from the total metadata value.

```
Additional metadata = Total metadata - current metadata
                    = 18956 - 2000
                    = 16956 kilobytes
```

2. When you add the chunk to the sbspace, use the **-Ms** option of the **onspaces -a** command to specify a metadata area of 16,956 kilobytes.

```
% onspaces -a sbchk2 -p /dev/raw_dev1 -o 200 -Ms 16956
```


Improving metadata I/O for smart large objects

The metadata pages in an sbspace contain information about the location of the smart large objects in the sbspace. Typically, these pages are read intensive. You can improve metadata I/O by redistributing it.

You can distribute I/O to these pages in one of the following ways:

- Mirror the chunks that contain metadata.
  For more information about the implications of mirroring, see Consider mirroring for critical data components.
- Position the metadata pages on the fastest portion of the disk.
  Because the metadata pages are the most read-intensive part of an sbspace, place the metadata pages toward the middle of the disk to minimize disk seek time. To position metadata pages, use the **-Mo** option when you create the sbspace or add a chunk with the **onspaces** utility.
- Spread metadata pages across disks.
  To spread metadata pages across disks, create multiple chunks in an sbspace, with each chunk residing on a separate disk. When you add a chunk to the sbspace with the **onspaces** utility, specify the **-Ms** option to allocate pages for the metadata information.

  Although the database server attempts to keep the metadata information with its corresponding data in the same chunk, it cannot guarantee that they will be together.
- Decrease the number of extents each smart large object occupies.
  When a smart large object spans multiple extents, the metadata area contains a separate descriptor for each extent. To decrease the number of descriptor entries that must be read for each smart large object, specify the expected final size of the smart large object when you create the smart large object.

  The database server allocates the smart large object as a single extent (if it has contiguous storage in the chunk) when you specify the final size in either of the following functions:

  - The DataBlade API **mi_lo_specset_estbytes** function
  - The Informix® ESQL/C **ifx_lo_specset_estbytes** function

  For more information about the functions to open a smart large object and to set the estimated number of bytes, see the *IBM® Informix ESQL/C Programmer's Manual* and *IBM Informix DataBlade API Programmer's Guide*.

  For more information about sizing extents, see Sbspace extents.

Important: For highest data availability, mirror all sbspace chunks that contain metadata.

Monitoring sbspaces

You can monitor the effectiveness of I/O operations on smart large objects. For better I/O performance, all smart large objects should be allocated in one extent to be contiguous.

For more information about sizing extents, see Sbspace extents.

Contiguity provides the following I/O performance benefits:

- Minimizes the disk-arm motion
- Requires fewer I/O operations to read the smart large object
- When doing large sequential reads, can take advantage of lightweight I/O, which reads in larger blocks of data (60 kilobytes or more, depending on your platform) in a single I/O operation

You can use the following command-line utilities to monitor the effectiveness of I/O operations on smart large objects:

- **oncheck -cS, -pe** and **-pS**
- **onstat -g smb s** option

The following sections describe how to use these utility options to monitor sbspaces.

- Monitoring sbspaces with oncheck -cS
  The **oncheck -cS** option checks smart-large-object extents and the sbspace partitions in the user-data area.
- Monitoring sbspaces with oncheck -pe
  The **oncheck -pe** option displays information that includes the size in pages of the chunk, the number of pages used, the number of pages that are free, and a list of all the tables in the chunk, with the initial page number and the length of the table in pages. This option also shows if smart large objects occupy contiguous space within an sbspace.
- Monitoring sbspaces with oncheck -pS
  The **oncheck -pS** option displays information about smart-large-object extents and metadata areas in sbspace partitions. If you do not specify an sbspace name on

the command line, **oncheck** checks and displays the metadata for all sbspaces.

- Monitoring sbspaces with onstat -g smb
  The **onstat -g smb s** option displays sbspace attributes.

Monitoring sbspaces with oncheck -cS

The **oncheck -cS** option checks smart-large-object extents and the sbspace partitions in the user-data area.

Figure 1 shows an example of the output from the **-cS** option for **s9_sbspc**.

The values in the **Sbs#**, **Chk#**, and **Seq#** columns correspond to the **Space Chunk Page** value in the **-pS** output. The **Bytes** and **Pages** columns display the size of each smart large object in bytes and pages.

To calculate the average size of smart large objects, you can total the numbers in the **Size (Bytes)** column and then divide by the number of smart large objects. In Figure 1, the average number of bytes allocated is 2690, as the following calculation shows:

```
Average size in bytes = (15736 + 98 + 97 + 62 + 87 + 56) / 6
                      = 16136 / 6
                      = 2689.3
```

For information about how to specify smart large object sizes to influence extent sizes, see Sbspace extents.

Figure 1. oncheck -cS output

```
Validating space 's9_sbspc' ...

Large Objects
  ID                 Ref Size        Allocced        Creat   Last
  Sbs# Chk#  Seq#    Cnt    (Bytes)     Pages Extns Flags   Modified
  ---- ---- ------ ---- ---------- -------- ----- ----- -------------------------
     2    2      1    1      15736        8     1 N-N-H Thu Jun 21 16:59:12 2007
     2    2      2    1         98        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2      3    1         97        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2      4    1         62        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2      5    1         87        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2      6    1         56        1     1 N-K-H Thu Jun 21 16:59:12 2007
```

The **Extns** field shows the minimum extent size, in number of pages, allocated to each smart large object.

Monitoring sbspaces with oncheck -pe

The **oncheck -pe** option displays information that includes the size in pages of the chunk, the number of pages used, the number of pages that are free, and a list of all the tables in the chunk, with the initial page number and the length of the table in pages. This option also shows if smart large objects occupy contiguous space within an sbspace.

Execute **oncheck -pe** to display the following information to determine if the smart large objects occupy contiguous space within an sbspace:

- Identifies each smart large object with the term `SBLOBSpace LO`
  The three values in brackets following `SBLOBSpace LO` correspond to the **Sbs#**, **Chk#**, and **Seq#** columns in the **-cS** output.
- Offset of each smart large object
- Number of disk pages (*not* sbpages) used by each smart large object

Tip: The **oncheck -pe** option provides information about sbspace use in terms of database server pages, not sbpages.

Figure 1 shows sample output. In this example, the **size** field shows that the first smart large object occupies eight pages. Because the **offset** field shows that the first smart large object starts at page 53 and the second smart large object starts at page 61, the first smart large object occupies contiguous pages.

Figure 1. oncheck -pe output that shows contiguous space use

```
Chunk Pathname                                      Size      Used      Free
                                                    1000       940        60

    Description                                      Offset     Size
    ------------------------------------------------ -------- --------
    RESERVED PAGES                                          0        2
    CHUNK FREELIST PAGE                                     2        1
    s9_sbspc:'informix'.TBLSpace                            3       50
    SBLOBSpace LO [2,2,1]                                  53        8
    SBLOBSpace LO [2,2,2]                                  61        1
    SBLOBSpace LO [2,2,3]                                  62        1
    SBLOBSpace LO [2,2,4]                                  63        1
    SBLOBSpace LO [2,2,5]                                  64        1
    SBLOBSpace LO [2,2,6]                                  65        1
...
```


Monitoring sbspaces with oncheck -pS

The **oncheck -pS** option displays information about smart-large-object extents and metadata areas in sbspace partitions. If you do not specify an sbspace name on the command line, **oncheck** checks and displays the metadata for all sbspaces.

Figure 1 shows an example of the **-pS** output for **s9_sbspc**.

To display information about smart large objects, execute the following command:

```
oncheck -pS spacename
```

The **oncheck -pS** output displays the following information for each smart large object in the sbspace:

- Space chunk page
- Size in bytes of each smart large object
- Object ID that DataBlade API and Informix® ESQL/C functions use
- Storage characteristics of each smart large object

When you use **onspaces -c -S** to create an sbspace, you can use the **-Df** option to specify various storage characteristics for the smart large objects. You can use **onspaces -ch** to change attributes after the sbspace is created. The **Create Flags** field in the **oncheck -pS** output displays these storage characteristics and other attributes of each smart large object. In Figure 1, the **Create Flags** field shows LO_LOG because the LOGGING tag was set to ON in the **-Df** option.

Figure 1. oncheck -pS output

```
Space Chunk Page = [2,2,2] Object ID = 987122917
  LO SW Version              4
  LO Object Version          1
  Created by Txid            7
  Flags               0x31 LO_LOG LO_NOKEEP_LASTACCESS_TIME LO_HIGH_INTEG
  Data Type                  0
  Extent Size                -1
  IO Size                    0
  Created                    Thu Apr 12 17:48:35 2007
  Last Time Modified         Thu Apr 12 17:48:43 2007
  Last Time Accessed         Thu Apr 12 17:48:43 2007
  Last Time Attributes Modified Thu Apr 12 17:48:43 2007
  Ref Count                  1
  Create Flags        0x31 LO_LOG LO_NOKEEP_LASTACCESS_TIME LO_HIGH_INTEG
  Status Flags               0x0 LO_FROM_SERVER
  Size (Bytes)               2048
  Size Limit                 -1
  Total Estimated Size       -1
  Deleting TxId              -1
  LO Map Size                200
  LO Map Last Row            -1
  LO Map Extents             2
  LO Map User Pages          2
```


Monitoring sbspaces with onstat -g smb

The **onstat -g smb s** option displays sbspace attributes.

Use the **onstat -g smb s** option to display the following characteristics that affect the I/O performance of each sbspace:

- Logging status
  If applications are updating temporary smart large objects, logging is not required. You can turn off logging to reduce the amount of I/O activity to the logical log, CPU utilization, and memory resources.

- Average smart-large-object size
  Average size and extent size should be similar to reduce the number of I/O operations required to read in an entire smart large object. The **avg s/kb** output field shows the average smart-large-object size in kilobytes. In Figure 1, the **avg s/kb** output field shows the value `30` kilobytes.

  Specify the final size of the smart large object in either of the following functions to allocate the object as a single extent:

  - The DataBlade API **mi_lo_specset_estbytes** function
  - The Informix® ESQL/C **ifx_lo_specset_estbytes** function

  For more information about the functions to open a smart large object and to set the estimated number of bytes, see the *IBM® Informix ESQL/C Programmer's Manual* and *IBM Informix DataBlade API Programmer's Guide*.

- First extent size, next extent size, and minimum extent size
  The **1st sz/p**, **nxt sz/p**, and **min sz/p** output fields show these extent sizes if you set the extent tags in the **-Df** option of **onspaces**. In Figure 1, these output fields show values of `0` and `-1` because these tags are not set in **onspaces**.

Figure 1. onstat -g smb s output

```
sbnum 7    address 2afae48
  Space    : flags    nchk owner    sbname
             -------- 1    informix client
  Defaults : LO_LOG LO_KEEP_LASTACCESS_TIME

  LO       : ud b/pg flags    flags    avg s/kb max lcks
             2048    0        -------- 30       -1
  Ext/IO   : 1st sz/p nxt sz/p min sz/p mx io sz
             4        0        0        -1

  HdrCache : max   free
             512   0
```

Changing storage characteristics of smart large objects

When you create an sbspace, but do not specify values in the **-Df** option of the **onspaces -c -S** command, you use the defaults for the storage characteristics and attributes (such as logging and buffering). After you monitor sbspaces, you might want to change the storage characteristics, logging status, lock mode, or other attributes for new smart large objects.

The database administrator or programmer can use the following methods to override these default values for storage characteristics and attributes:

- The database administrator can use one of the following **onspaces** options:
  - Specify values when the sbspace is first created with the **onspaces -c -S** command.
  - Change values after the sbspace is created with the **onspaces -ch** command.

  Specify these values in the tag options of the **-Df** option of **onspaces**. For more information about the **onspaces** utility, see the *IBM® Informix Administrator's Reference*.
- The database administrator can specify values in the PUT clause of the CREATE TABLE or ALTER TABLE statements.
  These values override the values in the **onspaces** utility and are valid only for smart large objects that are stored in the associated column of the specific table. Other smart large objects (from columns in other tables) might also reside in this same sbspace. These other columns continue to use the storage characteristics and attributes of the sbspace that **onspaces** defined (or the default values, if **onspaces** did not define them) unless these columns also used a PUT clause to override them for a particular column.

  If you do not specify the storage characteristics for a smart-large-object column in the PUT clause, they are inherited from the sbspace.

  If you do not specify the PUT clause when you create a table with smart-large-object columns, the database server stores the smart large objects in the system default sbspace, which is specified by the SBSPACENAME configuration parameter in the ONCONFIG file. In this case, the storage characteristics and attributes are inherited from the SBSPACENAME sbspace.
- Programmers can use functions in the DataBlade API and Informix® ESQL/C to alter storage characteristics for a smart-large-object column.
  For information about the DataBlade API functions for smart large objects, see the *IBM Informix DataBlade API Programmer's Guide*. For information about the Informix ESQL/C functions for smart large objects, see the *IBM Informix ESQL/C Programmer's Manual*.

Table 1 summarizes the ways to alter the storage characteristics for a smart large object.

Table 1. Altering storage characteristics and other attributes of an sbspace

Storage Characteristic or Attribute	System Default Value	System-Specified Storage Characteristics Specified by -Df Option in onspaces Utility	Column-Level Storage Characteristics Specified by PUT clause of CREATE TABLE or ALTER TABLE	Storage Characteristics Specified by a DataBlade API Function	Storage Characteristics Specified by an ESQL/C Function
Last-access time	OFF	ACCESSTIME	KEEP ACCESS TIME, NO KEEP ACCESS TIME	Yes	Yes
Lock mode	BLOB	LOCK_MODE	No	Yes	Yes
Logging status	OFF	LOGGING	LOG, NO LOG	Yes	Yes
Data integrity	HIGH INTEG	No	HIGH INTEG, MODERATE INTEG	Yes	No
Size of extent	None	EXTENT_SIZE	EXTENT SIZE	Yes	Yes
Size of next extent	None	NEXT_SIZE	No	No	No
Minimum extent size	2 kilobytes on Windows 4 kilobytes on UNIX	MIN_EXT_SIZE	No	No	No
Size of smart large object	8 kilobytes	Average size of all smart large objects in sbspace: AVG_LO_SIZE	No	Estimated size of a particular smart large object Maximum size of a particular smart large object	Estimated size of a particular smart large object Maximum size of a particular smart large object
Buffer pool usage	ON	BUFFERING	No	LO_BUFFER and LO_NOBUFFER flags	LO_BUFFER and LO_NOBUFFER flags
Name of sbspace	SBSPACE-NAME	Not in **-Df** option. Name specified in **onspaces -S** *option.*	Name of an existing sbspace in which a smart large object resides: PUT ... IN clause	Yes	Yes
Fragmenta-tion across multiple sbspaces	None	No	Round-robin distribution scheme: PUT ... IN clause	Round-robin or expression-based distribution scheme	Round-robin or expression-based distribution scheme
Last-access time	OFF	ACCESSTIME	KEEP ACCESS TIME, NO KEEP ACCESS TIME	Yes	Yes

- Altering smart-large-object columns
  When you create or modify a table, you have several options for choosing storage characteristics and other attributes (such as logging status, buffering, data integrity, and locking granularity) for specific smart-large-object columns.

Altering smart-large-object columns

When you create or modify a table, you have several options for choosing storage characteristics and other attributes (such as logging status, buffering, data integrity, and locking granularity) for specific smart-large-object columns.

When you create or modify a table that can store BLOB or CLOB objects, you have these options:

- Use the values that were set when the sbspace was created. These values are specified in one of the following ways:
  - With the various flags of the **-Df** option of the **onspaces -c -S** command
  - With the system default value for any flag that was not specified.

  For guidelines to change the default storage characteristics of the **-Df** flags, see onspaces options that affect sbspace I/O.
- Use the PUT clause of the CREATE TABLE statement to specify non-default values for particular characteristics or attributes, including the number of sbspaces, the extent size, the logging, buffering, and data integrity status, and the locking granularity.
  Characteristics or attributes that you do not specify in the PUT clause default to the values set in the **onspaces -c -S** command, or to system default values (for example, no logging).

Later, you can use the PUT clause of the ALTER TABLE statement to change the optional storage characteristics of BLOB or CLOB columns. See Table 1 for characteristics and attributes of sbspaces that you can change.

You can use the PUT clause of the ALTER TABLE statement to perform the following actions:

- Specify the smart-large-object characteristics and storage location when you add a new BLOB or CLOB column to a table.
  The smart large objects in the new columns can have characteristics different from those in the existing columns.
- Change the smart-large-object characteristics of an existing column.
  The new column characteristics apply only to smart large objects in new rows inserted after the ALTER TABLE PUT statement was issued. The old characteristics persist for any smart large objects that already existed in the column before the ALTER TABLE PUT statement modified the column.

For example, the BLOB data in the **catalog** table in the **superstores_demo** database is stored in **s9_sbspc** with logging turned off and has an extent size of 100 kilobytes. You can use the PUT clause of the ALTER TABLE statement to turn on logging and store new smart large objects in a different sbspace.

For information about changing sbspace extents with the CREATE TABLE statement, see Extent sizes for smart large objects in sbspaces.

Related information:
Sbspace logging
CREATE TABLE statement

Managing extents

As you add rows to a table, the database server allocates disk space in units called *extents*. Each extent is a block of physically contiguous pages from the dbspace. Even when the dbspace includes more than one chunk, each extent is allocated entirely within a single chunk, so that it remains contiguous.

Contiguity is important to performance. When the pages of data are contiguous, and when the database server reads the rows sequentially during read-ahead, light scans, or lightweight I/O operations, disk-arm motion is minimized. For more information about these operations, see Sequential scans, Light scans, and Configuration parameters that affect sbspace I/O.

The mechanism of extents is a compromise between the following competing requirements:

- Most dbspaces are shared among several tables.
- The size of some tables is not known in advance.
- Tables can grow at different times and different rates.
- All the pages of a table should be adjacent for best performance.

If you have a table that needs more extents and the database server runs out of space on the partition header page, the database server automatically allocates extended secondary partition header pages to accommodate new extent entries. The database server can allocate up to 32767 extents for any partition, unless the size of a table dictates a limit to the number of extents.

Because table sizes are not known, the database server cannot preallocate table space. Therefore, the database server adds extents only as they are needed, but all the pages in any one extent are contiguous for better performance. In addition, when the database server creates an extent that is next to the previous one, it treats both as a single extent.

A frequently updated table can become fragmented over time which degrades the performance every time the table is accessed by the server. Defragmenting a table brings data rows closer together and avoids partition header page overflow problems.

- Choosing table extent sizes
  When you create a table, you can specify extent sizes for the data rows of a table in a dbspace and for each fragment of a fragmented table, and the smart large objects in an sbspace. The database server calculates extent sizes for smart large objects in sbspaces.
- Monitoring active tblspaces
  Monitor tblspaces to determine which tables are active. Active tables are those that a thread has currently opened.
- Monitoring the upper limit on extents and extent interleaving
  You can monitor the upper limit on the number of extents. You can also check for and eliminate extent interleaving.
- Reclaiming unused space within an extent
  After the database server allocates disk space to a tblspace as part of an extent, that space remains dedicated to the tblspace. Even if all extent pages become empty after you delete data, the disk space remains unavailable for use by other tables unless you reclaim the space.
- Managing extent deallocation with the TRUNCATE keyword
  TRUNCATE is an SQL keyword that quickly deletes active rows from a table and the b-tree structures of its indexes, without dropping the table or its schema, access

privileges, triggers, constraints, and other attributes. With this SQL data-definition language statement, you can depopulate a local table and reuse the table without re-creating it, or you can release the storage space that formerly held its data rows and b-tree structures.
- Defragment partitions to merge extents
  You can improve performance by defragmenting partitions to merge non-contiguous extents.

Choosing table extent sizes

When you create a table, you can specify extent sizes for the data rows of a table in a dbspace and for each fragment of a fragmented table, and the smart large objects in an sbspace. The database server calculates extent sizes for smart large objects in sbspaces.

- Extent sizes for tables in a dbspace
  When you create a table, you can specify the size of the first extent, as well as the size of the extents to be added as the table grows. You can also modify the size of the first extent in a table in a dbspace, and you can modify the size of new subsequent extents.
- Extent sizes for table fragments
  When you fragment an existing table, you might want to adjust the next-extent size because each fragment requires less space than the original, unfragmented table.
- Extent sizes for smart large objects in sbspaces
  When you create a table, you should use the extent size that the database server calculates for smart large objects in sbspaces. Alternatively, you can use the final size of the smart large object, as indicated by a particular function when you open the sbspace in an application program.

Extent sizes for tables in a dbspace

When you create a table, you can specify the size of the first extent, as well as the size of the extents to be added as the table grows. You can also modify the size of the first extent in a table in a dbspace, and you can modify the size of new subsequent extents.

The following sample SQL statement creates a table with a 512-kilobyte initial extent and 100-kilobyte added extents:

```
CREATE TABLE big_one (…column specifications…)
   IN big_space
   EXTENT SIZE 512
   NEXT SIZE 100
```

The default value for the extent size and the next-extent size is eight times the disk page size on your system. For example, if you have a 2-kilobyte page, the default length is 16 kilobytes.

You can use the ALTER TABLE statement with the MODIFY EXTENT SIZE clause to change the size of the first extent of a table in a dbspace. When you change the size of the first extent, Informix® records the change in the system catalog and on the partition page, but only makes the actual change when the table is rebuilt or a new partition or fragment is created.

You might want to change the size of the first extent of a table in a dbspace in either of these situations:

- If a table was created with small first extent size and you need to keep adding a lot of next extents, the table becomes fragmented across multiple extents and the data is scattered.
- If a table was created with a first extent that is much larger than the amount of data that is stored, space is wasted.

The following example changes the size of the first extent of a table in a dbspace to 50 kilobytes:

```
ALTER TABLE customer MODIFY EXTENT SIZE 50;
```

Changes to the first extent size are recorded into the system catalog table and on the partition page on the disk. However, changes to the first extent size do not take effect immediately. Instead, whenever a change that rebuilds the table occurs, the server uses the new first extent size.

For example, if a table has a first extent size of 8 kilobytes and you use the ALTER TABLE statement to change this to 16 kilobytes, the server does not drop the current first extent and recreate it with the new size. Instead, the new first extent size of 16 kilobytes takes effect only when the server rebuilds the table after actions such as creating a cluster index on the table or detaching a fragment from the table.

If a TRUNCATE TABLE statement without the REUSE option is executed before the ALTER TABLE statement with the MODIFY EXTENT SIZE clause, there is no change in the current first extent.

Use the MODIFY NEXT SIZE clause to change the size of the next extent to be added. This change does not affect next extents that already exist.

The following example changes the size of the next extent of a table to 50 kilobytes:

```
ALTER TABLE big_one MODIFY NEXT SIZE 50;
```

The next extent sizes of the following kinds of tables do not affect performance significantly:

- A small table is defined as a table that has only one extent. If such a table is heavily used, large parts of it remain buffered in memory.
- An infrequently used table is not important to performance no matter what size it is.
- A table that resides in a dedicated dbspace always receives new extents that are adjacent to its old extents. The size of these extents is not important because, being adjacent, they perform as one large extent.

Avoid creating large numbers of extents

When you assign an extent size to these kinds of tables, the only consideration is to avoid creating large numbers of extents. A large number of extents causes the database server to spend extra time finding the data. In addition, an upper limit exists on the number of extents allowed. (Considering the upper limit on extents covers this topic.)

Tips for allocating space for table extents

No upper limit exists on extent sizes except the size of the chunk. The maximum size for a chunk is 4 terabytes. When you know the final size of a table (or can confidently predict it within 25 percent), allocate all its space in the initial extent. When tables grow steadily to unknown size, assign them next-extent sizes that let them share the dbspace with a small number of extents each.

Allocating space for table extents

To allocate space for table extents:

1. Decide how to allocate space among the tables.
   For example, you might divide the dbspace among three tables in the ratio 0.4: 0.2: 0.3 (reserving 10 percent for small tables and overhead).
2. Give each table one-fourth of its share of the dbspace as its initial extent.
3. Assign each table one-eighth of its share as its next-extent size.
4. Monitor the growth of the tables regularly with **oncheck**.

As the dbspace fills up, you might not have enough contiguous space to create an extent of the specified size. In this case, the database server allocates the largest contiguous extent that it can.

Related information:
TBLTBLFIRST configuration parameter
TBLTBLNEXT configuration parameter
MODIFY EXTENT SIZE

Extent sizes for table fragments

When you fragment an existing table, you might want to adjust the next-extent size because each fragment requires less space than the original, unfragmented table.

If the unfragmented table was defined with a large next-extent size, the database server uses that same size for the next-extent on *each* fragment, which results in over-allocation of disk space. Each fragment requires only a proportion of the space for the entire table.

For example, if you fragment the preceding **big_one** sample table across five disks, you can alter the next-extent size to one-fifth the original size. The following example changes the next-extent size to one-fifth of the original size:

```
ALTER TABLE big_one MODIFY NEXT SIZE 2;
```

Related information:
MODIFY NEXT SIZE clause

Extent sizes for smart large objects in sbspaces

When you create a table, you should use the extent size that the database server calculates for smart large objects in sbspaces. Alternatively, you can use the final size of the smart large object, as indicated by a particular function when you open the sbspace in an application program.

You can use the final size of the smart large object when you open one of the following application programs:

- *For DB-Access:* Use the DataBlade API **mi_lo_specset_estbytes** function. For more information about the DataBlade API functions to open a smart large object and set the estimated number of bytes, see the *IBM® Informix DataBlade API Programmer's Guide*.
- *For ESQL/C:* Use the Informix® ESQL/C **ifx_lo_specset_estbytes** function. For more information about the Informix ESQL/C functions to open a smart large object and set the estimated number of bytes, see the *IBM Informix ESQL/C Programmer's Manual*.

For more information about sizing extents, see Sbspace extents. For more information, see Monitoring sbspaces.

Monitoring active tblspaces

Monitor tblspaces to determine which tables are active. Active tables are those that a thread has currently opened.

Output from the **onstat -t** option includes the tblspace number and the following four fields.

Field
Description
npages
Pages allocated to the tblspace

nused
Pages used from this allocated pool
nextns
Number of extents used
npdata
Number of data pages used

If a specific operation needs more pages than are available (**npages** minus **nused**), a new extent is required. If enough space is available in this chunk, the database server allocates the extent here; if not, the database server looks for space in other available chunks. If none of the chunks contains adequate contiguous space, the database server uses the largest block of contiguous space that it can find in the dbspace. Figure 1 shows an example of the output from this option.
Figure 1. onstat -t output

```
Tblspaces
 n address  flgs ucnt tblnum   physaddr npages nused  npdata nrows  nextns
 0 422528   1    1    100001   10000e   150    124    0      0      3
 1 422640   1    1    200001   200004   50     36     0      0      1
54 426038   1    6    100035   1008ac   3650   3631   3158   60000  3
62 4268f8   1    6    100034   1008ab   8      6      4      60     1
63 426a10   3    6    100036   1008ad   368    365    19     612    3
64 426b28   1    6    100033   1008aa   8      3      1      6      1
193 42f840  1    6    10001b   100028   8      5      2      30     1
 7 active, 200 total, 64 hash buckets
```


Monitoring the upper limit on extents and extent interleaving

You can monitor the upper limit on the number of extents. You can also check for and eliminate extent interleaving.

The maximum number of extents for a partition is 32767.

- Considering the upper limit on extents
  Do not allow a table to acquire a large number of extents because an upper limit exists on the number of extents allowed. Trying to add an extent after you reach the limit causes error -136 (`No more extents`) to follow an INSERT request.
- Checking for extent interleaving
  When two or more growing tables share a dbspace, extents from one tblspace can be placed between extents from another tblspace. When this situation occurs, the extents are said to be *interleaved*. Performance suffers when disk seeks for a table must span more than one extent, particularly for sequential scans.
- Eliminating interleaved extents
  You can eliminate interleaved extents by reorganizing the tables with the UNLOAD and LOAD statements, creating or altering an index to cluster, or using the ALTER TABLE statement.

Considering the upper limit on extents

Do not allow a table to acquire a large number of extents because an upper limit exists on the number of extents allowed. Trying to add an extent after you reach the limit causes error -136 (`No more extents`) to follow an INSERT request.

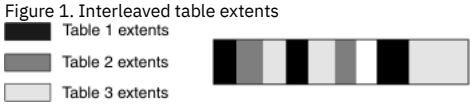
To help ensure that the limit is not exceeded, the database server performs the following actions:

- The database server checks the number of extents each time that it creates an extent. If the number of the extent being created is a multiple of 16, the database server automatically doubles the next-extent size for the table. Therefore, at every 16th creation, the database server doubles the next-extent size.
- When the database server creates an extent next to the previous extent, it treats both extents as a single extent.

Checking for extent interleaving

When two or more growing tables share a dbspace, extents from one tblspace can be placed between extents from another tblspace. When this situation occurs, the extents are said to be *interleaved*. Performance suffers when disk seeks for a table must span more than one extent, particularly for sequential scans.

Interleaving creates gaps between the extents of a table. Figure 1 shows gaps between table extents.

Figure 1. Interleaved table extents


Try to optimize the table-extent sizes to allocate contiguous disk space, which limits head movement. Also consider placing the tables in separate dbspaces.

Check periodically for extent interleaving by monitoring chunks. Execute **oncheck -pe** to obtain the physical layout of information in the chunk. The following information appears:

- Dbspace name and owner
- Number of chunks in the dbspace
- Sequential layout of tables and free space in each chunk

- Number of pages dedicated to each table extent or free space

This output is useful for determining the degree of extent interleaving. If the database server cannot allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly interleaved.

Eliminating interleaved extents

You can eliminate interleaved extents by reorganizing the tables with the UNLOAD and LOAD statements, creating or altering an index to cluster, or using the ALTER TABLE statement.

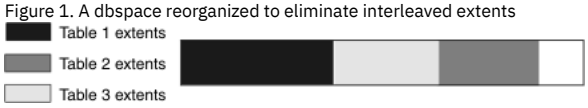
- Reorganizing dbspaces and tables to eliminate extent interleaving
  You can rebuild a dbspace to eliminate interleaved extents so that the extents for each table are contiguous.
- Creating or altering an index to cluster
  Depending on the circumstances, you can eliminate extent interleaving if you create a clustered index or alter a clustered index. When you use the TO CLUSTER clause of the CREATE INDEX or ALTER INDEX statement, the database server sorts and reconstructs the table.
- Using ALTER TABLE to eliminate extent interleaving
  If you use the ALTER TABLE statement to add or drop a column or to change the data type of a column, the database server copies and reconstructs the table. When the database server reconstructs the entire table, it rewrites the table to other areas of the dbspace. However, if other tables are in the dbspace, no guarantee exists that the new extents will be adjacent to each other.

Reorganizing dbspaces and tables to eliminate extent interleaving

You can rebuild a dbspace to eliminate interleaved extents so that the extents for each table are contiguous.

The order of the reorganized tables within the dbspace is not important, but the pages of each reorganized table should be contiguous so that no lengthy seeks are required to read the table sequentially. When the disk arm reads a table nonsequentially, it ranges only over the space that table occupies.

Figure 1. A dbspace reorganized to eliminate interleaved extents



To reorganize tables in a dbspace:

1. *For DB-Access users:* Copy the tables in the dbspace individually to tape with the UNLOAD statement in DB-Access.
2. Drop all the tables in the dbspace.
3. Re-create the tables with the LOAD statement or the **dbload** utility.

The LOAD statement re-creates the tables with the same properties they had before, including the same extent sizes.

You can also unload a table with the **onunload** utility and reload the table with the companion **onload** utility.

Related information:
LOAD statement
UNLOAD statement
The onunload and onload utilities

Creating or altering an index to cluster

Depending on the circumstances, you can eliminate extent interleaving if you create a clustered index or alter a clustered index. When you use the TO CLUSTER clause of the CREATE INDEX or ALTER INDEX statement, the database server sorts and reconstructs the table.

The TO CLUSTER clause reorders rows in the physical table to match the order in the index. For more information, see Clustering.

The TO CLUSTER clause eliminates interleaved extents under the following conditions:

- The chunk must contain contiguous space that is large enough to rebuild each table.
- The database server must use this contiguous space to rebuild the table.
  If blocks of free space exist before this larger contiguous space, the database server might allocate the smaller blocks first. The database server allocates space for the ALTER INDEX process from the beginning of the chunk, looking for blocks of free space that are greater than or equal to the size that is specified for the next extent. When the database server rebuilds the table with the smaller blocks of free space that are scattered throughout the chunk, it does not eliminate extent interleaving.

To display the location and size of the blocks of free space, execute the **oncheck -pe** command.

To use the TO CLUSTER clause of the ALTER INDEX statement:

1. For each table in the chunk, drop all fragmented or detached indexes except the one that you want to cluster.

2. Cluster the remaining index with the TO CLUSTER clause of the ALTER INDEX statement. This step eliminates interleaving the extents when you rebuild the table by rearranging the rows.
3. Re-create all the other indexes.

You do not need to drop an index before you cluster it. However, the ALTER INDEX process is faster than CREATE INDEX because the database server reads the data rows in cluster order using the index. In addition, the resulting indexes are more compact.

To prevent the problem from recurring, consider increasing the size of the tblspace extents.

Using ALTER TABLE to eliminate extent interleaving

If you use the ALTER TABLE statement to add or drop a column or to change the data type of a column, the database server copies and reconstructs the table. When the database server reconstructs the entire table, it rewrites the table to other areas of the dbspace. However, if other tables are in the dbspace, no guarantee exists that the new extents will be adjacent to each other.

Important: For certain types of operations that you specify in the ADD, DROP, and MODIFY clauses, the database server does not copy and reconstruct the table during the ALTER TABLE operation. In these cases, the database server uses an in-place alter algorithm to modify each row when it is updated (rather than during the ALTER TABLE operation). For more information about the conditions for this in-place alter algorithm, see In-place alter.

Reclaiming unused space within an extent

After the database server allocates disk space to a tblspace as part of an extent, that space remains dedicated to the tblspace. Even if all extent pages become empty after you delete data, the disk space remains unavailable for use by other tables unless you reclaim the space.

Important: When you delete rows in a table, the database server reuses that space to insert new rows into the same table. This section describes the procedures for reclaiming unused space for use by other tables.
You might want to resize a table that does not require the entire amount of space that was originally allocated to it. You can reallocate a smaller dbspace and release the unneeded space for other tables to use.

As the database server administrator, you can reclaim the disk space in empty extents and make it available to other users by rebuilding the table. To rebuild the table, use any of the following SQL statements:

- ALTER INDEX
- UNLOAD and LOAD
- ALTER FRAGMENT

- Reclaiming space in an empty extent with ALTER INDEX
  If the table with the empty extents includes an index, you can run the ALTER INDEX statement with the TO CLUSTER clause. Clustering an index rebuilds the table in a different location within the dbspace.
- Reclaiming space in an empty extent by unloading and re-creating or reloading a table
  If the table does not include an index, you can unload the table, re-create the table (either in the same dbspace or in another one), and reload the data with the UNLOAD and LOAD statements or the **onunload** and **onload** utilities.
- Releasing space in an empty extent with ALTER FRAGMENT
  You can use the ALTER FRAGMENT statement to rebuild a table. When you run this statement, it releases space within the extents that were allocated to that table.

Reclaiming space in an empty extent with ALTER INDEX

If the table with the empty extents includes an index, you can run the ALTER INDEX statement with the TO CLUSTER clause. Clustering an index rebuilds the table in a different location within the dbspace.

When you run the ALTER INDEX statement with the TO CLUSTER clause, all of the extents associated with the previous version of the table are released. Also, the newly built version of the table has no empty extents.

Related concepts:
Clustering
Related information:
ALTER INDEX statement

Reclaiming space in an empty extent by unloading and re-creating or reloading a table

If the table does not include an index, you can unload the table, re-create the table (either in the same dbspace or in another one), and reload the data with the UNLOAD and LOAD statements or the **onunload** and **onload** utilities.

Related information:
LOAD statement
UNLOAD statement
The onunload and onload utilities

Releasing space in an empty extent with ALTER FRAGMENT

You can use the ALTER FRAGMENT statement to rebuild a table. When you run this statement, it releases space within the extents that were allocated to that table.

For more information about the syntax of the ALTER FRAGMENT statement, see the *IBM® Informix® Guide to SQL: Syntax*.

Managing extent deallocation with the TRUNCATE keyword

TRUNCATE is an SQL keyword that quickly deletes active rows from a table and the b-tree structures of its indexes, without dropping the table or its schema, access privileges, triggers, constraints, and other attributes. With this SQL data-definition language statement, you can depopulate a local table and reuse the table without re-creating it, or you can release the storage space that formerly held its data rows and b-tree structures.

Two implementations of TRUNCATE exist:

- The first implementation, called "fast truncate," operates on most tables.
- The second implementation, called "slow truncate," operates on tables that include opaque or smart large object data types, or inherited indexes that are defined on ROW types within data type hierarchies.

The performance advantages of using the TRUNCATE TABLE statement instead of the DELETE statement are much better for the fast truncate implementation, because this implementation does not examine or run all of the rows in a table. Slow truncation implementation occurs on tables that include opaque or smart large object data types or inherited indexes that are defined on ROW types within data types, because the truncate operation examines each row containing these items.

For more information about using TRUNCATE, see the *IBM® Informix® Guide to SQL: Syntax*.

Defragment partitions to merge extents

You can improve performance by defragmenting partitions to merge non-contiguous extents.

A frequently updated table can become fragmented over time which degrades the performance every time the table is accessed by the server. Defragmenting a table brings data rows closer together and avoids partition header page overflow problems.

Defragmenting an index brings the entries closer together which improves the speed at which the table information is accessed.

You cannot stop a defragment request after the request has been submitted. Additionally, there are specific objects that cannot be defragmented and you cannot defragment a partition if another operation is running that conflicts with the defragment request.

Tip: Before you defragment a partition:

- Review the information about important limitations and considerations in Partition defragmentation.
- Run the **oncheck -pt and pT** command to determine the number of extents for a specific table or fragment.

To defragment a table, index, or partition, run the EXECUTE FUNCTION command with the defragment argument. You can specify the table name, index name, or partition number that you want to defragment.

You can use the **onstat -g defragment** command to display information about the active defragment requests.

Related information:
Scheduling data optimization
onstat -g defragment command: Print defragment partition extents
oncheck -pt and -pT: Display tblspaces for a Table or Fragment
defragment argument: Dynamically defragment partition extents (SQL administration API)

Storing multiple table fragments in a single dbspace

You can store multiple fragments of the same table or index in a single dbspace, thus reducing the total number of dbspaces needed for a fragmented table. You must specify a name for each fragment that you want to store in the same dbspace. Storing multiple table or index fragments in a single dbspace simplifies the management of

dbspaces.

You can also use this feature to improve query performance over storing each fragment in a different dbspace when a dbspace is located on a faster device.

For more information, see information about managing partitions in the *IBM® Informix® Administrator's Guide*.

Displaying a list of table and index partitions

Use the **onstat -g opn** option to display a list of the table and index partitions, by thread ID, that are currently open in the system.

For an example of **onstat -g opn** output and an explanation of output fields, see the *IBM® Informix® Administrator's Reference*.

Changing tables to improve performance

You can change tables to improve performance by dropping indexes, attaching or detaching fragments, and altering table definitions. You can also create databases for decision-support applications by unloading and loading tables in OLTP databases.

You might want to change an existing table for various reasons:

- To refresh large decision-support tables with data periodically
- To add or drop historical data from a certain time period
- To add, drop, or modify columns in large decision-support tables when the need arises for different data analysis

- Loading and unloading tables
  You can create databases for decision-support applications by periodically loading tables that have been unloaded from active OLTP databases.
- Dropping indexes for table-update efficiency
  In some applications, you can confine most table updates to a single time period. You can set up your system so that all updates are applied overnight or on specified dates. When updates are performed as a batch, you can drop all nonunique indexes while you make updates and then create new indexes afterward.
- Creating and enabling referential constraints efficiently
  When you create or enable foreign-key constraints on existing tables that contain data, you can sometimes achieve better performance by reducing the time that the database server spends searching for violating rows.
- Attaching or detaching fragments
  You can use ALTER FRAGMENT ATTACH and DETACH statements to perform data warehouse-type operations. ALTER FRAGMENT DETACH provides a way to delete a segment of the table data rapidly. Similarly, ALTER FRAGMENT ATTACH provides a way to load large amounts of data into an existing table incrementally by taking advantage of the fragmentation technology.
- Altering a table definition
  The database server uses one of these algorithms to process an ALTER TABLE statement in SQL: slow alter, in-place alter, or fast alter.

Loading and unloading tables

You can create databases for decision-support applications by periodically loading tables that have been unloaded from active OLTP databases.

You can use one or more of the following methods to load large tables quickly:

- External tables
- Nonlogging tables
  The database server provides support to:
  - Create nonlogging or logging tables in a logging database.
  - Alter a table from nonlogging to logging and vice versa.

  The two table types are STANDARD (logging tables) and RAW (nonlogging tables). You can use any loading utility such as **dbimport** or HPL to load raw tables.
- High-Performance Loader (HPL)
  You can use HPL in express mode to load tables quickly.

The following sections describe:

- Advantages of logging and nonlogging tables
- Step-by-step procedures to load data using nonlogging tables

- Advantages of logging tables
  Logging type options specify the logging characteristics that can improve performance in various bulk operations on the table.
- Advantages of nonlogging tables
  Nonlogging tables, which are also called raw tables, have characteristics that enable you to load very large data warehousing tables quickly.

Related information:
Moving data with external tables
CREATE EXTERNAL TABLE Statement

Advantages of logging tables

Logging type options specify the logging characteristics that can improve performance in various bulk operations on the table.

STANDARD, which corresponds to a table in a logged database of previous versions, is the default logging type that is used when you issue the CREATE TABLE statement without specifying the table type.

Standard tables have the following features:

- Logging to allow rollback, recovery, and restoration from archives.
- Recovery from backups
- All insert, delete, and update operations
- Constraints to maintain the integrity of your data
- Indexes to quickly retrieve a small number of rows

OLTP applications usually use standard tables. OLTP applications typically have the following characteristics:

- Real-time insert, update, and delete transactions
  Logging and recovery of these transactions is critical to preserve the data. Locking is critical to allow concurrent access and to ensure the consistency of the data selected.
- Update, insert, or delete one row or a few rows at a time
  Indexes speed access to these rows. An index requires only a few I/O operations to access the pertinent row, but scanning a table to find the pertinent row might require many I/O operations.

Advantages of nonlogging tables

Nonlogging tables, which are also called raw tables, have characteristics that enable you to load very large data warehousing tables quickly.

Raw tables have following characteristics:

- They do not use CPU and I/O resources for logging.
- They avoid problems such as running out of logical-log space.
- They are locked exclusively during an express load so that no other user can access the table during the load.
- They do not support referential constraints and unique constraints, so overhead for constraint-checking is eliminated.

- Quickly loading a large standard table
  You can change a large, existing standard table into a nonlogging table and then load the table.
- Quickly loading a new nonlogging table
  You quickly create a new nonlogging table and load the table.

Quickly loading a large standard table

You can change a large, existing standard table into a nonlogging table and then load the table.

To quickly load a large, existing standard table:

1. Drop indexes, referential constraints, and unique constraints.
2. Change the table to nonlogging.
   The following sample SQL statement changes a STANDARD table to nonlogging:

   ```
   ALTER TABLE largetab TYPE(RAW);
   ```

3. Load the table using a load utility such as **dbexport** or the High-Performance Loader (HPL). For more information about **dbexport** and **dbload**, see the *IBM® Informix® Migration Guide*. For more information about HPL, see the *IBM Informix High-Performance Loader User's Guide*.
4. Perform a level-0 backup of the nonlogging table. You must make a level-0 backup of any nonlogging table that has been modified before you convert it to STANDARD type. The level-0 backup provides a starting point from which to restore the data.
5. Change the nonlogging table to a logging table before you use it in a transaction. The following sample SQL statement changes a raw table to a standard table:

   ```
   ALTER TABLE largetab TYPE(STANDARD);
   ```

   Warning: Do not use nonlogging tables within a transaction where multiple users can modify the data. If you need to use a nonlogging table within a transaction, either set Repeatable Read isolation level or lock the table in exclusive mode to prevent concurrency problems.
   For more information about standard tables, see the previous section, Advantages of logging tables.

6. Re-create indexes, referential constraints, and unique constraints.

Quickly loading a new nonlogging table

You quickly create a new nonlogging table and load the table.

To quickly create and load a new, large table:

1. Create a nonlogging table in a logged database.
   The following sample SQL statements create a nonlogging table:

   ```
   CREATE DATABASE history WITH LOG;
   CONNECT TO DATABASE history;
   CREATE RAW TABLE history (...
   );
   ```

2. Load the table using a load utility such as **dbexport**. For more information about **dbexport** and **dbload**, see the *IBM® Informix® Migration Guide*.
3. Perform a level-0 backup of the nonlogging table.
   You must make a level-0 backup of any nonlogging table that has been modified before you convert it to STANDARD type. The level-0 backup provides a starting point from which to restore the data.
4. Change the nonlogging table to a logging table before you use it in a transaction.
   The following sample SQL statement changes a raw table to a standard table:

   ```
   ALTER TABLE largetab TYPE(STANDARD);
   ```

   Warning: Do not use nonlogging tables within a transaction where multiple users can modify the data. If you need to use a nonlogging table within a transaction, either set Repeatable Read isolation level or lock the table in exclusive mode to prevent concurrency problems.
   For more information about standard tables, see the previous section, Advantages of logging tables.
5. Create indexes on columns most often used in query filters.
6. Create any referential constraints and unique constraints, if needed.

Dropping indexes for table-update efficiency

In some applications, you can confine most table updates to a single time period. You can set up your system so that all updates are applied overnight or on specified dates. When updates are performed as a batch, you can drop all nonunique indexes while you make updates and then create new indexes afterward.

This strategy can have two positive effects:

- The updating program runs much faster if it does not need to update indexes at the same time that it updates tables.
- Re-created indexes are more efficient.

For more information about when to drop indexes, see Nonunique indexes.

To load a table that has no indexes:

1. Drop the table (if it exists).
2. Create the table without specifying any unique constraints.
3. Load all rows into the table.
4. Alter the table to apply the unique constraints.
5. Create the nonunique indexes.

If you cannot guarantee that the loaded data satisfies all unique constraints, you must create unique indexes before you load the rows. You save time if the rows are presented in the correct sequence for at least one of the indexes. If you have a choice, make it the row with the largest key. This strategy minimizes the number of leaf pages that must be read and written.

Creating and enabling referential constraints efficiently

When you create or enable foreign-key constraints on existing tables that contain data, you can sometimes achieve better performance by reducing the time that the database server spends searching for violating rows.

By maintaining the referential integrity of the database during DML operations, and by supporting efficient join-query execution paths on tables that are related by a star schema, foreign-key constraints can improve the performance of DML operations in databases where the primary key of each dimension table corresponds to a foreign key of the fact table.

When you use the ALTER TABLE ADD CONSTRAINT or ALTER TABLE MODIFY statement to define a foreign-key constraint on an existing table, you might be able to reduce the time required to validate of the new foreign-key constraint, if the referenced table already has a unique index or a primary-key constraint on the column corresponding to the key of the foreign-key constraint. When it creates a foreign-key constraint on a table that already contains data, the database server checks the table for any rows that violate the constraint. If an index exists, the database server makes a cost-based decision whether to scan every row in the table for violations, or to scan only the index valses.

For large tables, scanning only the index values can provide substantial performance improvement, unless one of the following requirements is not satisfied:

- The ALTER TABLE statement is creating only one foreign-key constraint.

- The ALTER TABLE statement is not also creating or enabling a CHECK constraint.
- The ALTER TABLE statement is not also changing the data type of any existing column in the table.
- The foreign-key columns do not include user-defined data types (UDTs) or built-in opaque data types.
- The new mode of the foreign-key constraint is not DISABLED.
- The table is not associated with an active violation table.

Except in the case of one or more violating rows, the ALTER TABLE ADD CONSTRAINT or ALTER TABLE MODIFY statement can create and validate a foreign-key constraint when some of these requirements are not satisfied, but the database server will not consider using the index-key algorithm to validate the foreign-key constraint. The additional validation costs to scan the entire table tend to be proportional to the size of the table.

Enabling a foreign-key constraint using index-scan validation

To validate the enabled foreign-key constraint, the database server performs a full-table scan to search for violating rows, unless a unique index or a primary-key constraint already exists on the foreign-key column values. In that case, the database server consider using an index scan for validation, unless one or more of the following requirements is not satisfied:

- The SET CONSTRAINTS statement is enabling only one foreign-key constraint.
- The same statement is not enabling a CHECK constraint.
- The foreign-key columns do not include user-defined data types (UDTs) or built-in opaque data types.
- The new mode of the foreign-key constraint is not DISABLED.
- The table is not associated with an active violation table.

Unless the table has one or more violating rows, the SET CONSTRAINTS statement can enable and validate a foreign-key constraint when some of these requirements are not satisfied, but the database server will not consider using the index-key algorithm to validate the foreign-key constraint. The additional validation costs for a full table scan can be substantial for very large tables.

Skipping validation of foreign-key constraints

In both the ALTER TABLE and SET CONSTRAINTS operations described above, the goal was to use a more efficient algorithm for validating the referential constraint. Greater efficiencies can be achieved, at least temporarily, by postponing or avoiding the validation of ENABLED or FILTERING foreign-key constraints that are being created by ALTER TABLE ADD CONSTRAINT statements, or while a DISABLED foreign-key constraint is being reset to an ENABLED or FILTERING mode.

This feature can be useful when tables that enforced referential constraints need to be moved from an OLTP environment to another database or to a data warehouse. To export the tables and restore their constraints without validation might be necessary if the time available for relocation is insufficient for violations checking. The tables might seem unlikely to include violating rows, if the constraints were dropped or disabled immediately before the tables were exported.

Three alternative mechanisms are available for bypassing the validation of enabled or filtering foreign-key constraints while they are being created, or while they are being exported, or while their mode is being changed from DISABLED:

- You can include the `NOVALIDATE` keyword in the constraint mode specification
  - of the ALTER TABLE ADD CONSTRAINT statement,
  - or of the SET CONSTRAINTS ENABLED statement,
  - or of the SET CONSTRAINTS FILTERING WITH ERROR statement,
  - or of the SET CONSTRAINTS FILTERING WITHOUT ERROR statements.
- If you plan to run multiple ALTER TABLE ADD CONSTRAINT or SET CONSTRAINTS statements, run the SET ENVIRONMENT NOVALIDATE ON statement to disable the validation of foreign-key constraints during the current session.
  Setting this session environment option makes NOVIOLATE the default mode for enabled or filtering referential constraints while the DDL statement is running.
- If you are migrating data, include the **-nv** option in the **dbimport** command.
  The effect of the **-nv** command-line option is that the constraint modes of any ALTER TABLE ADD CONSTRAINT or SET CONSTRAINTS statements that create or enable foreign-key constraints are processed so that the ENABLED, or FILTERING WITH ERROR, or FILTERING WITHOUT ERROR constraint mode specifications are instead implemented (respectively) as the ENABLED NOVALIDATE, or FILTERING WITH ERROR NOVALIDATE, or FILTERING WITHOUT ERROR NOVALIDATE modes.

In each case, no constraint validation of existing rows occurs during the DDL statement.

The effect of the NOVALIDATE keyword or of the **-nv** command-line flag of **dbimport** does not persist outside the DDL operation that created or changed the mode of the foreign-key constraint. The same constraint enforces referential integrity during subsequent DELETE, INSERT, MERGE, and UPDATE operations. The NOVALIDATE mode of the referential constraint is not registered in the sysobjstate system catalog table.

If a NOVALIDATE constraint mode is used on a table that might already contains rows that violate the foreign-key constraint, it is the responsibility of the user to verify that no violating rows exist in the data.

Attaching or detaching fragments

You can use ALTER FRAGMENT ATTACH and DETACH statements to perform data warehouse-type operations. ALTER FRAGMENT DETACH provides a way to delete a segment of the table data rapidly. Similarly, ALTER FRAGMENT ATTACH provides a way to load large amounts of data into an existing table incrementally by taking advantage of the fragmentation technology.

For more information about how to take advantage of the performance enhancements for the ATTACH and DETACH options of the ALTER FRAGMENT statement, see Improve the performance of operations that attach and detach fragments.

Altering a table definition

The database server uses one of these algorithms to process an ALTER TABLE statement in SQL: slow alter, in-place alter, or fast alter.

- Slow alter
  When the database server uses the slow alter algorithm to process an ALTER TABLE statement, the table can be unavailable to other users for a long period of time.
- In-place alter
  The in-place alter algorithm provides numerous performance advantages over the slow alter algorithm
- Fast alter
  The database server uses the fast alter algorithm when the ALTER TABLE statement changes attributes of the table but does not affect the data.

Slow alter

When the database server uses the slow alter algorithm to process an ALTER TABLE statement, the table can be unavailable to other users for a long period of time.

The table might be unavailable because the database server:

- Locks the table in exclusive mode for the duration of the ALTER TABLE operation
- Makes a copy of the table in order to convert the table to the new definition
- Converts the data rows during the ALTER TABLE operation
- Can treat the ALTER TABLE statement as a long transaction and abort it if the LTXHWM threshold is exceeded

Because the database server makes a copy of the table to convert the table to the new definition, a slow alter operation requires space at least twice the size of the original table plus log space.

The database server uses the slow alter algorithm when the ALTER TABLE statement makes column changes that it cannot perform in place:

- Adding or dropping a column created with the ROWIDS keyword
- Adding or dropping a column created with the REPLCHECK keyword
- Dropping a column of the TEXT or BYTE data type
- Modifying a SMALLINT column to SERIAL, SERIAL8, or BIGSERIAL
- Converting an INT column to SERIAL, SERIAL8, or BIGSERIAL
- Modifying the data type of a column so that some possible values of the old data type cannot be converted to the new data type (For example, if you modify a column of data type INTEGER to CHAR(n), the database server uses the slow alter algorithm if the value of `n` is less than `11`. An INTEGER requires 10 characters plus one for the minus sign for the lowest possible negative values.)
- Modifying the data type of a fragmentation column in a way that value conversion might cause rows to move to another fragment
- Adding, dropping or modifying any column when the table contains user-defined data types, smart large objects, or LVARCHAR, SET, MULTISET, ROW, or COLLECTION data types
- Modifying the original size or reserve specifications of VARCHAR or NVARCHAR columns
- Adding ERKEY shadow columns

In-place alter

The in-place alter algorithm provides numerous performance advantages over the slow alter algorithm

The in-place alter algorithm:

- Increases table availability
  Other users can access the table sooner when the ALTER TABLE operation uses the in-place alter algorithm, because the database server locks the table for only the time that it takes to update the table definition and rebuild indexes that contain altered columns.

  This increase in table availability can increase system throughput for application systems that require 24 by seven operations.

  When the database server uses the in-place alter algorithm, it locks the table for a shorter time than the slow alter algorithm because the database server:

  - Does not make a copy of the table to convert the table to the new definition
  - Does not convert the data rows during the ALTER TABLE operation
  - Alters the physical columns in place with the latest definition after the alter operation when you later update or insert rows. The database server converts the rows that reside on each page that you updated.
- Requires less space than the slow alter algorithm
  When the ALTER TABLE operation uses the slow alter algorithm, the database server makes a copy of the table to convert the table to the new definition. The ALTER TABLE operation requires space at least twice the size of the original table plus log space.

  When the ALTER TABLE operation uses the in-place alter algorithm, the space savings can be substantial for very large tables.

- Improves system throughput during the ALTER TABLE operation
  The database server does not log any changes to the table data during the in-place alter operation. Not logging changes has the following advantages:

  - Log space savings can be substantial for very large tables.
  - The alter operation is not a long transaction.

If the **check_for_ipa** Scheduler task is enabled, each table that has one or more outstanding in-place alter operations is listed in the **ph_alter** table in the **sysadmin** database. The alert text is: `Table database:owner.table_name has outstanding in place alters.` The alert type is informative.

- Conditions for in-place alter operations
  The database server can use the in-place alter algorithm to process only certain ADD, DROP, or MODIFY operations of the ALTER TABLE statement, and only if the

table schema or the ALTER TABLE statement does not require a slow alter algorithm.

- Performance considerations for DML statements
  The database server performs additional actions if it detects any down-level version page during the execution of data manipulation language (DML) statements (INSERT, UPDATE, DELETE, SELECT). These actions can impact performance.
- Performance of in-place alters for DDL operations
  In-place alter operations on data definition language (DDL) statements can slow performance. Therefore, monitor outstanding in-place alter operation because many outstanding alter operations affect subsequent ALTER TABLE statements.
- Altering a column that is part of an index
  If the altered column is part of an index, the table is still altered in place, but in this case the database server rebuilds the index or indexes implicitly. If you do not need to rebuild the index, you should drop or disable it before you perform the alter operation. Taking these steps improves performance.

Related information:
The ph_alert Table

Conditions for in-place alter operations

The database server can use the in-place alter algorithm to process only certain ADD, DROP, or MODIFY operations of the ALTER TABLE statement, and only if the table schema or the ALTER TABLE statement does not require a slow alter algorithm.

ALTER TABLE operations that can be done in place

The database server can use the in-place alter algorithm in the following ALTER TABLE operations:

- Add columns of built-in data types, except the data types that are listed in Conditions that prevent in-place alter operations.
- Drop a column of built-in data types, except a column that contains TEXT or BYTE data types, or a column that was created with the ROWIDS keyword.
- In Enterprise Replication, add or drop a column that is created with the CRCOLS keyword.
- Modify a column for which the database server can convert all possible values of the old data type to the new data type.
- Modify a column that is part of the fragmentation expression for its table, only if value changes do not require any data row to move from one fragment to another fragment after data type conversion.

The following table shows the conditions under which the ALTER TABLE MODIFY statement uses the in-place alter algorithm to convert columns of supported data types.

Key:

- All = The database server uses the in-place alter algorithm for all cases of the specific column operation.
- nf = The database server uses the in-place alter algorithm when the modified column is not part of the table fragmentation expression.

Table 1. MODIFY operations and conditions that use the in-place alter algorithm

Operation on Column	Condition
Convert a SMALLINT column to an INTEGER column	All
Convert a SMALLINT column to a BIGINT column	All
Convert a SMALLINT column to an INT8 column	All
Convert a SMALLINT column to a DEC(p2,s2) column	p2-s2 >= 5
Convert a SMALLINT column to a DEC(p2) column	p2-s2 >= 5 OR nf
Convert a SMALLINT column to a SMALLFLOAT column	All
Convert a SMALLINT column to a FLOAT column	All
Convert a SMALLINT column to a CHAR(n) column	n >= 6 AND nf
Convert an INT column to an INT8 column	All
Convert an INT column to a DEC(p2,s2) column	p2-s2 >= 10
Convert an INT column to a DEC(p2) column	p2 >= 10 OR nf
Convert an INT column to a SMALLFLOAT column	nf
Convert an INT column to a FLOAT column	All
Convert an INT column to a CHAR(n) column	n >= 11 AND nf
Convert a SERIAL column to an INT8 column	All
Convert a SERIAL column to a DEC(p2,s2) column	p2-s2 >= 10
Convert a SERIAL column to a DEC(p2) column	p2 >= 10 OR nf
Convert a SERIAL column to a SMALLFLOAT column	nf
Convert a SERIAL column to a FLOAT column	All
Convert a SERIAL column to a CHAR(n) column	n >= 11 AND nf
Convert a SERIAL column to a BIGSERIAL column	All
Convert a SERIAL column to a SERIAL8 column	All
Convert a SERIAL8 column to a BIGSERIAL column	All
Convert a BIGSERIAL column to a SERIAL8 column	All

Operation on Column	Condition
Convert a DEC(p1,s1) column to a SMALLINT column	p1-s1 < 5 AND (s1 == 0 OR nf)
Convert a DEC(p1,s1) column to an INTEGER column	p1-s1 < 10 AND (s1 == 0 OR nf)
Convert a DEC(p1,s1) column to an INT8 column	p1-s1 < 20 AND (s1 == 0 OR nf)
Convert a DEC(p1,s1) column to a SERIAL column	p1-s1 < 10 AND (s1 == 0 OR nf)
Convert a DEC(p1,s1) column to a BIGSERIAL column	p1-s1 < 20 AND (s1 == 0 OR nf)
Convert a DEC(p1,s1) column to a SERIAL8 column	p1-s1 < 20 AND (s1 == 0 OR nf)
Convert a DEC(p1,s1) column to a DEC(p2,s2) column	p2-s2 >= p1-s1 AND (s2 >= s1 OR nf)
Convert a DEC(p1,s1) column to a DEC(p2) column	p2 >= p1 OR nf
Convert a DEC(p1,s1) column to a SMALLFLOAT column	nf
Convert a DEC(p1,s1) column to a FLOAT column	nf
Convert a DEC(p1,s1) column to a CHAR(n) column	n >= 8 AND nf
Convert a DEC(p1) column to a DEC(p2) column	p2 >= p1 OR nf
Convert a DEC(p1) column to a SMALLFLOAT column	nf
Convert a DEC(p1) column to a FLOAT column	nf
Convert a DEC(p1) column to a CHAR(n) column	n >= 8 AND nf
Convert a SMALLFLOAT column to a DEC(p2) column	nf
Convert a SMALLFLOAT column to a FLOAT column	nf
Convert a SMALLFLOAT column to a CHAR(n) column	n >= 8 AND nf
Convert a FLOAT column to a DEC(p2) column	nf
Convert a FLOAT column to a SMALLFLOAT column	nf
Convert a FLOAT column to a CHAR(n) column	n >= 8 AND nf
Convert a CHAR(m) column to a CHAR(n) column	n >= m OR (nf AND not ANSI mode)
Increase the length of a character-type column	Not in ANSI mode databases
Increase the length of a DECIMAL or MONEY column	All
Convert an INT column to a SERIAL column	All
Convert an INT column to a BIGSERIAL column	All
Convert an INT column to a SERIAL8 column	All
Convert a BIGINT column to a BIGSERIAL column	All
Convert a BIGINT column to a SERIAL8 column	All
Convert a INT8 column to a BIGSERIAL column	All
Convert a INT8 column to a SERIAL8 column	All

Note: If first column of an index is altered, the operation to find the next serial value is very fast as it can make use of the index. If altered column is not first column of an index, the operation will do a sequential scan of the table to find the next serial value.
If you supply the serial value of the altered column, the operation is fast as the serial value is provided and does not require any calculation.

Conditions that prevent in-place alter operations

When the table contains an opaque data type, a user-defined data type, an LVARCHAR data type, a BOOLEAN data type, or a smart large object (BLOB or CLOB), the database server does not use the in-place alter algorithm, even when the column that is being altered is of a data type that can support in-place alter operations.

The in-place alter algorithm is not used if the ALTER TABLE DROP statement specifies BYTE or TEXT columns, or the ROWIDS keyword, or if the ALTER TABLE ADD statement includes the ROWID keyword.

If any column data types in an ALTER TABLE MODIFY statement cannot be converted by in-place alter operations, or if data movement is required for a fragmented table, the database server uses the slow alter algorithm for data type conversion instead of using the in-place alter algorithm.

For example, the database server does not use the in-place alter algorithm in the following situations:

- When more than one algorithm is needed
  For example, assume that an ALTER TABLE MODIFY statement converts a SMALLINT column to a DEC(8,2) column and converts an INTEGER column to a CHAR(8) column. The conversion of the first column is an in-place alter operation, but the conversion of the second column is a slow alter operation. The database server uses the slow alter algorithm to execute this statement.
- When the ALTER TABLE operation moves data records to another fragment
  For example, suppose you have a table with two integer columns and the following fragment expression:

  ```
  col1 < col2 IN dbspace1, REMAINDER IN dbspace2
  ```

  If you issue an ALTER TABLE MODIFY statement to convert the integer values to character values, the database server stores the row `(4, 30)` in **dbspace1** before the alter operation, but stores it in **dbspace2** after the alter operation, not as integers, `4 < 30`, but as characters, `'30' < '4'`.
- When the database server cannot convert all possible values of the old data type to the new data type.

For example, you cannot convert a BIGSERIAL column to a SERIAL column, because the modified column cannot store BIGSERIAL values that are beyond the range of SERIAL values. (However, you can change a column from SERIAL to BIGSERIAL with an in-place alter operation, if other columns in the table do not conflict with any of the other restrictions on in-place alter operations.)

Related information:
IBMInformix data types
DECIMAL

Performance considerations for DML statements

The database server performs additional actions if it detects any down-level version page during the execution of data manipulation language (DML) statements (INSERT, UPDATE, DELETE, SELECT). These actions can impact performance.

Each time you execute an ALTER TABLE statement that uses the in-place alter algorithm, the database server creates a new version of the table structure. The database server keeps track of all versions of table definitions. The database server resets the version status and all of the version structures and alter structures until the entire table is converted to the final format, or until a slow alter is performed.

If the database server detects any down-level version page during the execution of DML statements (INSERT, UPDATE, DELETE, and SELECT statements, and MERGE statements that specify Insert, Update, or Delete clauses), it performs the following actions:

- For UPDATE statements, the database server converts the entire data page or pages to the final format.
- For INSERT statements, the database server converts the inserted row to the final format and inserts it in the best-fit page. The database server converts the existing rows on the best-fit page to the final format.
- For DELETE statements, the database server does not convert the data pages to the final format.
- For SELECT statements, the database server does not convert the data pages to the final format.
  If your query accesses rows that are not yet converted to the new table definition, you might notice a slight degradation in the performance of your individual query, because the database server reformats each row before it is returned.

Performance of in-place alters for DDL operations

In-place alter operations on data definition language (DDL) statements can slow performance. Therefore, monitor outstanding in-place alter operation because many outstanding alter operations affect subsequent ALTER TABLE statements.

The **oncheck -pT** command displays data-page versions for outstanding in-place alter operations. An in-place alter is *outstanding* when data pages still exist with the old definition.

Figure 1 shows a portion of the output that the following **oncheck** command produces after four in-place alter operations are run on the **customer** demonstration table:
Figure 1. Sample oncheck -pT output for the customer table

```
oncheck -pT stores_demo:customer

...
Home Data Page Version Summary

            Version            Count

            0 (oldest)             2
            1                      0
            2                      0
            3                      0
            4 (current)            0
...
```

The **Count** field in Figure 1 displays the number of pages that currently use that version of the table definition. This **oncheck** output shows that four versions are outstanding:

- A value of `2` in the **Count** field for the oldest version indicates that two pages use the oldest version.
- A value of `0` in the **Count** fields for the next four versions indicates that no pages were to the latest table definition.

Important: As you perform more in-place alter operation on a table, each subsequent ALTER statement or the SQL statements that run against the tables with outstanding alters take more time to run than the previous statement. To maintain efficient performance, regularly remove outstanding in-place alter operations.
You can remove in-place alter operations by running the **admin( )** or **task( )** SQL administration command with the **table update_ipa** or **fragment update_ipa** argument. You can include the **parallel** option to run the operation in parallel. For example, the following statement removes in-place alter operations in parallel from a table that is named auto:

```
EXECUTE FUNCTION task('table update_ipa parallel','auto');
```

If your goal is saving runtime CPU, then plan to keep as few outstanding alters operations on a table as possible (generally no more than 3 or 4). If your goal is to save on disk space and your alter operations add or grow columns, then leaving in-place alters outstanding helps reduce disk space. If you need to revert to an earlier version of the database server, however, one requirement is that no data pages can include incomplete ALTER TABLE or ALTER FRAGMENT operations.

After all outstanding in-place alter operations have been completed on a table or fragment, the **oncheck -pT** command displays the total number of data pages in the **Count** field for the current version of the table.

Related information:
Resolve outstanding in-place alter operations

Altering a column that is part of an index

If the altered column is part of an index, the table is still altered in place, but in this case the database server rebuilds the index or indexes implicitly. If you do not need to rebuild the index, you should drop or disable it before you perform the alter operation. Taking these steps improves performance.

However, if the column that you modify is a primary key or foreign key and you want to keep this constraint, you must specify those keywords again in the ALTER TABLE statement, and the database server rebuilds the index.

For example, suppose you create tables and alter the parent table with the following SQL statements:

```
CREATE TABLE parent
   (si SMALLINT PRIMARY KEY CONSTRAINT pkey);
CREATE TABLE child
   (si SMALLINT REFERENCES parent ON DELETE CASCADE
   CONSTRAINT ckey);
INSERT INTO parent (si) VALUES (1);
INSERT INTO parent (si) VALUES (2);
INSERT INTO child (si) VALUES (1);
INSERT INTO child (si) VALUES (2);
ALTER TABLE parent
   MODIFY (si INT PRIMARY KEY CONSTRAINT pkey);
```

This ALTER TABLE example converts a SMALLINT column to an INT column. The database server retains the primary key because the ALTER TABLE statement specifies the PRIMARY KEY keywords and the **pkey** constraint. When you specify a PRIMARY KEY constraint in the MODIFY clause, the database server also silently creates a NOT NULL constraint on the same primary key column. However, the database server drops any referential constraints to that primary key. Therefore, you must also specify the following ALTER TABLE statement for the child table:

```
ALTER TABLE child
   MODIFY (si int references parent on delete cascade
         constraint ckey);
```

Even though the ALTER TABLE operation on a primary key or foreign key column rebuilds the index, the database server still takes advantage of the in-place alter algorithm. The in-place alter algorithm can provide performance benefits, including the following:

- It does not make a copy of the table in order to convert the table to the new definition.
- It does not convert the data rows during the alter operation.
- It does not rebuild all indexes on the table.

Warning: If you alter a table that is part of a view, you must re-create the view to obtain the latest definition of the table.

Fast alter

The database server uses the fast alter algorithm when the ALTER TABLE statement changes attributes of the table but does not affect the data.

The database server uses the fast alter algorithm when you use the ALTER TABLE statement to:

- Change the next-extent size.
- Add or drop a constraint.
- Change the lock mode of the table.
- Change the unique index attribute without modifying the column type.
- Add shadow columns for row versioning with the ADD VERCOLS keywords.

With the fast alter algorithm, the database server holds the lock on the table for just a short time. In some cases, the database server locks the system catalog tables only to change the attribute. In either case, the table is unavailable for queries for only a short time.

Denormalize the data model to improve performance

You might need to denormalize the data model to reduce overhead and optimize performance.

The entity-relationship data model, which the *IBM® Informix® Guide to SQL: Tutorial* describes, produces tables that contain no redundant or derived data. According to the tenets of relational database theory, these tables are well structured.

Sometimes, to meet extraordinary demands for high performance, you might need to denormalize the data model by modifying it in ways that are undesirable from a theoretical standpoint. This section describes some modifications and their associated costs.

- Shortening rows
  Usually, tables with shorter rows yield better performance than those with longer rows because disk I/O is performed in pages, not in rows. The shorter the rows of a table, the more rows occur on a page. The more rows per page, the fewer I/O operations it takes to read the table sequentially, and the more likely it is that a nonsequential access can be performed from a buffer.
- Expelling long strings
  The most bulky attributes are often character strings. To make the rows shorter, you can remove long strings from the entity table.

- Splitting wide tables
  Consider all the attributes of an entity that has rows that are too wide for good performance. Look for some theme or principle to divide them into two groups. Then split the table into two tables, a primary table and a companion table, repeating the primary key in each one.
- Redundant data
  Normalized tables contain no redundant data. Every attribute appears in only one table.

Shortening rows

Usually, tables with shorter rows yield better performance than those with longer rows because disk I/O is performed in pages, not in rows. The shorter the rows of a table, the more rows occur on a page. The more rows per page, the fewer I/O operations it takes to read the table sequentially, and the more likely it is that a nonsequential access can be performed from a buffer.

The entity-relationship data model puts all the attributes of one entity into a single table for that entity. For some entities, this strategy can produce rows of awkward lengths.

To shorten the rows, you can break columns into separate tables that are associated by duplicate key values in each table. As the rows get shorter, query performance should improve.

Expelling long strings

The most bulky attributes are often character strings. To make the rows shorter, you can remove long strings from the entity table.

You can use the following methods to expel long strings:

- Use VARCHAR columns.
- Use TEXT data.
- Move strings to a companion table.
- Build a symbol table.

- Convert CHAR columns into VARCHAR columns to shorten rows (GLS)
  A database might contain CHAR columns that you can convert to VARCHAR columns. You can use a VARCHAR column to shorten the average row length when the average length of the text string in the CHAR column is at least 2 bytes shorter than the width of the column.
- Convert a long string to a TEXT data type column
  When a string fills half a disk page or more, consider converting it to a TEXT data type column in a separate blobspace.
- Move strings to a companion table
  Strings that are less than half a page waste disk space if you treat them as TEXT data, but you can move them from the main table to a companion table.
- Build a symbol table
  If a column contains strings that are not unique in each row, you can move those strings to a table in which only unique copies are stored.

Convert CHAR columns into VARCHAR columns to shorten rows (GLS)

A database might contain CHAR columns that you can convert to VARCHAR columns. You can use a VARCHAR column to shorten the average row length when the average length of the text string in the CHAR column is at least 2 bytes shorter than the width of the column.

VARCHAR data is immediately compatible with most existing programs, forms, and reports. You might need to recompile any forms produced by application development tools to recognize VARCHAR columns. Always test forms and reports on a sample database after you modify the table schema.

For information about other character data types, see the *IBM® Informix® GLS User's Guide*.

Convert a long string to a TEXT data type column

When a string fills half a disk page or more, consider converting it to a TEXT data type column in a separate blobspace.

The column within the row page is only 56 bytes long, which allows more rows on a page than when you include a long string. However, the TEXT data type is not automatically compatible with existing programs. The application needed to fetch a TEXT value is a bit more complicated than the code for fetching a CHAR value into a program.

Move strings to a companion table

Strings that are less than half a page waste disk space if you treat them as TEXT data, but you can move them from the main table to a companion table.

If you split a table into two tables, the primary table and a companion table, repeat the primary key in each table.

Build a symbol table

If a column contains strings that are not unique in each row, you can move those strings to a table in which only unique copies are stored.

For example, the **customer.city** column contains city names. Some city names are repeated in the column, and most rows have some trailing blanks in the field. Using the VARCHAR data type eliminates the blanks but not the duplication.

You can create a table named **cities**, as the following example shows:

```
CREATE TABLE cities (
   city_num SERIAL PRIMARY KEY,
   city_name VARCHAR(40) UNIQUE
)
```

You can change the definition of the **customer** table so that its **city** column becomes a foreign key that references the **city_num** column in the **cities** table.

To insert the city of the new customer into **cities**, you must change any program that inserts a new row into **customer.** The database server return code in the **SQLCODE** field of the SQL Communications Area (SQLCA) can indicate that the insert failed because of a duplicate key. It is not a logical error; it simply means that an existing customer is located in that city. For more information about the SQLCA, see the *IBM® Informix® Guide to SQL: Tutorial*.

Besides changing programs that insert data, you must also change all programs and stored queries that retrieve the city name. The programs and stored queries must use a join to the new **cities** table in order to obtain their data. The extra complexity in programs that insert rows and the extra complexity in some queries is the result of giving up theoretical correctness in the data model. Before you make the change, be sure that it returns a reasonable savings in disk space or execution time.

Splitting wide tables

Consider all the attributes of an entity that has rows that are too wide for good performance. Look for some theme or principle to divide them into two groups. Then split the table into two tables, a primary table and a companion table, repeating the primary key in each one.

The shorter rows allow you to query or update each table quickly.

Division by Bulk

One principle on which you can divide an entity table is bulk. Move the bulky attributes, which are usually character strings, to the companion table. Keep the numeric and other small attributes in the primary table. In the demonstration database, you can split the **ship_instruct** column from the **orders** table. You can call the companion table **orders_ship**. It has two columns, a primary key that is a copy of **orders.order_num** and the original **ship_instruct** column.

Division by Frequency of Use

Another principle for division of an entity is frequency of use. If a few attributes are rarely queried, move them to a companion table. In the demonstration database, for example, perhaps only one program queries the **ship_instruct**, **ship_weight**, and **ship_charge** columns. In that case, you can move them to a companion table.

Division by Frequency of Update

Updates take longer than queries, and updating programs lock index pages and rows of data during the update process, preventing querying programs from accessing the tables. If you can separate one table into two companion tables, one with the most-updated entities and the other with the most-queried entities, you can often improve overall response time.

Performance Costs of Splitting Tables

Splitting a table uses extra disk space and adds complexity. Two copies of the primary key occur for each row, one copy in each table. Two primary-key indexes also exist. You can use the methods described in earlier sections to estimate the number of added pages.

You must modify existing programs, reports, and forms that use SELECT * because fewer columns are returned. Programs, reports, and forms that use attributes from both tables must perform a join to bring the tables together.

In this case, when you insert or delete a row, two tables are altered instead of one. If you do not coordinate the alteration of the two tables (by making them within a single transaction, for example), you lose semantic integrity.

Redundant data

Normalized tables contain no redundant data. Every attribute appears in only one table.

Normalized tables also contain no derived data. Instead, data that can be computed from existing attributes is selected as an expression based on those attributes.

Normalizing tables minimizes the amount of disk space used and makes updating the tables as easy as possible. However, normalized tables can force you to use joins and aggregate functions often, and those processes can be time consuming.

As an alternative, you can introduce new columns that contain redundant data, provided you understand the trade-offs involved.

- Adding redundant data
  A correct data model avoids redundancy by keeping any attribute only in the table for the entity that it describes. If the attribute data is needed in a different context, you join tables to make the connection. But joining takes time. If a frequently used join affects performance, you can eliminate it by duplicating the joined data in another table.

Adding redundant data

A correct data model avoids redundancy by keeping any attribute only in the table for the entity that it describes. If the attribute data is needed in a different context, you join tables to make the connection. But joining takes time. If a frequently used join affects performance, you can eliminate it by duplicating the joined data in another table.

In the **stores_demo** database, the **manufact** table contains the names of manufacturers and their delivery times. An actual working database might contain many other attributes of a supplier, such as address and sales representative name.

The contents of **manufact** are primarily a supplement to the **stock** table. Suppose that a time-critical application frequently refers to the delivery lead time of a particular product but to no other column of **manufact**. For each such reference, the database server must read two or three pages of data to perform the lookup.

You can add a new column, **lead_time**, to the **stock** table and fill it with copies of the **lead_time** column from the corresponding rows of **manufact**. That arrangement eliminates the lookup and therefore speeds up the application.

Like derived data, redundant data takes space and poses an integrity risk. In the example described in the previous paragraph, many extra copies of the lead time for each manufacturer can exist. (Each manufacturer can appear in **stock** many times.) The programs that insert or update a row of **manufact** must also update multiple rows of **stock**.

The integrity risk is simply that the redundant copies of the data might not be accurate. If a lead time is changed in **manufact**, the **stock** column is outdated until it is also updated. As you do with derived data, define the conditions under which redundant data might be wrong.

For more information about database design, see the *IBM® Informix® Database Design and Implementation Guide*.

Reduce disk space in tables with variable length rows

You can enable the database server to insert more rows per page into tables with variable-length rows, if you set the MAX_FILL_DATA_PAGES configuration parameter to `1`. Allowing more variable length rows per page has advantages and disadvantages.

Potential advantages of allowing more variable length rows per page are:

- Reducing the disk space required to store data
- Enabling the server to use the buffer pool more efficiently
- Reducing table scan times

Possible disadvantages of using the MAX_FILL_DATA_PAGES allowing more variable length rows per page are:

- The server might store rows in a different physical order.
- As the page fills, updates made to the variable-length columns in a row could cause the row to expand so it no longer completely fits on the page. This causes the server to split the row onto two pages, increasing the access time for the row.

If the MAX_FILL_DATA_PAGES configuration parameter is enabled, the server will add a new row to a recently modified page with existing rows if adding the row leaves at least 10 percent of the page free for future expansion of all the rows in the page. If the MAX_FILL_DATA_PAGES configuration parameter is not enabled, the server will add the row only if there is sufficient room on the page to allow the new row to grow to its maximum length.

If you enable the MAX_FILL_DATA_PAGES configuration parameter and you want this to affect existing variable length rows, the existing tables must be reloaded.

Reduce disk space by compressing tables and fragments

You can reduce disk space by compressing data in tables and table fragments. After compressing data, you can repack the data to consolidate the free space in a table or fragment, and shrink the space for the data to return the free space to the dbspace.

Compression is advantageous for applications with a lot of I/O activity and for applications in which the reduction of disk space usage is critical. However, if your applications run with high buffer cache hit ratios and high performance is more important than space usage, you might not want to compress data, because compression might slightly decrease performance.

Compressing data, consolidating data, and returning free space have the following benefits:

- Significant savings in disk storage space
- Reduced disk usage for compressed fragments
- Significant saving of logical log usage, which saves additional space and can prevent bottlenecks for high-throughput OLTP after the compression operation is completed.
- Fewer page reads, because more rows can fit on a page
- Smaller buffer pools, because more data fits in the same size pool
- Reduced I/O activity, because:
  - More compressed rows than uncompressed rows fit on a page
  - Log records for insert, update, and delete operations of compressed rows are smaller
- Ability to compress older fragments of time-fragmented data that are not often accessed, while leaving more recent data that is frequently accessed in uncompressed form
- Ability to free space no longer needed for a table
- Faster backup and restore

Because compressed data covers fewer pages and has more rows per page than uncompressed data, the query optimizer might choose different plans after compression.

You can speed up compression and repacking by running the operations in parallel.

Related information:
Compression
table or fragment arguments: Compress data and optimize storage (SQL administration API)

Boosted Partition Free Space Caches (PFSC)

When MAX_FILL_DATA_PAGES is set, a small cache is automatically created in memory for each table with variable-length rows. The purpose of this "lite" PFSC is to track exactly how much free space exists on up to 32 pages from which rows have recently been deleted. This information can be searched very quickly when inserting a row. If a spot for the new row is found on a page being tracked by the cache, the row is inserted on that page. If not, the table's bitmaps will have to be consulted, which may be less efficient in some cases.

A *boosted* PFSC is a cache that tracks *all free space* in the table or fragment. All inserts to this table use this larger cache in lieu of the bitmaps. This feature can significantly improve insert performance, though additional memory is required.

Consider creating a boosted PFSC for a table under the following conditions:

- MAX_FILL_DATA_PAGES = 1
- The table is either compressed or its schema contains variable-length columns
- The table is relatively large
- The table is relatively volatile, with deletes affecting rows of all ages

Indexes and index performance considerations

Informix® provides several types of indexes. Some performance issues are associated with indexes.

- Types of indexes
  Informix uses B-tree indexes, R-tree indexes, functional indexes, and indexes that DataBlade modules provide for user-defined data. The server also uses forest of trees (FOT) indexes, which are alternatives to B-tree indexes.
- Estimating index pages
  The index pages associated with a table can add significantly to the size of a dbspace.
- Managing indexes
  An index on the appropriate column can save thousands, tens of thousands, or in extreme cases, even millions of disk operations during a query. However, indexes entail costs.
- Improve query performance with a forest of trees index
  A forest of trees index is an alternate indexing method that alleviates the performance bottlenecks and root node contention that can occur when many concurrent users access a traditional B-tree index.
- Creating and dropping an index in an online environment
  You can use the CREATE INDEX ONLINE and DROP INDEX ONLINE statements to create and drop an index in an online environment, when the database and its associated tables are continuously available.
- Improving performance for index builds
  You can improve performance for index builds by adjusting the PDQ priority and by allocating enough memory and temporary space for the entire index.
- Storing multiple index fragments in a single dbspace
  You can store multiple fragments of the same index in a single dbspace, reducing the total number of dbspaces needed for a fragmented table. You must specify a name for each fragment that you want to store in the same dbspace. Storing multiple index fragments in a single dbspace simplifies the management of dbspaces.
- Improving performance for index checks
  The **oncheck** utility provides better concurrency for tables that use row locking. When a table uses page locking, **oncheck** places a shared lock on the table when it performs index checks. Shared locks do not allow other users to perform updates, inserts, or deletes on the table while **oncheck** checks or prints the index information.
- Indexes on user-defined data types
  You can define your own data types and the functions that operate on these data types. You can define indexes on some kinds of user-defined data types.

Types of indexes

Informix® uses B-tree indexes, R-tree indexes, functional indexes, and indexes that DataBlade modules provide for user-defined data. The server also uses forest of trees (FOT) indexes, which are alternatives to B-tree indexes.

- B-tree indexes
  Informix uses a B-tree index for columns that contain built-in data types (referred to as a *traditional B-tree index*), columns that contain one-dimensional user-defined data types (referred to as a *generic B-tree index*), and values that a user-defined data type returns.
- Forest of trees indexes
  A forest of trees index is like a B-tree index, but it has multiple root nodes and potentially fewer levels. Multiple root nodes can alleviate root node contention, because more concurrent users can access the index. A forest of trees index can also improve the performance of a query by reducing the number of levels involved in buffer read operations.
- R-tree indexes
  Informix uses an R-tree index for spatial data (such as two-dimensional or three-dimensional data).
- Indexes that DataBlade modules provide
  DataBlade modules can contain user-defined data types. A DataBlade module can also provide a user-defined index for the new data type.

Related concepts:
Estimating index pages
Managing indexes
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Improving performance for index checks
Indexes on user-defined data types
What is a functional index?
Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

B-tree indexes

Informix® uses a B-tree index for columns that contain built-in data types (referred to as a *traditional B-tree index*), columns that contain one-dimensional user-defined data types (referred to as a *generic B-tree index*), and values that a user-defined data type returns.

Built-in data types include character, datetime, integer, float, and so forth. For more information about built-in data types, see *IBM® Informix Guide to SQL: Reference*.

User-defined data types include opaque and distinct data types. For more information about user-defined data types, see *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

The return value of a user-defined function can be a built-in or user-defined data type, but not a simple large object (TEXT or BYTE data type) or a smart large object (BLOB or CLOB data type). For more information about how to use functional indexes, see Using a functional index.

For information about how to estimate B-tree index size, see Estimating index pages.

- Structure of conventional index pages
  A conventional index is arranged as a hierarchy of pages (technically, a *B-tree*).

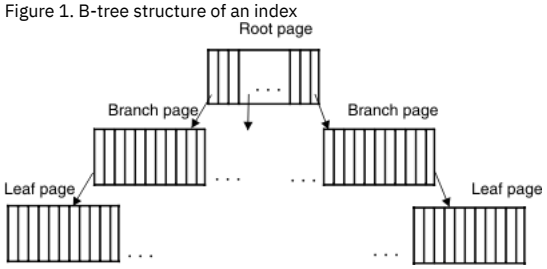
Related concepts:
Forest of trees indexes
R-tree indexes
Indexes that DataBlade modules provide

Structure of conventional index pages

A conventional index is arranged as a hierarchy of pages (technically, a *B-tree*).

The following figure shows the B-tree structure of an index. The topmost level of the hierarchy contains a single *root page*. Intermediate levels, when needed, contain *branch pages*. Each branch page contains entries that see a subset of pages in the next level of the index. The bottom level of the index contains a set of *leaf pages*. Each leaf page contains a list of index entries that see rows in the table.

Figure 1. B-tree structure of an index



The number of levels needed to hold an index depends on the number of unique keys in the index and the number of index entries that each page can hold. The number of entries per page depends, in turn, on the size of the columns being indexed.

If the index page for a given table can hold 100 keys, a table of up to 100 rows requires a single index level: the root page. When this table grows beyond 100 rows, to a size between 101 and 10,000 rows, it requires a two-level index: a root page and between 2 and 100 leaf pages. When the table grows beyond 10,000 rows, to a size between 10,001 and 1,000,000 rows, it requires a three-level index: the root page, a set of 100 branch pages, and a set of up to 10,000 leaf pages.

Index entries contained within leaf pages are sorted in key-value order. An index entry consists of a *key* and one or more *row pointers*. The key is a copy of the indexed columns from one row of data. A row pointer provides an address used to locate a row that contains the key. A unique index contains one index entry for every row in the table.

For information about special indexes for Informix®, see Indexes on user-defined data types.

Related concepts:
Forest of trees indexes

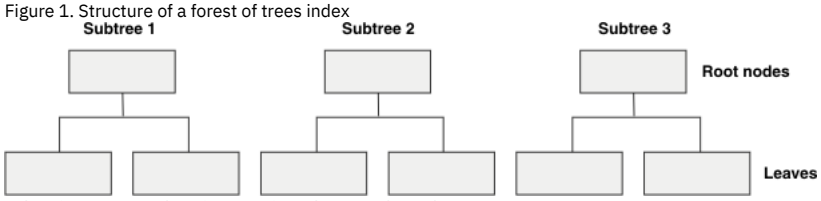
Forest of trees indexes

A forest of trees index is like a B-tree index, but it has multiple root nodes and potentially fewer levels. Multiple root nodes can alleviate root node contention, because more concurrent users can access the index. A forest of trees index can also improve the performance of a query by reducing the number of levels involved in buffer read operations.

You can create a forest of trees index as an alternative to a B-Tree index, but not as an alternative to an R-Tree index or other types of indexes.

Unlike a traditional B-tree index, which contains one root node, a forest of trees index is a large B-Tree index that is divided into smaller subtrees (which you can think of as buckets). These subtrees contain multiple root nodes and leaves. The following figure shows the structure of a forest of trees index.

Figure 1. Structure of a forest of trees index



Informix® stores and retrieves an item from a subtree by:

1. Computing a hash value from the columns that you selected when creating the index.
2. Mapping the hash value to a subtree for storage or retrieval of the row.

Forest of trees indexes are detached indexes. The server does not support forest of trees attached indexes.

You create a forest of trees index with the CREATE INDEX statement of SQL and the HASH ON clause.

You enable or disable forest of trees indexes with the SET INDEXES statement of SQL.

You can identify a forest of trees index by the `FOT` indicator in the `Index Name` field in SET EXPLAIN output.

You can look up the number of hashed columns and subtrees in a forest of trees index by viewing information in the **sysindices** table for the database containing tables that have forest of trees indexes.

The server treats a forest of trees index the same way it treats a B-tree index. Therefore, in a logged database, you can control how the B-tree scanner threads remove deletions from both forest of trees and B-tree indexes.

Restrictions: You cannot:

- Create forest of trees indexes on columns with complex data types, UDTs, or functional columns.
- Use the FILLFACTOR option of the CREATE INDEX statement when you create forest of trees indexes, because the indexes are built from top to bottom.
- Create clustered forest of trees indexes.
- Run the ALTER INDEX statement on forest of trees indexes.
- Run the SET INDEXES statement on forest of trees indexes in a database of secondary servers within a cluster environment.
- Use forest of trees indexes in queries that use aggregates, including minimum and maximum range values.
- Perform range scans directly on the HASH ON columns of a forest of trees index.
  However, you can perform range scans on columns that are not listed in the HASH ON column list. For range scans on columns listed in HASH ON column list, you must create an additional B-tree index that contains the appropriate column list for the range scan. This additional B-tree index might have the same column list as the forest of trees index, plus or minus a column.

- Use a forest of trees index for an `OR` index path. The database server does not use forest of trees indexes for queries that have an `OR` predicate on the indexed columns.

Related concepts:
B-tree indexes
R-tree indexes
Indexes that DataBlade modules provide
Structure of conventional index pages
Related tasks:
Improve query performance with a forest of trees index
Detecting root node contention
Creating a forest of trees index

Disabling and enabling a forest of trees index
Determining if you are using a forest of trees index
Related information:
CREATE INDEX statement
HASH ON clause

R-tree indexes

Informix® uses an R-tree index for spatial data (such as two-dimensional or three-dimensional data).

For information about sizing an R-tree index, see the *IBM® Informix R-Tree Index User's Guide*.

Related concepts:
B-tree indexes
Forest of trees indexes
Indexes that DataBlade modules provide

Indexes that DataBlade modules provide

DataBlade modules can contain user-defined data types. A DataBlade module can also provide a user-defined index for the new data type.

For example, the Excalibur Text Search DataBlade provides an index to search text data. For more information, see the Informix® Excalibur Text Search DataBlade.

For more information about the types of data and functions that each DataBlade module provides, see the user guide of each DataBlade module. For information about how to determine the types of indexes available in your database, see Identifying the available access methods.

Related concepts:
B-tree indexes
Forest of trees indexes
R-tree indexes

Estimating index pages

The index pages associated with a table can add significantly to the size of a dbspace.

By default, the database server creates the index in the same dbspace as the table, but in a separate tblspace from the table. To place the index in a separate dbspace, specify the IN keyword in the CREATE INDEX statement.

Although you cannot explicitly specify the extent size of an index, you can estimate the number of pages that an index might occupy to determine if your dbspace or dbspaces have enough space allocated.

- Index extent sizes
  The database server determines the extent size of an index based on the extent size for the corresponding table, regardless of whether the index is fragmented or not fragmented.
- Estimating conventional index pages
  You can estimate the size of index pages, using a series of formulas.

Related concepts:
Types of indexes
Managing indexes
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Improving performance for index checks
Indexes on user-defined data types
Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

Index extent sizes

The database server determines the extent size of an index based on the extent size for the corresponding table, regardless of whether the index is fragmented or not fragmented.

- Formula for estimating the extent size of an attached index
  For an attached index, the database server uses the ratio of the index key size to the row size to assign an appropriate extent size for the index.
- Formula for estimating the extent size of a detached index
  For a detached index, the database server uses the ratio of the index key size plus some overhead bytes to the row size to assign an appropriate extent size for the index.

Formula for estimating the extent size of an attached index

For an attached index, the database server uses the ratio of the index key size to the row size to assign an appropriate extent size for the index.

The following formula shows how the database server uses the ratio of the index key size to the row size:

```
Index extent size = (index_key_size /
table_row_size) *
   table_extent_size
```

In this formula:

- *index_key_size* is the total widths of the indexed column or columns plus 5 for a key descriptor.
- *table_row_size* is the sum of all the columns in the row.
- *table_extent_size* is the value that you specify in the EXTENT SIZE keyword of the CREATE TABLE statement.

If the index is not unique, then the extent size is reduced by 20 percent.

The database server also uses this same ratio for the next-extent size for the index:

```
Index next extent size =
(index_key_size/table_row_size)*
   table_next_extent_size
```


Formula for estimating the extent size of a detached index

For a detached index, the database server uses the ratio of the index key size plus some overhead bytes to the row size to assign an appropriate extent size for the index.

The following formula shows how the database server uses the ratio of the index key size plus some overhead bytes to the row size:

```
Detached Index extent size = ( (index_key_size +
9) /                              table_row_size) *
 table_extent_size
```

For example, suppose you have the following values:

```
index_key_size = 8 bytes
table_row_size = 33 bytes
table_extent_size = 150 * 2-kilobyte page
```

The above formula calculates the extent size as follows:

```
Detached Index extent size = ( (8 + 9) /
33) * 150 * 2-kilobyte page
                           = (17/33) * 300 kilobytes
                           = 154 kilobytes
```

Important: For a non-unique index, the formula calculates an extent size that is reduced by 20 percent.

Estimating conventional index pages

You can estimate the size of index pages, using a series of formulas.

To estimate the number of index pages:

1. Add up the total widths of the indexed column or columns.
   This value is referred to as colsize. Add 4 to *colsize* to obtain *keysize*, the actual size of a key in the index. For example, if *colsize* is 6, the value of *keysize* is 10.
2. Calculate the expected proportion of unique entries to the total number of rows.
   The formulas in subsequent steps see this value as *propunique*.

   If the index is unique or has few duplicate values, use 1 for *propunique*.

   If a significant proportion of entries are duplicates, divide the number of unique index entries by the number of rows in the table to obtain a fractional value for *propunique*. For example, if the number of rows in the table is 4,000,000 and the number of unique index entries is 1,000,000, the value of *propunique* is .25.

   If the resulting value for *propunique* is less than .01, use .01 in the calculations that follow.

3. Estimate the size of a typical index entry with one of the following formulas, depending on whether the table is fragmented or not:
   a. For nonfragmented tables, use the following formula:

      ```
      entrysize = (keysize * propunique) + 5 + 4
      ```

      The value 5 represents the number of bytes for the row pointer in a nonfragmented table.

      For nonunique indexes, the database server stores the row pointer for each row in the index node but stores the key value only once. The `entrysize` value represents the average length of each index entry, even though some entries consist of only the row pointer.

      For example, if *propunique* is .25, the average number of rows for each unique key value is 4. If *keysize* is 10, the value of *entrysize* is 11.5, calculated as (10 * 0.25) + 5 + 4 = 2.5 + 9 = 11.5. The following calculation shows the space required for all four rows:

      ```
      space for four rows = 4 * 11.5 = 46
      ```

      This space requirement is the same when you calculate it for the key value and add the four row pointers, as the following formula shows:

      ```
      space for four rows = 10 + (4 * 9) = 46
      ```

   b. For fragmented tables, use the following formula:

      ```
      entrysize = (keysize * propunique) + 9 + 4
      ```

      The value 9 represents the number of bytes for the row pointer in a fragmented table.

4. Estimate the number of entries per index page with the following formula:

   ```
   pagents = trunc(pagefree/entrysize)
   ```

   In this formula:

   - *pagefree* is the page size minus the page header (2020 for a 2-kilobyte page size).
   - *entrysize* is the size of a typical index entry, which you estimated in the previous step.

   The **trunc()** function notation indicates that you should round down to the nearest integer value.

5. Estimate the number of leaf pages with the following formula:

   ```
   leaves = ceiling(rows/pagents)
   ```

   In this formula:

   - *rows* is the number of rows that you expect to be in the table.
   - *pagents* is the number of entries per index page, which you estimated in the previous step.

   The **ceiling()** function notation indicates that you should round up to the nearest integer value.

6. Estimate the number of branch pages at the second level of the index with the following formula:

   $$branches_0 = ceiling(leaves/node\_ents)$$

   Calculate the value for *node_ents* with the following formula:

   ```
   node_ents = trunc( pagefree / ( keysize + 4) + 4)
   ```

   In this formula:

   - *pagefree* is the page size minus the page header (2020 for a 2-kilobyte page size).
   - `keysize` is the *colsize* plus 4. You obtained this value in step 1.

   In the formula, 4 represents the number of bytes for the leaf node pointer.

7. If the value of $branches_0$ is greater than 1, more levels remain in the index.
   To calculate the number of pages contained in the next level of the index, use the following formula:

   $$branches_{n+1} = ceiling(branches_n/node\_ents)$$

   In this formula:

   - $branches_n$ is the number of branches for the last index level that you calculated.
   - $branches_{n+1}$ is the number of branches in the next level.
   - *node_ents* is the value that you calculated in step 6.

8. Repeat the calculation in step 7 for each level of the index until the value of $branches_{n+1}$ equals 1.
9. Add up the total number of pages for all branch levels calculated in steps 6 through 8. This sum is called *branchtotal*.
10. Use the following formula to calculate the number of pages in the compact index:

    ```
    compactpages = (leaves + branchtotal)
    ```

11. If your database server instance uses a fill factor for indexes, the size of the index increases.
    The default fill factor value is 90 percent. You can change the fill factor value for all indexes with the FILLFACTOR configuration parameter. You can also change the fill factor for an individual index with the FILLFACTOR clause of the CREATE INDEX statement in SQL.

    To incorporate the fill factor into your estimate for index pages, use the following formula:

    ```
    indexpages = 100 * compactpages / FILLFACTOR
    ```

The preceding estimate is a guideline only. As rows are deleted and new ones are inserted, the number of index entries can vary within a page. This method for estimating index pages yields a conservative (high) estimate for most indexes. For a more precise value, build a large test index with real data and check its size with the **oncheck** utility.

Tip: A forest of trees index can be larger than a B-Tree index. When you estimate the size of a forest of trees index, the estimates apply to each subtree in the index. Then, you must aggregate the buckets to calculate the total estimation.

Managing indexes

An index on the appropriate column can save thousands, tens of thousands, or in extreme cases, even millions of disk operations during a query. However, indexes entail costs.

An index is necessary on any column or combination of columns that must be unique. However, as discussed in Queries and the query optimizer, the presence of an index can also allow the query optimizer to speed up a query.

The optimizer can use an index in the following ways:

- To replace repeated sequential scans of a table with nonsequential access
- To avoid reading row data when processing expressions that name only indexed columns
- To avoid a sort (including building a temporary table) when executing the GROUP BY and ORDER BY clauses

- Space costs of indexes
  The first cost of an index is disk space. The presence of an index can add many pages to a dbspace; it is easy to have as many index pages as row pages in an indexed table. Additionally, in an environment where multiple languages are used, indexes created for each language require additional disk space.
- Time costs of indexes
  The second cost of an index is time whenever the table is modified.
- Unclaimed index space
  A background thread, the B-tree scanner, identifies an index with the most unclaimed index space. Unclaimed index space degrades performance and causes extra work for the server. When an index is chosen for scanning, the entire leaf of the index is scanned for deleted (dirty) items that were committed, but not yet removed from the index. The B-tree scanner removes these items when necessary.
- Indexes on columns
  You can create an index for one or more columns in a table. Indexes are required on columns that must be unique and are not specified as primary keys.
- Nonunique indexes
  In some applications, most table updates can be confined to a single time period. You might be able to set up your system so that all updates are applied overnight or on specified dates. Additionally, when updates are performed as a batch, you can drop all nonunique indexes while you make updates and then create new indexes afterward. This strategy can improve performance.

Related concepts:
Types of indexes
Estimating index pages
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Improving performance for index checks
Indexes on user-defined data types
Using a functional index
Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

Space costs of indexes

The first cost of an index is disk space. The presence of an index can add many pages to a dbspace; it is easy to have as many index pages as row pages in an indexed table. Additionally, in an environment where multiple languages are used, indexes created for each language require additional disk space.

When you consider space costs, also consider whether increasing the page size of a standard or temporary dbspace is beneficial in your environment. If you want a longer key length than is available for the default page size, you can increase the page size. If you increase the page size, the size must be an integral multiple of the default page size, not greater than 16K bytes.

You might not want to increase the page size if your application contains small sized rows. Increasing the page size for an application that randomly accesses small rows might decrease performance. In addition, a page lock on a larger page will lock more rows, reducing concurrency in some situations.

You can save disk space by compressing detached B-tree indexes, consolidating free space in the index, and returning the free space to the dbspace.

Related information:
B-tree index compression

Time costs of indexes

The second cost of an index is time whenever the table is modified.

The following descriptions assume that approximately two pages must be read to locate an index entry. That is the case when the index consists of a root page, one level of branch pages, and a set of leaf pages. The root page is assumed to be in a buffer already. The index for a very large table has at least two intermediate levels, so about three pages are read when the database server references such an index.

Presumably, one index is used to locate a row being altered. The pages for that index might be found in page buffers in shared memory for the database server. However, the pages for any other indexes that need altering must be read from disk.

Under these assumptions, index maintenance adds time to different kinds of modifications, as the following list shows:

- When you delete a row from a table, the database server must delete its entries from all indexes.
  The database server must look up the entry for the deleted row (two or three pages in) and rewrite the leaf page. The write operation to update the index is performed in memory, and the leaf page is flushed when the least recently used (LRU) buffer that contains the modified page is cleaned. This operation requires two or three page accesses to read the index pages if needed and one deferred page access to write the modified page.
- When you insert a row, the database server must insert its entries in all indexes.
  The database server must find a place in which to enter the inserted row within each index (two or three pages in) and rewrite (one deferred page out), for a total of three or four immediate page accesses per index.
- When you update a row, the database server must look up its entries in each index that applies to an altered column (two or three pages in).
  The database server must rewrite the leaf page to eliminate the old entry (one deferred page out) and then locate the new column value in the same index (two or three more pages in) and the row entered (one more deferred page out).

Insertions and deletions change the number of entries on a leaf page. Although virtually every *pagents* operation requires some additional work to deal with a leaf page that has either filled or been emptied, if *pagents* is greater than 100, this additional work occurs less than 1 percent of the time. You can often disregard it when you estimate the I/O impact.

In short, when a row is inserted or deleted at random, allow three to four added page I/O operations per index. When a row is updated, allow six to eight page I/O operations for each index that applies to an altered column. If a transaction is rolled back, all this work must be undone. For this reason, rolling back a transaction can take a long time.

Because the alteration of the row itself requires only two page I/O operations, index maintenance is clearly the most time-consuming part of data modification. For information about one way to reduce this cost, see Clustering.

Unclaimed index space

A background thread, the B-tree scanner, identifies an index with the most unclaimed index space. Unclaimed index space degrades performance and causes extra work for the server. When an index is chosen for scanning, the entire leaf of the index is scanned for deleted (dirty) items that were committed, but not yet removed from the index. The B-tree scanner removes these items when necessary.

The B-tree scanner allows multiple threads.

Use the BTSCANNER configuration parameter to specify the number of B-tree scanner threads to start and the priority of the B-tree scanner threads when the database server starts. For details, see the *IBM® Informix® Administrator's Reference*.

You can invoke the B-tree scanner from the command line.

Indexes on columns

You can create an index for one or more columns in a table. Indexes are required on columns that must be unique and are not specified as primary keys.

In addition, you must add an index on columns that:

- Are used in joins that are not specified as foreign keys
- Are frequently used in filter expressions
- Are frequently used for ordering or grouping
- Do not involve duplicate keys
- Are amenable to clustered indexing

- Filtered columns in large tables
  If a column is often used to filter the rows of a large table, consider placing an index on it. The optimizer can use the index to select the wanted columns and avoid a sequential scan of the entire table.
- Order-by and group-by columns
  You can place an index on the ordering column or columns of a table. The database server then uses the index that to sort the query results in the most efficient manner.
- Avoiding columns with duplicate keys
  Duplicate keys in indexes can cause performance problems. You can take steps to avoid these problems.
- Clustering
  Clustering is a method for arranging the rows of a table so that their physical order on disk closely corresponds to the sequence of entries in the index.

Filtered columns in large tables

If a column is often used to filter the rows of a large table, consider placing an index on it. The optimizer can use the index to select the wanted columns and avoid a sequential scan of the entire table.

Suppose you have a table that contains a large mailing list. If you find that a postal-code column is often used to filter a subset of rows, consider putting an index on that column.

This strategy yields a net savings of time only when the selectivity of the column is high; that is, when only a small fraction of rows holds any one indexed value. Nonsequential access through an index takes several more disk I/O operations than sequential access does, so if a filter expression on the column passes more than a fourth of the rows, the database server might as well read the table sequentially.

As a rule, indexing a filter column saves time in the following cases:

- The column is used in filter expressions in many queries or in slow queries.
- The column contains at least 100 unique values.
- Most column values appear in fewer than 10 percent of the rows.

Order-by and group-by columns

You can place an index on the ordering column or columns of a table. The database server then uses the index that to sort the query results in the most efficient manner.

When a large quantity of rows must be ordered or grouped, the database server must put the rows in order. One way that the database server performs this task is to select all the rows into a temporary table and sort the table. But, as explained in Queries and the query optimizer, if the ordering columns are indexed, the optimizer sometimes reads the rows in sorted order through the index, thus avoiding a final sort.

Because the keys in an index are in sorted sequence, the index really represents the result of sorting the table. By placing an index on the ordering column or columns, you can replace many sorts during queries with a single sort when the index is created.

Avoiding columns with duplicate keys

Duplicate keys in indexes can cause performance problems. You can take steps to avoid these problems.

When duplicate keys are permitted in an index, entries that match a given key value are grouped in lists. The database server uses these lists to locate rows that match a requested key value. When the selectivity of the index column is high, these lists are generally short. But when only a few unique values occur, the lists become long and can cross multiple leaf pages.

Placing an index on a column that has low selectivity (that is, a small number of distinct values relative to the number of rows) can reduce performance. In such cases, the database server must not only search the entire set of rows that match the key value, but it must also lock all the affected data and index pages. This process can impede the performance of other update requests as well.

To correct this problem, replace the index on the low-selectivity column with a composite index that has a higher selectivity. Use the low-selectivity column as the leading column and a high-selectivity column as your second column in the index. The composite index limits the number of rows that the database server must search to locate and apply an update.

You can use any second column to disperse the key values as long as its value does not change, or changes at the same time as the real key. The shorter the second column the better, because its values are copied into the index and expand its size.

Clustering

Clustering is a method for arranging the rows of a table so that their physical order on disk closely corresponds to the sequence of entries in the index.

When you know that a table is ordered by a certain index, you can avoid sorting. You can also be sure that when the table is searched on that column, it is read effectively in sequential order, instead of nonsequentially. These points are covered in Queries and the query optimizer.
Tip: For information about eliminating interleaved extents by altering an index to cluster, see Creating or altering an index to cluster.
In the **stores_demo** database, the **orders** table has an index, **zip_ix**, on the postal-code column. The following statement causes the database server to put the rows of the **customer** table in descending order by postal code:

```
ALTER INDEX zip_ix TO CLUSTER
```

To cluster a table on a nonindexed column, you must create an index. The following statement reorders the **orders** table by order date:

```
CREATE CLUSTER INDEX o_date_ix ON orders (order_date ASC)
```

To reorder a table, the database server must copy the table. In the preceding example, the database server reads all the rows in the table and constructs an index. Then it reads the index entries in sequence. For each entry, it reads the matching row of the table and copies it to a new table. The rows of the new table are in the desired sequence. This new table replaces the old table.

Clustering is not preserved when you alter a table. When you insert new rows, they are stored physically at the end of the table, regardless of their contents. When you update rows and change the value of the clustering column, the rows are written back into their original location in the table.

Clustering can be restored after the order of rows is disturbed by ongoing updates. The following statement reorders the table to restore data rows to the index sequence:

```
ALTER INDEX o_date_ix TO CLUSTER
```

Reclustering is usually quicker than the original clustering because reading out the rows of a nearly clustered table is similar in I/O impact to a sequential scan.

Clustering and reclustering take a lot of space and time. To avoid some clustering, build the table in the desired order initially.

- Configuration parameters that affect the degree of clustering
  The **clust** field in the **sysindexes** or the **sysindices** table represents the degree of clustering of the index. The values of several configuration parameters affect the **clust** field.

Related concepts:
Reclaiming space in an empty extent with ALTER INDEX

Configuration parameters that affect the degree of clustering

The **clust** field in the **sysindexes** or the **sysindices** table represents the degree of clustering of the index. The values of several configuration parameters affect the **clust** field.

The value of this field is affected by:

- The size of the buffer pool as specified by the BUFFERPOOL configuration parameter
- The DS_MAX_QUERIES configuration parameter, which specifies the maximum number of PDQ queries that can run concurrently

Each of these configuration parameters affects the amount of buffer space available for a single user session. Additional buffers can result in better clustering (a smaller **clust** value in the **sysindexes** or **sysindices** tables).

You can create more buffers by performing one or both of the following tasks:

- Increasing the size of the buffer pool by updating the value of the BUFFERPOOL configuration parameter
- Decreasing the value of the DS_MAX_QUERIES configuration parameter

Related information:
BUFFERPOOL configuration parameter
DS_MAX_QUERIES configuration parameter

Nonunique indexes

In some applications, most table updates can be confined to a single time period. You might be able to set up your system so that all updates are applied overnight or on specified dates. Additionally, when updates are performed as a batch, you can drop all nonunique indexes while you make updates and then create new indexes afterward. This strategy can improve performance.

Dropping nonunique indexes can have the following positive effects:

- The updating program can run faster with fewer indexes to update. Often, the total time to drop the indexes, update without them, and re-create them is less than the time to update with the indexes in place. (For a discussion of the time cost of updating indexes, see Time costs of indexes.)
- Newly made indexes are more efficient. Frequent updates tend to dilute the index structure so that it contains many partly full leaf pages. This dilution reduces the effectiveness of an index and wastes disk space.

As a time-saving measure, make sure that a batch-updating program calls for rows in the sequence that the primary-key index defines. That sequence causes the pages of the primary-key index to be read in order and only one time each.

The presence of indexes also slows down the population of tables when you use the LOAD statement or the **dbload** utility. Loading a table that has no indexes is a quick process (little more than a disk-to-disk sequential copy), but updating indexes adds a great deal of overhead.

To avoid this overhead, you can:

1. Drop the table (if it exists).
2. Create the table without specifying any unique constraints.
3. Load all rows into the table.
4. Alter the table to apply the unique constraints.
5. Create the nonunique indexes.

If you cannot guarantee that the loaded data satisfies all unique constraints, you must create unique indexes before you load the rows. It saves time if the rows are presented in the correct sequence for at least one of the indexes. If you have a choice, make it the row with the largest key. This strategy minimizes the number of leaf pages that must be read and written.

Improve query performance with a forest of trees index

A forest of trees index is an alternate indexing method that alleviates the performance bottlenecks and root node contention that can occur when many concurrent users access a traditional B-tree index.

A forest of trees index differs from a B-tree index in that it has multiple root nodes and fewer levels. Multiple root nodes can alleviate root node contention, because more concurrent users can access the index.

If you know that a particular table has a deep tree, you can improve performance by creating a forest of trees index with fewer levels in the tree. For example, suppose you create an index where one of the columns is a 100 byte column containing character data. If you have a large number of rows in that table, the tree might contain six or seven levels. If you create a forest of trees index instead of a B-tree index, you can create more than one tree with four levels, so that every index traversal goes only four levels deep rather than seven levels deep.

- Detecting root node contention
  You can analyze the output of the **onstat -g spi** command to identify the performance bottlenecks that a forest of trees index can alleviate.
- Creating a forest of trees index
  You use the CREATE INDEX statement with the HASH ON clause to create a forest of trees index.
- Disabling and enabling a forest of trees index
  You can use the INDEXES DISABLED option of the SET Database Object Mode statement of SQL to disable a forest of trees index, if you want the server to stop updating the index and to stop using it during queries. After you are ready to put the index into production, you can use the INDEXES ENABLED option to re-enable it.
- Performing a range scan on a forest of trees index
  While you cannot perform range scans directly on the HASH ON columns of a forest of trees index, you can perform range scans on the columns that are not listed in the HASH ON column list. To perform range scans on columns that are listed in HASH ON column list, you must create an additional B-tree index that contains the appropriate column list for the range scan.
- Determining if you are using a forest of trees index
  You can determine whether an index is a forest of trees index by viewing SET EXPLAIN output. A forest of trees index has `FOT` in the `Index Name` field of the output.
- Finding the number of hashed columns and subtrees in a forest of trees index
  You can look up the number of hashed columns and subtrees in a forest of trees index by viewing information in the **sysindices** table for the database containing tables that have forest of trees indexes.

Related concepts:
Types of indexes
Estimating index pages
Managing indexes
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Improving performance for index checks
Indexes on user-defined data types
Forest of trees indexes
Related tasks:
Improving performance for index builds

Detecting root node contention

You can analyze the output of the **onstat -g spi** command to identify the performance bottlenecks that a forest of trees index can alleviate.

To detect root node contention and determine whether you need a forest of trees index:

1. Run the **onstat -g spi | sort -nr** command to display information about spin locks with long spins.
   The output of the **onstat -g spi** command shows spin locks with waits, which occur when threads are reading from or writing to an index concurrently and a particular thread did not succeed in acquiring the lock on the first try.

2. Analyze the **onstat -g spi**output. Look for loop and wait information in these columns:
   - `Num Waits`: The Total number of times a thread waited for the spin lock.
   - `Num Loops`: The total number of attempts before a thread successfully acquired the spin lock.
   - `Avg Loop/Wait`: The average number of attempts to acquire the spin lock, computed as `Num Loops` / `Num Waits`.

   For example, the following output snippet shows spin locks with large numbers of waits and loops:

```
Spin locks with waits:
Num Waits Num Loops Avg Loop/Wait Name
332480    1568908   4.72  fast mutex, 3:bf[1234] 0x2d00008  0x1028a0d8000
39722     498769    12.56 mutex lock, name = log
20761     101831    4.90  fast mutex, 7:bf[62] 0x1300003  0x109da128000
14818     77680     5.24  mutex lock, name = MGM mutex
6523      34350     5.27  fast mutex, 3:bf[362] 0x20008e   0x10289a08000
```

3. Query **sysmaster:systabnames** with the hexadecimal representation of the part number shown in the **onstat -g spi** output. If the `tabname` represents an index name, the index is a forest of trees candidate.
   For example, run this query:

```
echo "select tabname, hex(partnum) from systabnames
 where hex(partnum) =  '0x02d00008'" | dbaccess sysmaster -

tabname       daily_market_idx
(expression)  0x02d00008

$ echo 'select tabname, hex(partnum) from systabnames'
 where hex(partnum) = 0x01300003 | dbaccess sysmaster -

tabname       trade_history_idx
(expression)  0x01300003

$ echo 'select tabname, hex(partnum) from systabnames'
 where hex(partnum) = 0x0020008E | dbaccess sysmaster -
```

```
tabname       trade_request_idx2
(expression)  0x0020008E
```

Related concepts:
Forest of trees indexes
Related tasks:
Creating a forest of trees index
Disabling and enabling a forest of trees index
Performing a range scan on a forest of trees index
Determining if you are using a forest of trees index
Finding the number of hashed columns and subtrees in a forest of trees index
Related information:
onstat -g spi command: Print spin locks with long spins

Creating a forest of trees index

You use the CREATE INDEX statement with the HASH ON clause to create a forest of trees index.

Prerequisite: Determine whether you need a forest of trees index to reduce performance bottlenecks and contention or to reduce the number of levels in a traditional B-Tree index.

To create a forest of trees index:

1. Choose the columns for the index and determine the number of subtrees to create.
2. Create the index by using the CREATE INDEX statement with the HASH ON clause:
   For example, the following command creates a forest of trees index with 100 subtrees (buckets) on the C1 column:

   ```
   CREATE INDEX fotidx ON tab(c1) hash on (c1) with 100 buckets
   ```

   After you create a forest of trees index, it is enabled.

You can monitor **onstat -g spi** command output to verify that root node contention no longer occurs. If you identify performance bottlenecks that are caused by highly contended spin locks, you can rebuild the forest of trees index with more buckets.

Related concepts:
Forest of trees indexes
Related tasks:
Detecting root node contention
Disabling and enabling a forest of trees index
Performing a range scan on a forest of trees index
Determining if you are using a forest of trees index
Finding the number of hashed columns and subtrees in a forest of trees index
Related information:
CREATE INDEX statement
HASH ON clause

Disabling and enabling a forest of trees index

You can use the INDEXES DISABLED option of the SET Database Object Mode statement of SQL to disable a forest of trees index, if you want the server to stop updating the index and to stop using it during queries. After you are ready to put the index into production, you can use the INDEXES ENABLED option to re-enable it.

To disable a forest of trees index:

Run the SET INDEXES DISABLED statement of SQL.
For example, for an index named `fotidx`, specify:

```
SET INDEXES fotidx DISABLED;
```

You can re-enable a disabled forest of trees index, for example, by specifying:

```
SET INDEXES fotidx ENABLED;
```

Related concepts:
Forest of trees indexes
Related tasks:
Detecting root node contention
Creating a forest of trees index
Performing a range scan on a forest of trees index
Determining if you are using a forest of trees index
Finding the number of hashed columns and subtrees in a forest of trees index

Performing a range scan on a forest of trees index

While you cannot perform range scans directly on the HASH ON columns of a forest of trees index, you can perform range scans on the columns that are not listed in the HASH ON column list. To perform range scans on columns that are listed in HASH ON column list, you must create an additional B-tree index that contains the appropriate column list for the range scan.

To create indexes for range scans:

1. Create a forest of trees index with at least one column that is not hashed.
   For example, specify:

   ```
   CREATE INDEX idx1 on tab(c1,c2) HASH ON (c1) with 100 buckets;
   ```

   You can perform a range scan directly on column `c2`, but not on column `c1`, which is listed in HASH ON column list.

2. For range scans on the columns listed in HASH ON column list, create an additional B-tree index that contains the appropriate column list for the range scan. This additional B-tree index might have the same column list as the forest of trees index, plus or minus a column.
   For example, specify:

   ```
   CREATE INDEX idx2 on tab(c1, c2, c3);
   ```

Related tasks:
Detecting root node contention
Creating a forest of trees index
Disabling and enabling a forest of trees index
Determining if you are using a forest of trees index
Finding the number of hashed columns and subtrees in a forest of trees index
Related information:
CREATE INDEX statement
HASH ON clause

Determining if you are using a forest of trees index

You can determine whether an index is a forest of trees index by viewing SET EXPLAIN output. A forest of trees index has `FOT` in the `Index Name` field of the output.

In the following example of partial SET EXPLAIN output, `informix.fot_idx` is the name of a forest of trees index.

```
Estimated Cost: 1
Estimated # of Rows Returned: 1

  1) informix.t: INDEX PATH

    (1) Index Name: informix.fot_idx (FOT)
        Index Keys: c1 c2   (Serial, fragments: ALL)
        Lower Index Filter: informix.t.c1 = 1
```

Related concepts:
Forest of trees indexes
Related tasks:
Detecting root node contention
Creating a forest of trees index
Disabling and enabling a forest of trees index
Performing a range scan on a forest of trees index
Finding the number of hashed columns and subtrees in a forest of trees index

Finding the number of hashed columns and subtrees in a forest of trees index

You can look up the number of hashed columns and subtrees in a forest of trees index by viewing information in the **sysindices** table for the database containing tables that have forest of trees indexes.

To view information about a forest of trees index:

1. Query the **sysindices** table for the index.
2. Go to the row containing the forest of trees index and view information in the `nhashcols` and `nbuckets` columns.

Related tasks:
Detecting root node contention
Creating a forest of trees index
Disabling and enabling a forest of trees index
Performing a range scan on a forest of trees index
Determining if you are using a forest of trees index

Creating and dropping an index in an online environment

You can use the CREATE INDEX ONLINE and DROP INDEX ONLINE statements to create and drop an index in an online environment, when the database and its associated tables are continuously available.

The CREATE INDEX ONLINE statement enables you to create an index without having an exclusive lock placed over the table during the duration of the index build. You can use the CREATE INDEX ONLINE statement even when reads or updates are occurring on the table. This means index creation can begin immediately.

When you create an index online, the database server logs the operation with a flag, so data recovery and restore operations can recreate the index.

When you create an index online, you can use the ONLIDX_MAXMEM configuration parameter to limit the amount of memory that is allocated to the *preimage* log pool and to the *updator* log pool in shared memory. You might want to do this if you plan to complete other operations on a table column while executing the CREATE INDEX ONLINE statement on the column. For more information about this parameter, see Limiting memory allocation while creating indexes online.

The DROP INDEX ONLINE statement enables you to drop indexes even when Dirty Read is the transaction isolation level.

The advantages of creating indexes using the CREATE INDEX ONLINE statement are:

- If a new index is needed to improve the performance of queries on a table, you can immediately create the index without a lock placed over the table.
- The database server can create an index while a table is being updated.
- The table is available for the duration of the index build.
- The query optimizer can establish better query plans, since the optimizer can update statistics in unlocked tables.

The advantages of dropping indexes using the DROP INDEX ONLINE statement are:

- You can drop an inefficient index without disturbing ongoing queries that are using that index.
- After the index is flagged, the query optimizer will not use the index for new SELECT operations on tables.

If you initiate a DROP INDEX ONLINE statement for a table that is being updated, the operation does not occur until after the table update is completed. After you issue the DROP INDEX ONLINE statement, no one can reference the index, but concurrent operations can use the index until the operations terminate. The database server waits to drop the index until all users have finished accessing the index.

An example of creating an index in an online environment is:

```
CREATE INDEX idx_1 ON table1(col1) ONLINE
```

An example of dropping an index in an online environment is:

```
DROP INDEX idx_1 ONLINE
```

For more information about the CREATE INDEX ONLINE and DROP INDEX ONLINE statements, see the *IBM® Informix Guide to SQL: Syntax*.

- When you cannot create or drop indexes online
  You cannot use the CREATE INDEX ONLINE and the DROP INDEX ONLINE statements under certain circumstances.
- Creating attached indexes in an online environment
  You can create attached indexes using the CREATE INDEX ONLINE statement, but the statement only operates when Dirty Read is the transaction isolation level.
- Limiting memory allocation while creating indexes online
  The ONLIDX_MAXMEM configuration parameter limits the amount of memory that is allocated to a single *preimage* pool and a single *updator* log pool.

Related concepts:
Types of indexes
Estimating index pages
Managing indexes
Storing multiple index fragments in a single dbspace
Improving performance for index checks
Indexes on user-defined data types
Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

When you cannot create or drop indexes online

You cannot use the CREATE INDEX ONLINE and the DROP INDEX ONLINE statements under certain circumstances.

You cannot use the CREATE INDEX ONLINE statement:

- To create an index at the same time that a table is being altered
- To create a clustered index
- To create a Virtual-Index Interface (VII) /R-tree index
- To create a functional index
- To create an index that is partitioned by an interval fragmentation strategy
- To create an index on a table that is partitioned by an interval fragmentation strategy

You cannot use the DROP INDEX ONLINE statement:

- To drop a Virtual-Index Interface (VII) /R-tree index
- To drop a clustered index

Creating attached indexes in an online environment

You can create attached indexes using the CREATE INDEX ONLINE statement, but the statement only operates when Dirty Read is the transaction isolation level.

The index creation takes an exclusive lock on the table and waits for all other concurrent processes scanning the table to quit using the index partitions before creating the attached index. If the table is being read or updated, the CREATE INDEX ONLINE statement waits for the exclusive lock for the duration of the lock mode setting.

Limiting memory allocation while creating indexes online

The ONLIDX_MAXMEM configuration parameter limits the amount of memory that is allocated to a single *preimage* pool and a single *updator* log pool.

The preimage and updator log pools, **pimage_<partnum>** and **ulog_<partnum>,** are shared memory pools that are created when a CREATE INDEX ONLINE statement is executed. The pools are freed when the execution of the statement is completed.

The default value of the ONLIDX_MAXMEM configuration parameter is `5120` kilobytes. The minimum value that you can specify is `16` kilobytes; the maximum value is `4294967295` kilobytes.

You can set the ONLIDX_MAXMEM configuration parameter before starting the database server, or you can change it dynamically through the **onmode -wf** and **onmode -wm** commands.

Improving performance for index builds

You can improve performance for index builds by adjusting the PDQ priority and by allocating enough memory and temporary space for the entire index.

Whenever possible, the database server uses parallel processing to improve the response time of index builds. The number of parallel processes is based on the number of fragments in the index and the value of the **PSORT_NPROCS** environment variable. The database server builds the index with parallel processing even when the value of PDQ priority is `0`.

You can often improve the performance of an index build by taking the following steps:

1. Set PDQ priority to a value greater than `0` to obtain more memory than the default 128 kilobytes.
   When you set PDQ priority to greater than `0`, the index build can take advantage of the additional memory for parallel processing.

   To set PDQ priority, use either the **PDQPRIORITY** environment variable or the SET PDQPRIORITY statement in SQL.
2. Do not set the **PSORT_NPROCS** environment variable. If you have a computer with multiple CPUs, the database server uses two threads per sort when it sorts index keys and **PSORT_NPROCS** is not set. The number of sorts depends on the number of fragments in the index, the number of keys, the key size, and the values of the PDQ memory configuration parameters.
3. Allocate enough memory and temporary space to build the entire index.
   a. Estimate the amount of virtual shared memory that the database server might need for sorting.
      For more information, see Estimating memory needed for sorting.

   b. Specify more memory with the DS_TOTAL_MEMORY and DS_MAX_QUERIES configuration parameters.
   c. If not enough memory is available, estimate the amount of temporary space needed for an entire index build.
      For more information, see Estimating temporary space for index builds.

   d. Use the **onspaces -t** utility to create large temporary dbspaces and specify them in the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable.
      For information about how to optimize temporary dbspaces, see Configure dbspaces for temporary tables and sort files.

- Estimating memory needed for sorting
  To calculate the amount of virtual shared memory that the database server might need for sorting, estimate the maximum number of sorts that might occur concurrently and multiply that number by the average number of rows and the average row size.
- Estimating temporary space for index builds
  You can estimate the number of bytes of temporary space needed for an entire index build.

Related concepts:
Types of indexes
Estimating index pages
Managing indexes
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Improving performance for index checks
Indexes on user-defined data types
Related tasks:
Improve query performance with a forest of trees index

Estimating memory needed for sorting

To calculate the amount of virtual shared memory that the database server might need for sorting, estimate the maximum number of sorts that might occur concurrently and multiply that number by the average number of rows and the average row size.

For example, if you estimate that 30 sorts could occur concurrently, the average row size is 200 bytes, and the average number of rows in a table is 400, you can estimate the amount of shared memory that the database server needs for sorting as follows:

```
30 sorts * 200 bytes * 400 rows = 2,400,000 bytes
```

You can use the DS_NONPDQ_QUERY_MEM configuration parameter to configure the amount sort memory available for non-PDQ queries.
Important: You can only use this parameter if the PDQ priority is set to zero. Its setting has no effect if the PDQ priority is greater than zero.
The minimum and default value of DS_NONPDQ_QUERY_MEM is 128 kilobytes. The maximum supported value is 25 percent of DS_TOTAL_MEMORY. For more information, see Configuring memory for queries with hash joins, aggregates, and other memory-intensive elements.

If the PDQ priority is greater than 0, the maximum amount of shared memory that the database server allocates for a sort is controlled by the memory grant manager (MGM). The MGM uses the settings of PDQ priority and the following configuration parameters to determine how much memory to grant for the sort:

- DS_TOTAL_MEMORY
- DS_MAX_QUERIES
- MAX_PDQPRIORITY

For more information about allocating memory for parallel processing, see The allocation of resources for parallel database queries.

Estimating temporary space for index builds

You can estimate the number of bytes of temporary space needed for an entire index build.

To estimate the amount of temporary space needed for an index build, perform the following steps:

1. Add the total widths of the indexed columns or returned values from user-defined functions. This value is referred to as *colsize*.
2. Estimate the size of a typical item to sort with one of the following formulas, depending on whether the index is attached or not:
   a. For a nonfragmented table and a fragmented table with an index created without an explicit fragmentation strategy, use the following formula:

      ```
      sizeof_sort_item = keysize + 4
      ```

   b. For fragmented tables with the index explicitly fragmented, use the following formula:

      ```
      sizeof_sort_item =

      keysize + 8
      ```

3. Estimate the number of bytes needed to sort with the following formula:

   ```
   temp_bytes = 2 * (rows *
   sizeof_sort_item)
   ```

   This formula uses the factor 2 because everything is stored twice when intermediate sort runs use temporary space. Intermediate sort runs occur when not enough memory exists to perform the entire sort in memory.

   The value for *rows* is the total number of rows that you expect to be in the table.

Storing multiple index fragments in a single dbspace

You can store multiple fragments of the same index in a single dbspace, reducing the total number of dbspaces needed for a fragmented table. You must specify a name for each fragment that you want to store in the same dbspace. Storing multiple index fragments in a single dbspace simplifies the management of dbspaces.

You can also use this feature to improve query performance over storing each fragment in a different dbspace when a dbspace is located on a faster device.

For more information, see information about managing partitions in the *IBM® Informix Administrator's Guide*.

Related concepts:
Types of indexes
Estimating index pages
Managing indexes
Creating and dropping an index in an online environment
Improving performance for index checks
Indexes on user-defined data types
Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

Improving performance for index checks

The **oncheck** utility provides better concurrency for tables that use row locking. When a table uses page locking, **oncheck** places a shared lock on the table when it performs index checks. Shared locks do not allow other users to perform updates, inserts, or deletes on the table while **oncheck** checks or prints the index information.

If the table uses page locking, the database server returns the following message if you run **oncheck** without the **-x** option:

```
WARNING: index check requires a s-lock on stable whose
lock level is page.
```

For detailed information about **oncheck** locking, see the *IBM® Informix Administrator's Reference*.

The following summary describes locking performed during index checks:

- By default, the database server does not place a shared lock on the table when you check an index with the **oncheck -ci, -cI, -pk, -pK, -pl,** or **-pL** options unless the table uses page locking. When **oncheck** checks indexes for a table with page locking, it places a shared lock on the table, so no other users can perform updates, inserts, or deletes until the check has completed.
- By not placing a shared lock on tables using row locks during index checks, the **oncheck** utility cannot be as accurate in the index check. For absolute assurance of a complete index check, execute **oncheck** with the **-x** option. With the **-x** option, **oncheck** places a shared lock on the table, and no other users can perform updates, inserts, or deletes until the check completes.

You can query the **systables** system catalog table to see the current lock level of the table, as the following sample SQL statement shows:

```
SELECT locklevel FROM systables
   WHERE tabname = "customer"
```

If you do not see a value of R (for row) in the **locklevel** column, you can modify the lock level, as the following sample SQL statement shows:

```
ALTER TABLE tab1 LOCK MODE (ROW);
```

Row locking might add other side effects, such as an overall increase in lock usage. For more information about locking levels, see Locking.

Related concepts:
Types of indexes
Estimating index pages
Managing indexes
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Indexes on user-defined data types
Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

Indexes on user-defined data types

You can define your own data types and the functions that operate on these data types. You can define indexes on some kinds of user-defined data types.

DataBlade modules also provide extended data types and functions to the database server.

You can define indexes on the following kinds of user-defined data types:

- Opaque data types
  An *opaque data type* is a fundamental data type that you can use to define columns in the same way you use built-in types. An opaque data type stores a single value and cannot be divided into components by the database server. For information about creating opaque data types, see the CREATE OPAQUE TYPE statement in the *IBM® Informix Guide to SQL: Syntax* and *IBM Informix User-Defined Routines and Data Types Developer's Guide*. For more information about the data types and functions that each DataBlade module provides, see the user guide of each DataBlade module.

- Distinct data types
  A *distinct data type* has the same representation as an existing opaque or built-in data type but is different from these types. For information about distinct data types, see the *IBM Informix Guide to SQL: Reference* and the CREATE DISTINCT TYPE statement in the *IBM Informix Guide to SQL: Syntax*.

For more information about data types, see the *IBM Informix Guide to SQL: Reference*.

- Defining indexes for user-defined data types
  As with built-in data types, you might improve the response time for a query when you define indexes for new data types.
- Using an index that a DataBlade module provides
  DataBlade modules can provide new data types that users can access. A DataBlade module can also provide a new index for the new data type.
- Choosing operator classes for indexes
  For most situations, use the default operators that are defined for a secondary-access method. However, when you want to order the data in a different sequence or provide index support for a user-defined data type, you must extend an operator class.

Related concepts:
Types of indexes
Estimating index pages
Managing indexes
Creating and dropping an index in an online environment
Storing multiple index fragments in a single dbspace
Improving performance for index checks

Related tasks:
Improve query performance with a forest of trees index
Improving performance for index builds

Defining indexes for user-defined data types

As with built-in data types, you might improve the response time for a query when you define indexes for new data types.

The response time for a query might improve when Informix® uses an index for:

- Columns used to join two tables
- Columns that are filters for a query
- Columns in an ORDER BY or GROUP BY clause
- Results of functions that are filters for a query

For more information about when the query performance can improve with an index on a built-in data type, see Improve performance by adding or removing indexes.

Informix and DataBlade modules provide a variety of different types of indexes (also referred to as *secondary-access methods*). A secondary-access method is a set of database server functions that build, access, and manipulate an index structure. These functions encapsulate index operations, such as how to scan, insert, delete, or update nodes in an index.

To create an index on a user-defined data type, you can use any of the following secondary-access methods:

- Generic B-tree index
  A B-tree index is good for a query that retrieves a range of data values. For more information, see B-tree secondary-access method.
- R-tree index
  An R-tree index is good for searches on multidimensional data. For more information, see the *IBM® Informix R-Tree Index User's Guide*.
- Secondary-access methods that a DataBlade module provides for a new data type
  A DataBlade module that supports a certain type of data can also provide a new index for that new data type. For more information, see Using an index that a DataBlade module provides.

You can create a functional index on the resulting values of a user-defined function on one or more columns. For more information, see Using a functional index.

After you choose the desired index type, you might also need to extend an operator class for the secondary-access method. For more information about how to extend operator classes, see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

- B-tree secondary-access method
  Informix provides the *generic B-tree index* for columns in database tables. In traditional relational database systems, the B-tree access method handles only built-in data types and therefore it can only compare two keys of built-in data types. The generic B-tree index is an extended version of a B-tree that Informix provides to support user-defined data types.
- Identifying the available access methods
  To supplement the built-in B-tree secondary-access method that Informix provides, your enterprise might have installed DataBlade modules that implement additional secondary-access methods. If additional access methods exist, they are defined in the **sysams** system catalog table. You can query the **sysams** system catalog to determine if additional access methods are available.
- User-defined secondary-access methods
  If the concepts of *less than*, *greater than*, and *equal* do not apply to the data to be indexed, you might consider using a *user-defined secondary-access method* instead of the built-in secondary-access method, which is a B-tree index. You can use a user-defined secondary-access method to access other indexing structures, such as an R-tree index.
- Using a functional index
  You can create a column index on the actual values in one or more columns. You can also create a functional index on the values of one or more columns that a user-defined function returns from arguments.

B-tree secondary-access method

Informix® provides the *generic B-tree index* for columns in database tables. In traditional relational database systems, the B-tree access method handles only built-in data types and therefore it can only compare two keys of built-in data types. The generic B-tree index is an extended version of a B-tree that Informix provides to support user-defined data types.

Tip: For more information about the structure of a B-tree index and how to estimate the size of a B-tree index, see Estimating index pages**.**
Informix uses the generic B-tree as the built-in secondary-access method. This built-in secondary-access method is registered in the **sysams** system catalog table with the name **btree**. When you use the CREATE INDEX statement (without the USING clause) to create an index, the database server creates a generic B-tree index. For more information, see the CREATE INDEX statement in the *IBM® Informix Guide to SQL: Syntax*.
Tip: *Informix* also defines another secondary-access method, the R-tree index. For more information about how to use an R-tree index, see the *IBM Informix R-Tree Index User's Guide*.

- Uses for a B-tree index
  A B-tree index is good for a query that retrieves a range of data values. If the data to be indexed has a logical sequence to which the concepts of *less than*, *greater than*, and *equal* apply, the generic B-tree index is a useful way to index your data.
- Extending a generic B-tree index
  Initially, the generic B-tree can index data that is one of the built-in data types, and it orders the data in lexicographical sequence. However, you can extend a generic B-tree for some other data types.

Uses for a B-tree index

A B-tree index is good for a query that retrieves a range of data values. If the data to be indexed has a logical sequence to which the concepts of *less than*, *greater than*, and *equal* apply, the generic B-tree index is a useful way to index your data.

Initially, the generic B-tree index supports the relational operators (<,<=,=,>=,>) on all built-in data types and orders the data in lexicographical sequence.

The optimizer considers whether to use the B-tree index to execute a query if you define a generic B-tree index on:

- Columns used to join two tables
- Columns that are filters for a query
- Columns in an ORDER BY or GROUP BY clause
- Results of functions that are filters for a query

Extending a generic B-tree index

Initially, the generic B-tree can index data that is one of the built-in data types, and it orders the data in lexicographical sequence. However, you can extend a generic B-tree for some other data types.

You can extend a generic B-tree to support columns and functions on the following data types:

- *User-defined data types* (opaque and distinct data types) that you want the B-tree index to support
  In this case, you must extend the default operator class of the generic B-tree index.
- *Built-in data types* that you want to order in a different sequence from the lexicographical sequence that the generic B-tree index uses
  In this case, you must define a different operator class from the default generic B-tree index.

An *operator class* is the set of functions (operators) that are associated with a nontraditional B-tree index. For more details on operator classes, see Choosing operator classes for indexes.

Identifying the available access methods

To supplement the built-in B-tree secondary-access method that Informix® provides, your enterprise might have installed DataBlade modules that implement additional secondary-access methods. If additional access methods exist, they are defined in the **sysams** system catalog table. You can query the **sysams** system catalog to determine if additional access methods are available.

To identify the secondary-access methods that are available for your database, query the **sysams** system catalog table with the following SELECT statement:

```
SELECT am_id, am_owner, am_name, am_type FROM sysams
   WHERE am_type = 'S';
```

An 'S' value in the **am_type** column identifies the access method as a secondary-access method. This query returns the following information:

- The **am_id** and **am_name** columns identify the secondary-access method.
- The **am_owner** column identifies the owner of the access method.

In an ANSI-compliant database, the access-method name must be unique within the name space of the user. The access-method name always begins with the owner in the format **am_owner.am_name**.

By default, Informix provides the following definitions in the **sysams** system catalog table for two secondary-access methods, **btree** and **rtree**.

Access Method	am_id Column	am_name Column	am_owner Column
Generic B-tree	1	btree	'informix'
R-tree	2	rtree	'informix'

Important: The **sysams** system catalog table does not contain a row for the built-in primary access method. This primary access method is internal to Informix and does not require a definition in **sysams**. However, the built-in primary access method is always available for use.
If you find additional rows in the **sysams** system catalog table (rows with **am_id** values greater than 2), the database supports additional user-defined access methods. Check the value in the **am_type** column to determine whether a user-defined access method is a primary- or secondary-access method.

For more information about the columns of the **sysams** system catalog table, see the *IBM® Informix Guide to SQL: Reference*. For information about how to determine the operator classes that are available in your database, see Identifying the available operator classes.

User-defined secondary-access methods

If the concepts of *less than*, *greater than*, and *equal* do not apply to the data to be indexed, you might consider using a *user-defined secondary-access method* instead of the built-in secondary-access method, which is a B-tree index. You can use a user-defined secondary-access method to access other indexing structures, such as an R-tree index.

If your database supports a user-defined secondary-access method, you can specify that the database server uses this access method when it accesses a particular index. For information about how to determine the secondary-access methods that your database defines, see Identifying the available access methods.

To choose a user-defined secondary-access method, use the USING clause of the CREATE INDEX statement. The USING clause specifies the name of the secondary-access method to use for the index you create. This name must be listed in the **am_name** column of the **sysams** system catalog table and must be a secondary-access method (the **am_type** column of **sysams** is `'S'`).

The secondary-access method that you specify in the USING clause of CREATE INDEX must already be defined in the **sysams** system catalog. If the secondary-access method has not yet been defined, the CREATE INDEX statement fails.

When you omit the USING clause from the CREATE INDEX statement, the database server uses B-tree indexes as the secondary-access method. For more information, see the CREATE INDEX statement in the *IBM® Informix Guide to SQL: Syntax*.

- R-tree indexes
  Informix supports the *R-tree index* for columns that contain spatial data such as maps and diagrams. An R-tree index uses a tree structure whose nodes store pointers to lower-level nodes.

R-tree indexes

Informix® supports the *R-tree index* for columns that contain spatial data such as maps and diagrams. An R-tree index uses a tree structure whose nodes store pointers to lower-level nodes.

At the leaves of the R-tree are a collection of data pages that store *n*-dimensional shapes. For more information about the structure of an R-tree index and how to estimate the size of an R-tree index, see the *IBM® Informix R-Tree Index User's Guide*.

Using a functional index

You can create a column index on the actual values in one or more columns. You can also create a functional index on the values of one or more columns that a user-defined function returns from arguments.

Important: The database server imposes the following restrictions on the user-defined routines (UDRs) on which a functional index is defined:

- The arguments cannot be column values of a collection data type.
- The function cannot return a large object (including built-in types BLOB, BYTE, CLOB, and TEXT).
- The function cannot be a VARIANT function.
- The function cannot include any DML statement of SQL.
- The function must be a UDR, rather than a built-in function. However, you can create an SPL wrapper that calls and returns the value from a built-in function of SQL.

In addition, do not create functional indexes using any routine that calls the built-in DECRYPT_BINARY( ) or DECRYPT_CHAR( ) functions, which can display encrypted data values in plain text. (Do not attempt to use data values in any encrypted column as an index key.)

To decide whether to use a column index or functional index, determine whether a column index is the right choice for the data that you want to index. An index on a column of some data types might not be useful for typical queries. For example, the following query asks how many images are dark:

```
SELECT COUNT(*) FROM photos WHERE
darkness(picture) > 0.5
```

An index on the **picture** data itself does not improve the query performance. The concepts of *less than*, *greater than*, and *equal* are not particularly meaningful when applied to an image data type. Instead, a functional index that uses the **darkness()** function can improve performance. You might also have a user-defined function that runs frequently enough that performance improves when you create an index on its values.

- What is a functional index?
  A functional index can be a B-tree index, an R-tree index, or a user-defined index type that a DataBlade module provides.
- When is a functional index used?
  The optimizer considers whether to use a functional index to access the results of functions that are in a SELECT clause or are in the filters in the WHERE clause.
- Creating a functional index
  You can build a functional index on a user-defined function. The user-defined function can be either an external function or an SPL function.

Related concepts:
Managing indexes

What is a functional index?

A functional index can be a B-tree index, an R-tree index, or a user-defined index type that a DataBlade module provides.

When you create a functional index, the database server computes the values of the user-defined function and stores them as key values in the index. When a change in the table data causes a change in one of the values of an index key, the database server automatically updates the functional index.

You can use a functional index for functions that return values of both user-defined data types (opaque and distinct) and built-in data types. However, you cannot define a functional index if the function returns a simple-large-object data type (TEXT or BYTE).

For more information about the types of indexes, see Defining indexes for user-defined data types. For information about space requirements for functional indexes, see Estimating index pages.

Related concepts:
Types of indexes

When is a functional index used?

The optimizer considers whether to use a functional index to access the results of functions that are in a SELECT clause or are in the filters in the WHERE clause.

Creating a functional index

You can build a functional index on a user-defined function. The user-defined function can be either an external function or an SPL function.

To build a functional index on a user-defined function:

1. Write the code for the user-defined function if it is an external function.
2. Register the user-defined function in the database with the CREATE FUNCTION statement.
3. Build the functional index with the CREATE INDEX statement.

For example, to create a functional index on the darkness() function:

1. Write the code for the user-defined **darkness()** function that operates on the data type and returns a decimal value.
2. Register the user-defined function in the database with the CREATE FUNCTION statement:

   ```
   CREATE FUNCTION darkness(im image)
   RETURNS decimal
   EXTERNAL NAME '/lib/image.so'
   LANGUAGE C NOT VARIANT
   ```

   In this example, you can use the default operator class for the functional index because the return value of the **darkness()** function is a built-in data type, DECIMAL.

3. Build the functional index with the CREATE INDEX statement.

   ```
   CREATE TABLE photos
   (
       name char(20),
       picture image
   ...
   );
   CREATE INDEX dark_ix ON photos (darkness(picture));
   ```

   In this example, assume that the user-defined data type of **image** has already been defined in the database.

The optimizer can now consider the functional index when you specify the **darkness()** function as a filter in the query:

```
SELECT count(*) FROM photos WHERE
darkness(picture) > 0.5
```

You can also create a composite index with user-defined functions. For more information, see Use composite indexes.
Warning: Do not create a functional index using either the DECRYPT_BINARY() or the DECRYPT_CHAR() function. These functions store plain text data in the database, defeating the purpose of encryption. For more information about encryption, see the *IBM® Informix® Administrator's Guide*.

Using an index that a DataBlade module provides

DataBlade modules can provide new data types that users can access. A DataBlade module can also provide a new index for the new data type.

For example, the Excalibur Text Search DataBlade module provides an index to search text data. For more information, see the *Excalibur Text Search DataBlade Module User's Guide*.

For more information about the types of data and functions that each DataBlade module provides, see the user guide for the DataBlade module. For information about how to determine the types of indexes available in your database, see Identifying the available access methods.

Choosing operator classes for indexes

For most situations, use the default operators that are defined for a secondary-access method. However, when you want to order the data in a different sequence or provide index support for a user-defined data type, you must extend an operator class.

For more information about how to extend an operator class, see *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

- Operator classes
  An *operator class* is a set of function names that is associated with a secondary-access method. These functions allow the secondary-access method to store and search for values of a particular data type.
- Built-in B-tree operator class
  The built-in secondary-access method (the generic B-tree) has a default operator class called **btree_ops**, which is defined in the **sysopclasses** system catalog table.
- Identifying the available operator classes
  You can identify the operator classes that are available for your database by querying the **sysopclasses** system catalog table.
- User-defined operator classes
  The CREATE INDEX statement specifies the operator class to use for each component of an index. If you do not specify an operator class, the CREATE INDEX statement uses the default operator class for the secondary-access method that you create. You can use a user-defined operator class for components of an index.

Operator classes

An *operator class* is a set of function names that is associated with a secondary-access method. These functions allow the secondary-access method to store and search for values of a particular data type.

The query optimizer for the database server uses an operator class to determine if an index can process the query with the least cost. An operator class indicates two things to the query optimizer:

- Which functions that appear in an SQL statement can be evaluated with a given index
  These functions are called the *strategy functions* for the operator class.
- Which functions the index uses to evaluate the strategy functions
  These functions are called the *support functions* for the operator class.

With the information that the operator class provides, the query optimizer can determine whether a given index is applicable to the query. The query optimizer can consider whether to use the index for the given query when the following conditions are true:

- An index exists on the particular column or columns in the query.
- For the index that exists, the operation on the column or columns in the query matches one of the strategy functions in the operator class associated with the index.

The query optimizer reviews the available indexes for the table or tables and matches the index keys with the column specified in the query filter. If the column in the filter matches an index key, and the function in the filter is one of the strategy functions of the operator class, the optimizer includes the index when it determines which query plan has the lowest execution cost. In this manner, the optimizer can determine which index can process the query with the least cost.

Informix® stores information about operator classes in the **sysopclasses** system catalog table.

- Strategy and support functions of a secondary access method
  Informix uses the *strategy functions* of a secondary-access method to help the query optimizer determine whether a specific index is applicable to a specific operation on a data type.
- Default operator classes
  Each secondary-access method has a *default operator class* associated with it. By default, the CREATE INDEX statement associates the default operator class with an index.

Strategy and support functions of a secondary access method

Informix® uses the *strategy functions* of a secondary-access method to help the query optimizer determine whether a specific index is applicable to a specific operation on a data type.

If an index exists and the operator in the filter matches one of the strategy functions in the operator class, the optimizer considers whether to use the index for the query.

Informix uses the *support functions* of a secondary-access method to build and access the index. These functions are not called directly by end users. When an operator in the query filter matches one of the strategy functions, the secondary-access method uses the support functions to traverse the index and obtain the results. Identification of the actual support functions is left to the secondary-access method.

Default operator classes

Each secondary-access method has a *default operator class* associated with it. By default, the CREATE INDEX statement associates the default operator class with an index.

For example, the following CREATE INDEX statement creates a B-tree index on the **postalcode** column and automatically associates the default B-tree operator class with this column:

```
CREATE INDEX postal_ix ON customer(postalcode)
```

For more information about how to specify a new default operator class for an index, see User-defined operator classes.

Built-in B-tree operator class

The built-in secondary-access method (the generic B-tree) has a default operator class called **btree_ops**, which is defined in the **sysopclasses** system catalog table.

By default, the CREATE INDEX statement associates the **btree_ops** operator class with it when you create a B-tree index. For example, the following CREATE INDEX statement creates a generic B-tree index on the **order_date** column of the **orders** table and associates with this index the default operator class for the B-tree secondary-access method:

```
CREATE INDEX orddate_ix ON orders (order_date)
```

Informix® uses the **btree_ops** operator class to specify:

- The strategy functions to tell the query optimizer which filters in a query can use a B-tree index
- The support function to build and search the B-tree index

- B-tree strategy functions
  The **btree_ops** operator class defines the names of strategy functions for the **btree** access method.
- B-tree support function
  The **btree_ops** operator class has one support function, a comparison function called **compare()**. The **btree_ops** operator class has one support function, a comparison function called **compare()**.

B-tree strategy functions

The **btree_ops** operator class defines the names of strategy functions for the **btree** access method.

The strategy functions that the **btree_ops** operator class defines are:

- **lessthan** (<)
- **lessthanorequal** (<=)
- **equal** (=)
- **greaterthanorequal** (>=)
- **greaterthan** (>)

These strategy functions are all *operator functions*. That is, each function is associated with an operator symbol; in this case, with a relational-operator symbol. For more information about relational-operator functions, see the *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

When the query optimizer examines a query that contains a column, it checks to see if this column has a B-tree index defined on it. If such an index exists *and* if the query contains one of the relational operators that the **btree_ops** operator class supports, the optimizer can choose a B-tree index to execute the query.

B-tree support function

The **btree_ops** operator class has one support function, a comparison function called **compare()**. The **btree_ops** operator class has one support function, a comparison function called **compare()**.

The **compare()** function is a user-defined function that returns an integer value to indicate whether its first argument is equal to, less than, or greater than its second argument, as follows:

- A value of `0` when the first argument is *equal to* the second argument
- A value less than `0` when the first argument is *less than* the second argument
- A value greater than `0` when the first argument is *greater than* the second argument

The B-tree secondary-access method uses the **compare()** function to traverse the nodes of the generic B-tree index. To search for data values in a generic B-tree index, the secondary-access method uses the **compare()** function to compare the key value in the query to the key value in an index node. The result of the comparison determines if the secondary-access method needs to search the next-lower level of the index or if the key resides in the current node.

The generic B-tree access method also uses the **compare()** function to perform the following tasks for generic B-tree indexes:

- Sort the keys before the index is built
- Determine the linear order of keys in a generic B-tree index
- Evaluate the relational operators

- Search for data values in an index

The database server uses the **compare()** function to evaluate comparisons in the SELECT statement. To provide support for these comparisons for opaque data types, you must write the **compare()** function. For more information, see the *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

The database server also uses the **compare()** function when it uses a B-tree index to process an ORDER BY clause in a SELECT statement. However, the optimizer does not use the index to perform an ORDER BY operation if the index does not use the btree-ops operator class.

Identifying the available operator classes

You can identify the operator classes that are available for your database by querying the **sysopclasses** system catalog table.

The database server provides the default operator class for the built-in secondary-access method, the generic B-tree index. In addition, your environment might have installed DataBlade modules that implement other operator classes. All operator classes are defined in the **sysopclasses** system catalog table.

To identify the operator classes that are available for your database, query the **sysopclasses** system catalog table with the following SELECT statement:

```
SELECT opclassid, opclassname, amid, am_name
FROM sysopclasses, sysams
WHERE sysopclasses.amid = sysams.am_id
```

This query returns the following information:

- The **opclassid** and **opclassname** columns identify the operator class.
- The **am_id** and **am_name** columns identify the associated secondary-access methods.

By default, the database server provides the following definitions in the **sysopclasses** system catalog table for two operator classes, **btree_ops** and **rtree_ops**.

Access Method	opclassid Column	opclassname Column	amid Column	am_name Column
Generic B-tree	1	btree_ops	1	btree
R-tree	2	rtree_ops	2	rtree

If you find additional rows in the **sysopclasses** system catalog table (rows with **opclassid** values greater than 2), your database supports user-defined operator classes. Check the value in the **amid** column to determine the secondary-access methods to which the operator class belongs.

The **am_defopclass** column in the **sysams** system catalog table stores the operator-class identifier for the default operator class of a secondary-access method. To determine the default operator class for a given secondary-access method, you can run the following query:

```
SELECT am_id, am_name, am_defopclass, opclass_name
FROM sysams, sysopclasses
WHERE sysams.am_defopclass = sysopclasses.opclassid
```

By default, the database server provides the following default operator classes.

Access Method	am_id Column	am_name Column	am_defopclass Column	opclass_name Column
Generic B-tree	1	btree	1	btree_ops
R-tree	2	rtree	2	rtree_ops

For more information about the columns of the **sysopclasses** and **sysams** system catalog tables, see the *IBM® Informix Guide to SQL: Reference*. For information about how to determine the access methods that are available in your database, see Identifying the available access methods.

User-defined operator classes

The CREATE INDEX statement specifies the operator class to use for each component of an index. If you do not specify an operator class, the CREATE INDEX statement uses the default operator class for the secondary-access method that you create. You can use a user-defined operator class for components of an index.

To specify a user-defined operator class for a particular component of an index, you can:

- Use a user-defined operator class that your database already defines.
- Use an R-tree operator class, if your database defined the R-tree secondary-access method. For more information about R-trees, see the *IBM® Informix® R-Tree Index User's Guide*.

If your database supports multiple-operator classes for the secondary-access method that you want to use, you can specify which operator classes the database server is to use for a particular index. For information on how to determine the operator classes that your database defines, see Identifying the available operator classes.

Each part of a composite index can specify a different operator class. You choose the operator classes when you create the index. In the CREATE INDEX statement, you specify the name of the operator class to use after each column or function name in the index-key specification. Each name must be listed in the **opclassname** column of the **sysopclasses** system catalog table and must be associated with the secondary-access method that the index uses.

For example, if your database defines the **abs_btree_ops** secondary-access method to define a new sort order, the following CREATE INDEX statement specifies that the **table1** table associates the **abs_btree_ops** operator class with the **col1_ix** B-tree index:

```
CREATE INDEX col1_ix ON table1(col1 abs_btree_ops)
```

The operator class that you specify in the CREATE INDEX statement must already be defined in the **sysopclasses** system catalog with the CREATE OPCLASS statement. If the operator class has not yet been defined, the CREATE INDEX statement fails. For information about how to create an operator class, see *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Locking

The database server uses locks, which can affect concurrency and performance. You can monitor and administer locks.

- Locks
  A *lock* is a software mechanism that you can set to prevent others from using a resource. You can place a lock on a single row or key, a page of data or index keys, a whole table, or an entire database.
- Configuring the lock mode
  When you create a table, the default lock mode is `page`. You can change the lock mode (and thus increase or decrease concurrency) when you create or alter tables or by setting the **IFX_DEF_TABLE_LOCKMODE** environment variable or the DEF_TABLE_LOCKMODE configuration parameter.
- Setting the lock mode to wait
  When an application process encounters a lock, the default behavior of the database server is to return an error. Instead, you can run an SQL statement to set the lock mode to wait. This specifies that an application process does not proceed until the lock is removed.
- Locks with the SELECT statement
  The type and duration of locks that the database server places depend on the isolation level set in the application, the database mode (logging, nonlogging, or ANSI,) and on whether the SELECT statement is within an update cursor. These locks can affect overall performance because they affect concurrency.
- Locks placed with INSERT, UPDATE, and DELETE statements
  When you execute an INSERT, UPDATE, or DELETE statement, the database server uses exclusive locks. An exclusive lock means that no other users can update or delete the item until the database server removes the lock.
- The internal lock table
  The database server stores locks in an internal lock table. When the database server reads a row, it checks if the row or its associated page, table, or database is listed in the lock table. If it is in the lock table, the database server must also check the lock type.
- Monitoring locks
  You can analyze information about locks and monitor locks by viewing information in the internal lock table that contains stored locks.
- Locks for smart large objects
  Smart large objects have several unique locking behaviors because their columns are typically much larger than other columns in a table.

Locks

A *lock* is a software mechanism that you can set to prevent others from using a resource. You can place a lock on a single row or key, a page of data or index keys, a whole table, or an entire database.

Additional types of locks are available for smart large objects. For more information, see Locks for smart large objects.

The maximum number of rows or pages locked in a single transaction is controlled by the total number of locks configured. The number of tables in which those rows or pages are locked is not explicitly controlled.

- Locking granularity
  The level and type of information that the lock protects is called *locking granularity*. Locking granularity affects performance.
- Row and key locks
  Row and key locks generally provide the best overall performance when you are updating a relatively small number of rows, because they increase concurrency. However, the database server incurs some overhead in obtaining a lock. For an operation that changes a large number of rows, obtaining one lock per row might not be cost effective.
- Page locks
  Page locking is the default mode when you create a table without the LOCK MODE clause. With page locking, instead of locking only the row, the database server locks the entire page that contains the row. If you update several rows on the same page, the database server uses only one lock for the page.
- Table locks
  In a data warehouse environment, it might be more appropriate for queries to acquire locks of larger granularity. For example, if a query accesses most of the rows in a table, its efficiency increases if it acquires a smaller number of table locks instead of many page or row locks.
- Database locks
  You can place a lock on the entire database when you open the database with the DATABASE statement. A database lock prevents read or update access by anyone but the current user.

Related concepts:
Configuring the lock mode
Locks with the SELECT statement
Locks placed with INSERT, UPDATE, and DELETE statements
The internal lock table
Locks for smart large objects
Related tasks:
Setting the lock mode to wait
Related reference:
Monitoring locks

Locking granularity

The level and type of information that the lock protects is called *locking granularity*. Locking granularity affects performance.

When a user cannot access a row or key, the user can wait for another user to unlock the row or key. If a user locks an entire page, a higher probability exists that more users will wait for a row in the page.

The ability of more than one user to access a set of rows is called *concurrency*. The goal of the database administrator is to increase concurrency to increase total performance without sacrificing performance for an individual user.

Row and key locks

Row and key locks generally provide the best overall performance when you are updating a relatively small number of rows, because they increase concurrency. However, the database server incurs some overhead in obtaining a lock. For an operation that changes a large number of rows, obtaining one lock per row might not be cost effective.

For an operation that changes a large number of rows, consider Page locks.

The default locking mode is page-locking. If you want row or key locks, you must create the table with row locking on or alter the table.

The following example shows how to create a table with row locking on:

```
CREATE TABLE customer(customer_num serial, lname char(20)...)
   LOCK MODE ROW;
```

The ALTER TABLE statement can also change the lock mode.

When the lock mode is ROW and you insert or update a row, the database server creates a row lock. In some cases, you place a row lock by simply reading the row with a SELECT statement.

When the lock mode is ROW and you insert, update, or delete a key (performed automatically when you insert, update, or delete a row), the database server creates a lock on the key in the index.

- Key-value locks
  When a user deletes a row within a transaction, the row cannot be locked because it does not exist. However, the database server must somehow record that a row existed until the end of the transaction. The database server uses *key-value locking* to lock the deleted row.

Key-value locks

When a user deletes a row within a transaction, the row cannot be locked because it does not exist. However, the database server must somehow record that a row existed until the end of the transaction. The database server uses *key-value locking* to lock the deleted row.

When the database server deletes a row, key values in the indexes for the table are not removed immediately. Instead, each key value is marked as deleted, and a lock is placed on the key value.

Other users might encounter key values that are marked as deleted. The database server must determine whether a lock exists. If a lock exists, the delete has not been committed, and the database server sends a lock error back to the application (or it waits for the lock to be released if the user executed SET LOCK MODE TO WAIT).

One of the most important uses for key-value locking is to assure that a unique key remains unique through the end of the transaction that deleted it. Without this protection mechanism, user A might delete a unique key within a transaction, and user B might insert a row with the same key before the transaction commits. This scenario makes rollback by user A impossible. Key-value locking prevents user B from inserting the row until the end of user A's transaction.

Page locks

Page locking is the default mode when you create a table without the LOCK MODE clause. With page locking, instead of locking only the row, the database server locks the entire page that contains the row. If you update several rows on the same page, the database server uses only one lock for the page.

When you insert or update a row, the database server creates a page lock on the data page. In some cases, the database server creates a page lock when you simply read the row with a SELECT statement.

When you insert, update, or delete a key (performed automatically when you insert, update, or delete a row), the database server creates a lock on the page that contains the key in the index.
Important: A page lock on an index page can decrease concurrency more substantially than a page lock on a data page. Index pages are dense and hold a large number of keys. By locking an index page, you make a potentially large number of keys unavailable to other users until you release the lock. Tables that use page locks cannot support the USELASTCOMMITTED concurrency feature, which is described in the Committed Read isolation section.
Page locks are useful for tables in which the normal user changes a large number of rows at one time. For example, an orders table that holds orders that are commonly inserted and queried individually is not a good candidate for page locking. But a table that holds old orders and is updated nightly with all of the orders placed during the

day might be a good candidate. In this case, the type of isolation level that you use to access the table is important. For more information, see Isolation level.

Table locks

In a data warehouse environment, it might be more appropriate for queries to acquire locks of larger granularity. For example, if a query accesses most of the rows in a table, its efficiency increases if it acquires a smaller number of table locks instead of many page or row locks.

The database server can place two types of table locks:

- Shared lock
  No other users can write to the table.
- Exclusive lock
  No other users can read from or write to the table.

Another important distinction between these two types of table locks is the actual number of locks placed:

- In shared mode, the database server places one shared lock on the table, which informs other users that no updates can be performed. In addition, the database server adds locks for every row updated, deleted, or inserted.
- In exclusive mode, the database server places only one exclusive lock on the table, no matter how many rows it updates. If you update most of the rows in the table, place an exclusive lock on the table.

Important: A table lock on a table can decrease update concurrency radically. Only one update transaction can access that table at any given time, and that update transaction locks out all other transactions. However, multiple read-only transactions can simultaneously access the table. This behavior is useful in a data warehouse environment where the data is loaded and then queried by multiple users.
You can switch a table back and forth between table-level locking and the other levels of locking. This ability to switch locking levels is useful when you use a table in a data warehouse mode during certain time periods but not in others.

A transaction tells the database server to use table-level locking for a table with the LOCK TABLE statement. The following example places an exclusive lock on the table:

```
LOCK TABLE tab1 IN EXCLUSIVE MODE;
```

The following example places a shared lock on the table:

```
LOCK TABLE tab1 IN SHARE MODE:
```

In some cases, the database server places its own table locks. For example, if the isolation level is Repeatable Read, and the database server must read a large portion of the table, it places a table lock automatically instead of setting row or page locks. The database server places a table lock on a table when it creates or drops an index.

Database locks

You can place a lock on the entire database when you open the database with the DATABASE statement. A database lock prevents read or update access by anyone but the current user.

The following statement opens and locks the sales database:

```
DATABASE sales EXCLUSIVE
```


Configuring the lock mode

When you create a table, the default lock mode is `page`. You can change the lock mode (and thus increase or decrease concurrency) when you create or alter tables or by setting the **IFX_DEF_TABLE_LOCKMODE** environment variable or the DEF_TABLE_LOCKMODE configuration parameter.

If you know that most of your applications might benefit from a lock mode of row, you can take one of the following actions:

- Use the LOCK MODE ROW clause in each CREATE TABLE statement or ALTER TABLE statement.
- Set the **IFX_DEF_TABLE_LOCKMODE** environment variable to ROW so that all tables you subsequently create within a session use ROW without the need to specify the lock mode in the CREATE TABLE statement or ALTER TABLE statement.
- Set the DEF_TABLE_LOCKMODE configuration parameter to ROW so that all tables subsequently created within the database server use ROW without the need to specify the lock mode in the CREATE TABLE statement or ALTER TABLE statement.

If you change the lock mode with the **IFX_DEF_TABLE_LOCKMODE** environment variable or DEF_TABLE_LOCKMODE configuration parameter, the lock mode of existing tables are not affected. Existing tables continue to use the lock mode with which they were defined at the time they were created.

In addition, if you previously changed the lock mode of a table to ROW, and subsequently execute an ALTER TABLE statement to alter some other characteristic of the table (such as add a column or change the extent size), you do not need to specify the lock mode. The lock mode remains at ROW and is not set to the default PAGE mode.

You can still override the lock mode of individual tables by specifying the LOCK MODE clause in the CREATE TABLE statement or ALTER TABLE statement.

The following list shows the order of precedence for the lock mode on a table:

- The system default is page locks. The database server uses this system default if you do not set the configuration parameter, do not set the environment variable, or do not specify the LOCK MODE clause in the SQL statements.
- If you set the DEF_TABLE_LOCKMODE configuration parameter, the database server uses this value when you do not set the environment variable, or do not specify the LOCK MODE clause in the SQL statements.
- If you set the **IFX_DEF_TABLE_LOCKMODE** environment variable, this value overrides the DEF_TABLE_LOCKMODE configuration parameter and system default. The database server uses this value when you do not specify the LOCK MODE clause in the SQL statements.
- If you specify the LOCK MODE clause in the CREATE TABLE statement or ALTER TABLE statement, this value overrides the **IFX_DEF_TABLE_LOCKMODE,** the DEF_TABLE_LOCKMODE configuration parameter and system default.

Related concepts:
Locks
Locks with the SELECT statement
Locks placed with INSERT, UPDATE, and DELETE statements
The internal lock table
Locks for smart large objects
Related tasks:
Setting the lock mode to wait
Related reference:
Monitoring locks

Setting the lock mode to wait

When an application process encounters a lock, the default behavior of the database server is to return an error. Instead, you can run an SQL statement to set the lock mode to wait. This specifies that an application process does not proceed until the lock is removed.

To suspend the current process until the lock releases, run the following SQL statement :

```
SET LOCK MODE TO WAIT;
```

You can also specify the maximum number of seconds that a process waits for a lock to be released before issuing an error. In the following example, the database server waits for 20 seconds before issuing an error:

```
SET LOCK MODE TO WAIT 20;
```

To return to the default behavior (no waiting for locks), execute the following statement:

```
SET LOCK MODE TO NOT WAIT;
```

Related concepts:
Locks
Configuring the lock mode
Locks with the SELECT statement
Locks placed with INSERT, UPDATE, and DELETE statements
The internal lock table
Locks for smart large objects
Related reference:
Monitoring locks

Locks with the SELECT statement

The type and duration of locks that the database server places depend on the isolation level set in the application, the database mode (logging, nonlogging, or ANSI,) and on whether the SELECT statement is within an update cursor. These locks can affect overall performance because they affect concurrency.

- Isolation level
  The number and duration of locks placed on data during a SELECT statement depend on the level of isolation that the user sets. The type of isolation can affect overall performance because it affects concurrency.
- Locking nonlogging tables
  The database server does not place page or row locks on a nonlogging table when you use the table within a transaction. However, you can lock nonlogging tables to prevent concurrency problems when other users are modifying a nonlogging table
- Update cursors
  An update cursor is a special kind of cursor that applications can use when the row might potentially be updated. Update cursors use *promotable locks* in which the database server places an update lock on the row when the application fetches the row. The lock is changed to an exclusive lock when the application uses an update cursor and UPDATE...WHERE CURRENT OF to update the row.

Related concepts:
Locks
Configuring the lock mode
Locks placed with INSERT, UPDATE, and DELETE statements
The internal lock table
Locks for smart large objects
Related tasks:
Setting the lock mode to wait
Related reference:
Monitoring locks

Isolation level

The number and duration of locks placed on data during a SELECT statement depend on the level of isolation that the user sets. The type of isolation can affect overall performance because it affects concurrency.

Before you execute a SELECT statement, you can set the isolation level with the SET ISOLATION statement, which is part of the Informix® extension to the ANSI SQL-92 standard, or with the ANSI/ISO-compliant SET TRANSACTION. The main differences between the two statements are that SET ISOLATION has an additional isolation level, Cursor Stability, and SET TRANSACTION cannot be executed more than once in a transaction as SET ISOLATION can. The SET ISOLATION statement is part of the Informix extension to the ANSI SQL-92 standard. The SET ISOLATION statement can change the enduring isolation level for the session

- Dirty Read isolation
  The Dirty Read isolation (or ANSI Read Uncommitted) level does not place any locks on any rows fetched during a SELECT statement. Dirty Read isolation is appropriate for static tables that are used for queries.
- Committed Read isolation
  A reader with the Committed Read isolation (or ANSI Read Committed) isolation level checks for locks before returning a row. By checking for locks, the reader cannot return any uncommitted rows.
- Cursor Stability isolation
  A reader with Cursor Stability isolation acquires a shared lock on the row that is currently fetched. This action assures that no other user can update the row until the user fetches a new row.
- Repeatable Read isolation
  Repeatable Read isolation (ANSI Serializable and ANSI Repeatable Read) is the strictest isolation level. With Repeatable Read, the database server locks all rows examined (not just fetched) for the duration of the transaction.

Dirty Read isolation

The Dirty Read isolation (or ANSI Read Uncommitted) level does not place any locks on any rows fetched during a SELECT statement. Dirty Read isolation is appropriate for static tables that are used for queries.

Use Dirty Read isolation with care if update activity occurs at the same time. With Dirty Read, the reader can read a row that has not been committed to the database and might be eliminated or changed during a rollback. For example, consider the following scenario:

```
User 1 starts a transaction.
User 1 inserts row A.
User 2 reads row A.
User 1 rolls back row A.
```

User 2 reads row A, which user 1 rolls back seconds later. In effect, user 2 read a row that was never committed to the database. Uncommitted data that is rolled back can be a problem in applications.

Because the database server does not check or place any locks for queries, Dirty Read isolation offers the best performance of all isolation levels. However, because of potential problems with uncommitted data that is rolled back, use Dirty Read isolation with care.

Because problems with uncommitted data that is rolled back are an issue only with transactions, databases that do not have transaction (and hence do not allow transactions) use Dirty Read as a default isolation level. In fact, Dirty Read is the only isolation level allowed for databases that do not have transaction logging.

Committed Read isolation

A reader with the Committed Read isolation (or ANSI Read Committed) isolation level checks for locks before returning a row. By checking for locks, the reader cannot return any uncommitted rows.

The database server does not actually place any locks for rows read during Committed Read. It simply checks for any existing rows in the internal lock table.

Committed Read is the default isolation level for databases with logging if the log mode is not ANSI-compliant. For databases created with a logging mode that is not ANSI-compliant, Committed Read is an appropriate isolation level for most activities. For ANSI-compliant databases, Repeatable Read is the default isolation level.

- Ways to reduce the risk of Committed Read isolation level conflicts
  In the Committed Read isolation level, locks held by other sessions can cause SQL operations to fail if the current session cannot acquire a lock or if the database server detects a *deadlock*. (A deadlock occurs when two users hold locks, and each user wants to acquire a lock that the other user owns.) The LAST COMMITTED keyword option to the SET ISOLATION COMMITTED READ statement of SQL reduces the risk of locking conflicts.

Ways to reduce the risk of Committed Read isolation level conflicts

In the Committed Read isolation level, locks held by other sessions can cause SQL operations to fail if the current session cannot acquire a lock or if the database server detects a *deadlock*. (A deadlock occurs when two users hold locks, and each user wants to acquire a lock that the other user owns.) The LAST COMMITTED keyword option to the SET ISOLATION COMMITTED READ statement of SQL reduces the risk of locking conflicts.

The LAST COMMITTED keyword option to the SET ISOLATION COMMITTED READ statement of SQL instructs the server to return the most recently committed version of the rows, even if another concurrent session holds an exclusive lock. You can use the LAST COMMITTED keyword option for B-tree and functional indexes, tables that support transaction logging, and tables that do not have page-level locking or exclusive locks. For more information, see information about the SET ISOLATION statement in the *IBM® Informix® Guide to SQL: Syntax*.

For databases created with transaction logging, you can set the USELASTCOMMITTED configuration parameter to specify whether the database server uses the last committed version of the data, rather than wait for the lock to be released, when sessions using the Dirty Read or Committed Read isolation level (or the ANSI/ISO level of Read Uncommitted or Read Committed) attempt to read a row on which a concurrent session holds a shared lock. The last committed version of the data is the version of the data that existed before any updates occurred.

If no value or a value of NONE is set for the USELASTCOMMITTED configuration parameter or for the USELASTCOMMITTED session environment variable, sessions in a COMMITTED READ or READ COMMITTED isolation level wait for any exclusive locks to be released, unless the SET ISOLATION COMMITTED READ LAST COMMITTED statement of SQL instructs the database server to read the most recently committed version of the data.

Setting the USELASTCOMMITTED configuration parameter to operate with the Committed Read isolation level can affect performance only if concurrent conflicting updates occur. When concurrent conflicting updates occur, the performance of queries depends on the dynamics of the transactions. For example, a reader using the last committed version of the data, might need to undo the updates made to a row by another concurrent transaction. This situation involves reading one or more log records, thereby increasing the I/O traffic, which can affect performance.

Related information:
USELASTCOMMITTED configuration parameter

Cursor Stability isolation

A reader with Cursor Stability isolation acquires a shared lock on the row that is currently fetched. This action assures that no other user can update the row until the user fetches a new row.

In the example for a cursor in Figure 1, at *fetch a row* the database server releases the lock on the previous row and places a lock on the row being fetched. At *close the cursor*, the server releases the lock on the last row.
Figure 1. Locks placed for cursor stability

```
set isolation to cursor stability
declare cursor for SELECT * FROM customer
open the cursor
while there are more rows
   fetch a row
   do work
end while
close the cursor
```

If you do not use a cursor to fetch data, Cursor Stability isolation behaves in the same way as Committed Read. No locks are actually placed.

Repeatable Read isolation

Repeatable Read isolation (ANSI Serializable and ANSI Repeatable Read) is the strictest isolation level. With Repeatable Read, the database server locks all rows examined (not just fetched) for the duration of the transaction.

The example in Figure 1 shows when the database server places and releases locks for a repeatable read. At *fetch a row*, the server places a lock on the row being fetched and on every row it examines in order to retrieve this row. At *close the cursor*, the server releases the lock on the last row.
Figure 1. Locks placed for repeatable read

```
set isolation to repeatable read
begin work
declare cursor for SELECT * FROM customer
open the cursor
while there are more rows
    fetch a row
    do work
end while
close the cursor
commit work
```

Repeatable Read is useful during any processing in which multiple rows are examined, but none must change during the transaction. For example, suppose an application must check the account balance of three accounts that belong to one person. The application gets the balance of the first account and then the second. But, at the same time, another application begins a transaction that debits the third account and credits the first account. By the time that the original application obtains the account balance of the third account, it has been debited. However, the original application did not record the debit of the first account.

When you use Committed Read or Cursor Stability, the previous scenario can occur. However, it cannot occur with Repeatable Read. The original application holds a read lock on each account that it examines until the end of the transaction, so the attempt by the second application to change the first account fails (or waits, depending upon SET LOCK MODE).

Because even examined rows are locked, if the database server reads the table sequentially, a large number of rows unrelated to the query result can be locked. For this reason, use Repeatable Read isolation for tables when the database server can use an index to access a table. If an index exists and the optimizer chooses a sequential scan instead, you can use directives to force use of the index. However, forcing a change in the query path might negatively affect query performance.

Locking nonlogging tables

The database server does not place page or row locks on a nonlogging table when you use the table within a transaction. However, you can lock nonlogging tables to prevent concurrency problems when other users are modifying a nonlogging table

Use one of the following methods to prevent concurrency problems when other users are modifying a nonlogging table:

- Lock the table in exclusive mode for the whole transaction.
- Use Repeatable Read isolation level for the whole transaction.

Important: Nonlogging raw tables are intended for fast loading of data. You should change the table to STANDARD before you use it in a transaction or modify the data within it.

Update cursors

An update cursor is a special kind of cursor that applications can use when the row might potentially be updated. Update cursors use *promotable locks* in which the database server places an update lock on the row when the application fetches the row. The lock is changed to an exclusive lock when the application uses an update cursor and UPDATE...WHERE CURRENT OF to update the row.

When the update lock is on the row as the application fetches it, other users can still view the row.

In some cases, the database server might place locks on rows that the database server has examined but not actually fetched. Whether this behavior occurs depends on how the database server executes the SQL statement.

The advantage of an update cursor is that you can view the row with the confidence that other users cannot change it or view it with an update cursor while you are viewing it and before you update it.

If you do not update the row, the default behavior of the database server is to release the update lock when you execute the next FETCH statement or close the cursor. However, if you execute the SET ISOLATION statement with the RETAIN UPDATE LOCKS clause, the database server does not release any currently existing or subsequently placed update locks until the end of the transaction.

The code in Figure 1 shows when the database server places and releases update locks with a cursor. At *fetch row 1*, the database server places an update lock on row 1. At *fetch row 2*, the server releases the update lock on row 1 and places an update lock on row 2. However, after the database server executes the SET ISOLATION statement with the RETAIN UPDATE LOCKS clause, it does not release any update locks until the end of the transaction. At *fetch row 3*, it places an update lock on row 3. At *fetch row 4*, it places an update lock on row 4. At *commit work*, the server releases the update locks for rows 2, 3, and 4.
Figure 1. When update locks are released

```
declare update cursor
begin work
open the cursor
fetch row 1
fetch row 2
SET ISOLATION TO COMMITTED READ
    RETAIN UPDATE LOCKS
fetch row 3
fetch row 4
commit work
```

In an ANSI-compliant database, update cursors are usually not needed because any select cursor behaves the same as an update cursor without the RETAIN UPDATE LOCKS clause.

The code in Figure 2 shows the database server promoting an update lock to an exclusive lock. At *fetch the row*, the server places an update lock on the row being fetched. At *update the row*, the server promotes the lock to exclusive. At *commit work*, it releases the lock.
Figure 2. When update locks are promoted

```
declare update cursor
begin work
open the cursor
fetch the row
do work
update the row (use WHERE CURRENT OF)
commit work
```

To use an update cursor, run SELECT FOR UPDATE in your application.

Locks placed with INSERT, UPDATE, and DELETE statements

When you execute an INSERT, UPDATE, or DELETE statement, the database server uses exclusive locks. An exclusive lock means that no other users can update or delete the item until the database server removes the lock.

In addition, no other users can view the row unless they are using the Dirty Read isolation level.

When the database server removes the exclusive lock depends on whether the database supports transaction logging:

- If the database supports logging, the database server removes all exclusive locks when the transaction completes (commits or rolls back).
- If the database does not support logging, the database server removes all exclusive locks immediately after the INSERT, MERGE, UPDATE, or DELETE statement completes, except when the lock is on the row that is currently being fetched into an update cursor.
  In this situation, the lock is retained during the fetch operation on the row, but only until the server fetches the next row, or until the server updates the current row by promoting the lock to an exclusive lock.

In a nonlogging database, the promotable update lock on a row fetched for update can be released by a DDL operation on the database while the INSERT, MERGE, UPDATE, or DELETE statement that originally created the lock is still running. To reduce the risk of data corruption if a concurrent session modifies the unlocked row, restrict operations that use promotable update locks to databases that support transaction logging.

Related concepts:
Locks
Configuring the lock mode
Locks with the SELECT statement
The internal lock table
Locks for smart large objects
Related tasks:
Setting the lock mode to wait
Related reference:
Monitoring locks

The internal lock table

The database server stores locks in an internal lock table. When the database server reads a row, it checks if the row or its associated page, table, or database is listed in the lock table. If it is in the lock table, the database server must also check the lock type.

The following table shows the types of locks that the lock table can contain.

Lock Type	Description	Statement That Usually Places the Lock
S	Shared lock	SELECT
X	Exclusive lock	INSERT, UPDATE, DELETE
U	Update lock	SELECT in an update cursor
B	Byte lock	Any statement that updates VARCHAR columns

A byte lock is generated only if you shrink the size of a data value in a VARCHAR column. The byte lock exists solely for roll forward and rollback execution, so a byte lock is created only if you are working in a database that uses logging. Byte locks appear in **onstat -k** output only if you are using row-level locking; otherwise, they are merged with the page lock.

In addition, the lock table might store *intent locks*, with the same lock type as previously shown. In some cases, a user might need to register his or her possible intent to lock an item, so that other users cannot place a lock on the item.

Depending on the type of operation and the isolation level, the database server might continue to read the row and place its own lock on the row, or it might wait for the lock to be released (if the user executed SET LOCK MODE TO WAIT). The following table shows the locks that a user can place if another user holds a certain type of lock. For example, if one user holds an exclusive lock on an item, another user requesting any kind of lock (exclusive, update, or shared) receives an error.

	Hold X lock	Hold U lock	Hold S lock
Request X lock	No	No	Yes
Request U lock	No	No	Yes
Request S lock	No	Yes	Yes

Related concepts:
Locks
Configuring the lock mode
Locks with the SELECT statement
Locks placed with INSERT, UPDATE, and DELETE statements
Locks for smart large objects
Related tasks:
Setting the lock mode to wait
Related reference:
Monitoring locks

Monitoring locks

You can analyze information about locks and monitor locks by viewing information in the internal lock table that contains stored locks.

View the lock table with **onstat -k**. Figure 1 shows sample output for **onstat -k**.
Figure 1. onstat -k output

```
Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
300b77d0 0        40074140 0        HDR+S    10002    106        0
300b7828 0        40074140 300b77d0 HDR+S    10197    123        0
300b7854 0        40074140 300b7828 HDR+IX   101e4    0          0
300b78d8 0        40074140 300b7854 HDR+X    101e4    102        0
 4 active, 5000 total, 8192 hash buckets
```

In this example, a user is inserting one row in a table. The user holds the following locks (described in the order shown):

- A shared lock on the database
- A shared lock on a row in the **systables** system catalog table
- An intent-exclusive lock on the table
- An exclusive lock on the row

To determine the table to which the lock applies, execute the following SQL statement. For *tblsnum*, substitute the value shown in the **tblsnum** field in the **onstat -k** output.

```
SELECT *
   FROM SYSTABLES
   WHERE HEX(PARTNUM) = "tblsnum";
```

Where tblsnum is the modified value that **onstat -k** returns. For example, if **onstat -k** returns `10027f`, tbslnum is `0x0010027F`.
You can also query the **syslocks** table in the **sysmaster** database to obtain information about each active lock. The **syslocks** table contains the following columns.

Column	Description
dbsname	Database on which the lock is held
tabname	Name of the table on which the lock is held
rowidlk	ID of the row on which the lock is held (0 indicates a table lock.)
keynum	The key number for the row
type	Type of lock
owner	Session ID of the lock owner
waiter	Session ID of the first waiter on the lock

- Configuring and managing lock usage
  The LOCKS configuration parameter specifies the initial size of the internal lock table. If the database server increases the size of the lock table, you should increase the size of the LOCKS configuration parameter.
- Monitoring lock waits and lock errors
  You can view information about sessions, lock usage, and lock waits.
- Monitoring the number of free locks
  You can find the current number of free locks on a lock-free list by viewing the output of the **onstat -L** command .
- Monitoring deadlocks
  You can monitor deadlocks. A *deadlock* occurs when two users hold locks, and each user wants to acquire a lock that the other user owns.
- Monitoring isolation levels that sessions use
  The **onstat -g ses** and **onstat -g sql** output shows the isolation level that a session is currently using.

Related concepts:
Locks
Configuring the lock mode
Locks with the SELECT statement
Locks placed with INSERT, UPDATE, and DELETE statements
The internal lock table
Locks for smart large objects
Related tasks:
Setting the lock mode to wait

Configuring and managing lock usage

The LOCKS configuration parameter specifies the initial size of the internal lock table. If the database server increases the size of the lock table, you should increase the size of the LOCKS configuration parameter.

For information about how to determine an initial value for the LOCKS configuration parameter, see The LOCKS configuration parameter and memory utilization.

If the number of locks needed by sessions exceeds the value set in the LOCKS configuration parameter, the database server attempts to increase the lock table by doubling its size. Each time that the lock table overflows (when the number of locks needed is greater than the current size of the lock table), the database server increases the size of the lock table, up to 99 times. Each time that the database server increases the size of the lock table, the server attempts to double its size. However, the server will limit each actual increase to no more than the maximum number of added locks shown in Table 1. After the 99th time that the database server increases the lock table, the server no longer increases the size of the lock table, and an application needing a lock receives an error.

Maximum number of locks allowed on 32-bit and 64-bit platforms

The following table shows the maximum number of allowed locks.

Table 1. Maximum number of locks on 32-bit and 64-bit platforms

Platform	Maximum Number of Initial Locks	Maximum Number of Dynamic Lock Table Extensions	Maximum Number of Locks Added Per Lock Table Extension	Maximum Number of Locks Allowed
32-bit	8,000,000	99	100,000	8,000,000 + (99 x 100,000)
64-bit	500,000,000	99	1,000,000	500,000,000 + (99 x 1,000,000)

View messages concerning increases to the size of the lock table

Every time the database server increases the size of the lock table, the server places a message in the message log file. You should monitor the message log file periodically and increase the size of the LOCKS configuration parameter if you see that the database server has increased the size of the lock table.

Monitor out-of-locks errors

To monitor the number of times that applications receive the out-of-locks error, view the **ovlock** field in the output of **onstat -p**. You can also see similar information from the **sysprofile** table in the **sysmaster** database. The following rows contain the relevant statistics.

Row	Description
ovlock	Number of times that sessions attempted to exceed the maximum number of locks
lockreqs	Number of times that sessions requested a lock
lockwts	Number of times that sessions waited for a lock

Examine how applications use locks

If the database server is using an unusually large number of locks, you can examine how individual applications are using locks, as follows:

1. Monitor sessions with **onstat -u** to see if a particular user is using an especially high number of locks (a high value in the **locks** column).
2. If a particular user uses a large number of locks, examine the SQL statements in the application to determine whether you should lock the table or use individual row or page locks.

A table lock is more efficient than individual row locks, but it reduces concurrency.

One way to reduce the number of locks placed on a table is to alter a table to use page locks instead of row locks. However, page locks reduce overall concurrency for the table, which can affect performance.

You can also reduce the number of locks placed on a table by locking the table in exclusive mode.

Related concepts:
The LOCKS configuration parameter and memory utilization

Monitoring lock waits and lock errors

You can view information about sessions, lock usage, and lock waits.

If the application executes SET LOCK MODE TO WAIT, the database server waits for a lock to be released instead of returning an error. An unusually long wait for a lock can give users the impression that the application is hanging.

In Figure 1, the **onstat -u** output shows that session ID 84 is waiting for a lock (`L` in the first column of the **Flags** field). To find out the owner of the lock, use the **onstat -k** command.
Figure 1. onstat -u output that shows lock usage

```
onstat -u

Userthreads
address  flags   sessid user     tty     wait     tout locks nreads nwrites
40072010 ---P--D 7      informix -          0        0     0     35       75
400723c0 ---P--- 0      informix -          0        0     0      0        0
40072770 ---P--- 1      informix -          0        0     0      0        0
40072b20 ---P--- 2      informix -          0        0     0      0        0
40072ed0 ---P--F 0      informix -          0        0     0      0        0
40073280 ---P--B 8      informix -          0        0     0      0        0
40073630 ---P--- 9      informix -          0        0     0      0        0
400739e0 ---P--D 0      informix -          0        0     0      0        0
40073d90 ---P--- 0      informix -          0        0     0      0        0
 40074140 Y-BP--- 81    lsuto    4  50205788        0     4    106      221
400744f0 --BP--- 83     jsmit    -          0        0     4      0        0
400753b0 ---P--- 86     worth    -          0        0     2      0        0
40075760 L--PR-- 84      jones    3   300b78d8    -1     2      0        0
 13 active, 128 total, 16 maximum concurrent

onstat -k

Locks
address  wtlist   owner    lklist   type    tblsum rowid  key#/bsiz
300b77d0 0        40074140 0        HDR+S   10002  106     0
300b7828 0        40074140 300b77d0 HDR+S   10197  122     0
```

```
300b7854 0         40074140 300b7828 HDR+IX  101e4  0       0
 300b78d8 40075760 40074140 300b7854 HDR+X  101e4  100     0
300b7904 0         40075760 0            S   10002  106     0
300b7930 0         40075760 300b7904     S   10197  122     0
 6 active, 5000 total, 8192 hash buckets
```

To find out the owner of the lock for which session ID 84 is waiting:

1. Obtain the address of the lock in the **wait** field (`300b78d8`) of the **onstat -u** output.
2. Find this address (`300b78d8`) in the **Locks address** field of the **onstat -k** output.
   The **owner** field of this row in the **onstat -k** output contains the address of the user thread (`40074140`).
3. Find this address (`40074140`) in the **Userthreads** field of the **onstat -u** output.
   The **sessid** field of this row in the **onstat -u** output contains the session ID (`81`) that owns the lock.

To eliminate the contention problem, you can have the user exit the application gracefully. If this solution is not possible, you can stop the application process or remove the session with **onmode -z**.

Monitoring the number of free locks

You can find the current number of free locks on a lock-free list by viewing the output of the **onstat -L** command .

Related information:
onstat -L command: Print the number of free locks

Monitoring deadlocks

You can monitor deadlocks. A *deadlock* occurs when two users hold locks, and each user wants to acquire a lock that the other user owns.

For example, user **pradeep** holds a lock on row 10. User **jane** holds a lock on row 20. Suppose that **jane** wants to place a lock on row 10, and **pradeep** wants to place a lock on row 20. If both users execute SET LOCK MODE TO WAIT, they potentially might wait for each other forever.

Informix® uses the lock table to detect deadlocks automatically and stop them before they occur. Before a lock is granted, the database server examines the lock list for each user. If a user holds a lock on the resource that the requestor wants to lock, the database server traverses the lock wait list for the user to see if the user is waiting for any locks that the requestor holds. If so, the requestor receives a deadlock error.

Deadlock errors can be unavoidable when applications update the same rows frequently. However, certain applications might always be in contention with each other. Examine applications that are producing a large number of deadlocks and try to run them at different times.

To monitor the number of deadlocks, use the **deadlks** field in the output of **onstat -p**.

In a distributed transaction, the database server does not examine lock tables from other database server systems, so deadlocks cannot be detected before they occur. Instead, you can set the DEADLOCK_TIMEOUT configuration parameter. DEADLOCK_TIMEOUT specifies the number of seconds that the database server waits for a remote database server response before it returns an error. Although reasons other than a distributed deadlock might cause the delay, this mechanism keeps a transaction from hanging indefinitely.

To monitor the number of distributed deadlock timeouts, use the **dltouts** field in the **onstat -p** output.

Monitoring isolation levels that sessions use

The **onstat -g ses** and **onstat -g sql** output shows the isolation level that a session is currently using.

The following table summarizes the values in the **IsoLvl** column in **onstat -g ses** and **onstat -g sql** output.

Value
: **Description**

DR
: Dirty Read

CR
: Committed Read

CS
: Cursor Stability

CRU
: Committed Read with RETAIN UPDATE LOCKS

CSU
: Cursor Stability with RETAIN UPDATE LOCKS

DRU
: Dirty Read with RETAIN UPDATE LOCKS

LC

Committed Read, Last Committed

RR
: Repeatable Read

If a great deal of lock contention occurs, check the isolation level of sessions to make sure it is appropriate for the application.

Locks for smart large objects

Smart large objects have several unique locking behaviors because their columns are typically much larger than other columns in a table.

This section discusses the following unique behaviors:

- Types of locks on smart large objects
- Byte-range locking
- Lock promotion
- Dirty Read isolation level with smart large objects

Types of locks on smart large objects

The database server uses one of the following granularity levels for locking smart large objects:

- The sbspace chunk header partition
- The smart large object
- A byte range of the smart large object

The default locking granularity is at the level of the smart large object. In other words, when you update a smart large object, by default the database server locks the smart large object that is being updated.

Locks on the sbspace chunk header partition only occur when the database server promotes locks on smart large objects. For more information, see Lock promotion.

- Byte-range locking
  Rather than locking the entire smart large object, you can lock only a specific byte range of a smart large object.
- Lock promotion
  The database server uses lock promotion to decrease the total number of locks held on smart large objects. Too many locks can result in poorer performance because the database server frequently searches the lock table to determine if a lock exists on an object.
- Dirty Read isolation level and smart large objects
  You can use the Dirty Read isolation level for smart large objects.

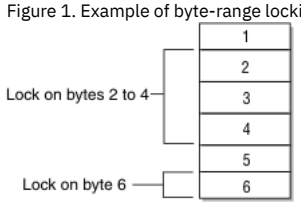
Related concepts:
Locks
Configuring the lock mode
Locks with the SELECT statement
Locks placed with INSERT, UPDATE, and DELETE statements
The internal lock table
Related tasks:
Setting the lock mode to wait
Related reference:
Monitoring locks

Byte-range locking

Rather than locking the entire smart large object, you can lock only a specific byte range of a smart large object.

Byte-range locking is advantageous because it allows multiple users to update the same smart large object simultaneously, as long as they are updating different parts of it. Also, users can read a part of a smart large object while another user is updating or reading a different part of the same smart large object.

Figure 1 shows two locks placed on a single smart large object. The first lock is on bytes 2, 3, and 4. The second lock is on byte 6 alone.
Figure 1. Example of byte-range locking



- How the database server manages byte-range locks
  The database server manages byte-range locks in the lock table in a similar fashion to other locks placed on rows, pages, and tables. However, the lock table must also store the byte range.
- Using byte-range locks
  By default, the database server places a lock on the smart large object. Instead, you can enable byte-range locking.

- Monitoring byte-range locks
  You can use **onstat -k** to list all byte-range locks. Use the **onstat -K** command to list byte-range locks and all waiters for byte-range locks.
- Setting number of locks for byte-range locking
  When you use byte-range locking, the database server can use more locks because of the possibility of multiple locks on one smart large object. Even though the lock table grows when it runs out of space, you might want to increase value of the LOCKS configuration parameter to match lock usage so that the database server does not need to allocate more space dynamically.

How the database server manages byte-range locks

The database server manages byte-range locks in the lock table in a similar fashion to other locks placed on rows, pages, and tables. However, the lock table must also store the byte range.

If you place a second lock on a byte range adjacent to a byte range that is currently locked, the database server consolidates the two locks into one lock on the entire range.

If a user holds locks that the Figure 1 shows, and the user requests a lock on byte five, the database server consolidates the locks placed on bytes two through six into one lock.

Likewise, if a user unlocks only a portion of the bytes included within a byte-range lock, the database server might be split into multiple byte-range locks. In the Figure 1 the user could unlock byte three, which causes the database server to change the one lock on bytes two through four to one lock on byte two and one lock on byte four.

Using byte-range locks

By default, the database server places a lock on the smart large object. Instead, you can enable byte-range locking.

To use byte-range locks, you must perform one of the following actions:

- To set byte-range locking for the sbspace that stores the smart large object, use the **onspaces** utility. The following example sets byte-range locking for the new sbspace:

  ```
  onspaces -c -S slo -g 2 -p /ix/9.2/liz/slo -o 0 -s 1000
     -Df LOCK_MODE=RANGE
  ```

  When you set the default locking mode for the sbspace to byte-range locking, the database server locks only the necessary bytes when it updates any smart large objects stored in the sbspace.

- To set byte-range locking for the smart large object when you open it, use one of the following methods:
  - *In DB-Access:* Set the MI_LO_LOCKRANGE flag in the **mi_lo_open()** DataBlade API function.
  - *In ESQL/C:* Set the LO_LOCKRANGE flag in the **ifx_lo_open()** Informix® ESQL/C function. When you set byte-range locking for the individual smart large object, the database server implicitly locks only the necessary bytes when it selects or updates the smart large object.
- To lock a byte range explicitly, use one of the following functions:
  - *For DB-Access:***mi_lo_lock()**
  - *For ESQL/C:***ifx_lo_lock()**
    These functions lock the range of bytes that you specify for the smart large object. If you specify an exclusive lock with either function, UPDATE statements do not place locks on the smart large object if they update the locked bytes.

    The database server releases exclusive byte-range locks placed with **mi_lo_lock()** or **ifx_lo_lock()** at the end of the transaction. The database server releases shared byte-range locks placed with **mi_lo_lock()** or **ifx_lo_lock()** based on the same rules as locks placed with SELECT statements, depending upon the isolation level. You can also release shared byte-range locks with one of the following functions:

  - *For DB-Access:***mi_lo_unlock()**. For more information about the DataBlade API functions, see the *IBM® Informix DataBlade API Programmer's Guide*.
  - *For ESQL/C:***ifx_lo_unlock()**. For more information about Informix ESQL/C functions, see the *IBM Informix ESQL/C Programmer's Manual*.

Monitoring byte-range locks

You can use **onstat -k** to list all byte-range locks. Use the **onstat -K** command to list byte-range locks and all waiters for byte-range locks.

Figure 1 shows an excerpt from the output of **onstat -k**.
Figure 1. Byte-range locks in onstat -k output

```
Byte-Range Locks
rowid/LOid   tblsnum   address  status   owner    offset   size     type
104          200004    a020e90  HDR
[2, 2, 3]              a020ee4  HOLD     a1b46d0  50       10          S
202          200004    a021034  HDR
[2, 2, 5]              a021088  HOLD     a1b51e0  40       5           S
102          200004    a035608  HDR
[2, 2, 1]              a0358fc  HOLD     a1b4148  0        500         S
                       a035758  HOLD     a1b3638  300      100         S
 21 active, 2000 total, 2048 hash buckets
```

Byte-range locks produce the following information in the **onstat -k** output.

Column	Description
rowid	The rowid of the row that contains the locked smart large object
LOid	The three values: sbspace number, chunk number, and sequence number (a value that represents the position in the chunk)
tblsnum	The number of the tblspace that holds the smart large object
address	The address of the lock
status	The status of the lock HDR is a placeholder. HOLD indicates the user specified in the **owner** column owns the lock. WAIT (shown only with **onstat -K**) indicates that the user specified in the owner column is waiting for the lock.
owner	The address of the owner (or waiter) Cross reference this value with the address in **onstat -u**.
offset	The offset into the smart large object where the bytes are locked
size	The number of bytes locked, starting at the value in the offset column
type	S (shared lock) or X (exclusive)

Setting number of locks for byte-range locking

When you use byte-range locking, the database server can use more locks because of the possibility of multiple locks on one smart large object. Even though the lock table grows when it runs out of space, you might want to increase value of the LOCKS configuration parameter to match lock usage so that the database server does not need to allocate more space dynamically.

Be sure to monitor the number of locks used with **onstat -k**, so you can determine if you need to increase the value of the LOCKS configuration parameter.

Lock promotion

The database server uses lock promotion to decrease the total number of locks held on smart large objects. Too many locks can result in poorer performance because the database server frequently searches the lock table to determine if a lock exists on an object.

If the number of locks held by a transaction exceeds 33 percent of the current number of allocated locks for the database server, the database server attempts to promote any existing byte-range locks to a single lock on the smart large object.

If the number of locks that a user holds on a smart large object (not on byte ranges of a smart large object) equals or exceeds 10 percent of the current capacity of the lock table, the database server attempts to promote all of the smart-large-object locks to one lock on the smart-large-object header partition. This kind of lock promotion improves performance for applications that are updating, loading, or deleting a large number of smart large objects. For example, a transaction that deletes millions of smart large objects would consume the entire lock table if the database server did not use lock promotion. The lock promotion algorithm has deadlock avoidance built in.

You can identify a smart-large-object header partition in **onstat -k** by `0` in the **rowid** column and a tablespace number with a high-order first byte-and-a-half that corresponds to the dbspace number where the smart large object is stored. For example, if the tblspace number is listed as `0x200004` (the high-order zeros are truncated), the dbspace number `2` corresponds to the dbspace number listed in **onstat -d**.

Even if the database server attempts to promote a lock, it might not be able to do so. For example, the database server might not be able to promote byte-range locks to one smart-large-object lock because other users have byte-range locks on the same smart large object. If the database server cannot promote a byte-range lock, it does not change the lock, and processing continues as normal.

Dirty Read isolation level and smart large objects

You can use the Dirty Read isolation level for smart large objects.

For information about how Dirty Reads affects consistency, see Dirty Read isolation.

Set the Dirty Read isolation level for smart large objects in one of the following ways:

- Use the SET TRANSACTION MODE or SET ISOLATION statement.
- Use the LO_DIRTY_READ flag in one of the following functions:
  - *For DB-Access:***mi_lo_open()**
  - *For ESQL/C:***ifx_lo_open()**

If consistency for smart large objects is not important, but consistency for other columns in the row is important, you can set the isolation level to Committed Read, Cursor Stability, or Repeatable Read and open the smart large object with the LO_DIRTY_READ flag.

Fragmentation guidelines

One of the most frequent causes of poor performance in relational database systems is contention for data that resides on a single I/O device. Proper fragmentation of high-use tables can significantly reduce I/O contention. These topics discuss the performance considerations that are involved when you use table fragmentation.

The database server supports table fragmentation (also *partitioning*), which allows you to store data from a single table on multiple disk devices. Fragmentation enables you to define groups of rows or index keys within a table according to some algorithm or scheme. You can store each group or fragment (also referred to as a *partition*) in a separate dbspace associated with a specific physical disk.

For information about fragmentation and parallel execution, see Parallel database query (PDQ).

For an introduction to fragmentation concepts and methods, see the *IBM Informix Database Design and Implementation Guide*. For information about the SQL statements that manage fragments, see the *IBM Informix Guide to SQL: Syntax*.

- Planning a fragmentation strategy
  You can decide on a fragmentation goal for your database and devise a strategy to meet that goal.
- Distribution schemes
  After you decide whether to fragment table rows, index keys, or both, and you decide how the rows and keys should be distributed over fragments, you can decide on a scheme to implement this distribution. Informix supports random distribution among fragments and value-based distribution among fragments.
- Strategy for fragmenting indexes
  When you fragment a table, the indexes that are associated with that table are fragmented implicitly, according to the distribution scheme that you use, except for the round-robin fragmentation scheme when automatic location is enabled. Indexes on tables that use the round-robin distribution scheme are not fragmented when the AUTOLOCATE configuration parameter or environment option is set to a positive integer. You can use the FRAGMENT BY clause of the CREATE INDEX statement to fragment the index on any table.
- Strategy for fragmenting temporary tables
  You can fragment an explicit temporary table across dbspaces that reside on different disks.
- Distribution schemes that eliminate fragments
  Fragment elimination is a database server feature that reduces the number of fragments involved in a database operation. This capability can improve performance significantly and reduce contention for the disks on which fragments reside.
- Improve the performance of operations that attach and detach fragments
  When you use ALTER FRAGMENT ATTACH and DETACH statements to add or remove a large amount of data in a very large table, you can take steps to improve the performance of the ATTACH and DETACH operations.
- Monitoring fragment use
  Once you determine a fragmentation strategy, you can monitor fragmentation.

Planning a fragmentation strategy

You can decide on a fragmentation goal for your database and devise a strategy to meet that goal.

A fragmentation strategy consists of two parts:

- A distribution scheme that specifies how to group rows into fragments
  You specify the distribution scheme in the FRAGMENT BY clause of the CREATE TABLE, CREATE INDEX, or ALTER FRAGMENT statements.
- The set of dbspaces in which you locate the fragments
  You specify the set of dbspaces or in the IN clause (storage option) of these SQL statements.

To formulate a fragmentation strategy:

1. Decide on your primary fragmentation goal, which should depend, to a large extent, on the types of applications that access the table.
2. Make the following decisions based on your primary fragmentation goal:
   - Whether to fragment the table data, the table index, or both
   - What the ideal distribution of rows or index keys is for the table
3. Choose either an expression-based or round-robin distribution scheme:
   - If you choose an expression-based distribution scheme, you must then design suitable fragment expressions.
   - If you choose a round-robin distribution scheme, the database server determines which rows to put into a specific fragment.

   For more information, see Distribution schemes.
4. To complete the fragmentation strategy, you must decide on the number and location of the fragments:
   - The number of fragments depends on your primary fragmentation goal.
   - Where you locate fragments depends on the number of disks available in your configuration.

When you plan a fragmentation strategy, be aware of these space and page issues:

- Although a 4-terabyte chunk can be on a 2-kilobyte page, only 32 gigabytes can be utilized in a dbspace because of a rowid format limitation.
- For a fragmented table, all fragments must use the same page size.
- For a fragmented index, all fragments must use the same page size.
- A table can be in one dbspace and the index for that table can be in another dbspace. These dbspaces do not need to have the same page size.

- Fragmentation goals
  You can analyze your application and workload to identify fragmentation goals and to determine the balance to strike among fragmentation goals.
- Examining your data and queries
  To determine a fragmentation strategy, you must gather information about the table that you might fragment. You must also know how the data in the table is used.
- Considering physical fragmentation factors
  When you fragment a table, the physical placement issues that pertain to tables apply to individual table fragments. Because each fragment resides in its own dbspace on a disk, you must address these issues separately for the fragments on each disk.

Related concepts:
Distribution schemes
Strategy for fragmenting indexes
Strategy for fragmenting temporary tables
Distribution schemes that eliminate fragments
Improve the performance of operations that attach and detach fragments
Monitoring fragment use

Fragmentation goals

You can analyze your application and workload to identify fragmentation goals and to determine the balance to strike among fragmentation goals.

Fragmentation goals can include:

- Improved performance for individual queries
  To improve the performance of individual queries, fragment tables appropriately and set resource-related parameters to specify system resource use (memory, CPU virtual processors, and so forth).
- Reduced contention between queries and between transactions
  If your database server is used primarily for online transaction processing (OLTP) and only incidentally for decision-support queries, you can often use fragmentation to reduce contention when simultaneous queries against the same table perform index scans to return a few rows.
- Increased data availability
  Careful fragmentation of dbspaces can improve data availability if devices fail. Table fragments on the failed device can be restored quickly, and other fragments are still accessible.
- Improved data-load performance
  When you use the High-Performance Loader (HPL) to load a table that is fragmented across multiple disks, it allocates threads to insert the data into the fragments in parallel, using light appends. For more information about this load method, see the *IBM® Informix High-Performance Loader User's Guide*.

  You can use the ALTER FRAGMENT ON TABLE statement with the ATTACH clause to add data quickly to a very large table. For more information, see Improve the performance of operations that attach and detach fragments.

The performance of a fragmented table is primarily governed by the following factors:

- The storage option that you use for allocating disk space to fragments (discussed in Considering physical fragmentation factors)
- The distribution scheme used to assign rows to individual fragments (discussed in Distribution schemes)

- Improved query performance through fragmentation strategy
  If the primary goal of fragmentation is improved performance for individual queries, try to distribute all of the rows of the table evenly over the different disks. Overall query-completion time is reduced when the database server does not need to wait for data retrieval from a table fragment that has more rows than other fragments.
- Reduced contention between queries and transactions
  Fragmentation can reduce contention for data in tables that multiple queries and OLTP applications use. Fragmentation often reduces contention when many simultaneous queries against a table perform index scans to return a few rows.
- Increased data availability
  When you distribute table and index fragments across different disks or devices, you improve the availability of data during disk or device failures. The database server continues to allow access to fragments stored on disks or devices that remain operational.
- Increased granularity for backup and restore
  You must consider backup and restore factors when you are deciding how to distribute dbspaces across disk.

Improved query performance through fragmentation strategy

If the primary goal of fragmentation is improved performance for individual queries, try to distribute all of the rows of the table evenly over the different disks. Overall query-completion time is reduced when the database server does not need to wait for data retrieval from a table fragment that has more rows than other fragments.

If queries access data by performing sequential scans against significant portions of tables, fragment the table rows only. Do not fragment the index. If an index is fragmented and a query has to cross a fragment boundary to access the data, the performance of the query can be worse than if you do not fragment.

If queries access data by performing an index read, you can improve performance by using the same distribution scheme for the index and the table.

If you use round-robin fragmentation, do not fragment your index. Consider placing that index in a separate dbspace from other table fragments.

For more information about improving performance for queries, see Query expressions for fragment elimination and Improving individual query performance.

Reduced contention between queries and transactions

Fragmentation can reduce contention for data in tables that multiple queries and OLTP applications use. Fragmentation often reduces contention when many simultaneous queries against a table perform index scans to return a few rows.

For tables subjected to this type of load, fragment both the index keys and data rows with a distribution scheme that allows each query to eliminate unneeded fragments from its scan. Use an expression-based distribution scheme. For more information, see Distribution schemes that eliminate fragments.

To fragment a table for reduced contention, start by investigating which queries access which parts of the table. Next, fragment your data so that some of the queries are routed to one fragment while others access a different fragment. The database server performs this routing when it evaluates the fragmentation rule for the table. Finally, store the fragments on separate disks.

Your success in reducing contention depends on how much you know about the distribution of data in the table and the scheduling of queries against the table. For example, if the distribution of queries against the table is set up so that all rows are accessed at roughly the same rate, try to distribute rows evenly across the fragments. However, if certain values are accessed at a higher rate than others, you can compensate for this difference by distributing the rows over the fragments to balance the access rate. For more information, see Designing an expression-based distribution scheme.

Increased data availability

When you distribute table and index fragments across different disks or devices, you improve the availability of data during disk or device failures. The database server continues to allow access to fragments stored on disks or devices that remain operational.

This availability has important implications for the following types of applications:

- Applications that do not require access to unavailable fragments
  A query that does not require the database server to access data in an unavailable fragment can still successfully retrieve data from fragments that are available. For example, if the distribution expression uses a single column, the database server can determine if a row is contained in a fragment without accessing the fragment. If the query accesses only rows that are contained in available fragments, a query can succeed even when some of the data in the table is unavailable. For more information, see Designing an expression-based distribution scheme.
- Applications that accept the unavailability of data
  Some applications might be designed in such a way that they can accept the unavailability of data in a fragment and require the ability to retrieve the data that is available. To specify which fragments can be skipped, these applications can execute the SET DATASKIP statement before they execute a query. Alternatively, the database server administrator can use the onspaces -f option to specify which fragments are unavailable.

If your fragmentation goal is increased availability of data, fragment both table rows and index keys so that if a disk drive fails, some of the data is still available. If applications must always be able to access a subset of your data, keep those rows together in the same mirrored dbspace.

Increased granularity for backup and restore

You must consider backup and restore factors when you are deciding how to distribute dbspaces across disk.

Backup and restore factors to consider are:

- **Data availability.** When you decide where to place your tables or fragments, remember that if a device that contains a dbspace fails, all tables or table fragments in that dbspace are inaccessible, even though tables and fragments in other dbspaces are accessible. The need to limit data unavailability in the event of a disk failure might influence which tables you group together in a particular dbspace.
- **Cold versus warm restores.** Although you must perform a cold restore if a dbspace that contains critical data fails, you need to perform only a warm restore if a noncritical dbspace fails. The desire to minimize the impact of cold restores might influence the dbspace that you use to store critical data.

For more information about backup and restore, see your *IBM® Informix® Backup and Restore Guide*.

Examining your data and queries

To determine a fragmentation strategy, you must gather information about the table that you might fragment. You must also know how the data in the table is used.

To gather information about your table:

1. Identify the queries that are critical to performance to determine if the queries are online transaction processing (OLTP) or decision-support system (DSS) queries.
2. Use the SET EXPLAIN statement to determine how the data is being accessed.
   For information about the output of the SET EXPLAIN statement, see Report that shows the query plan chosen by the optimizer. To determine how the data is accessed, you can sometimes simply review the SELECT statements along with the table schema.
3. Determine what portion of the data each query examines.
   For example, if certain rows in the table are read most of the time, you can isolate them in a small fragment to reduce I/O contention for other fragments.
4. Determine which statements create temporary files.
   Decision-support queries typically create and access large temporary files, and placement of temporary dbspaces can be critical to performance.
5. If particular tables are always joined together in a decision-support query, spread fragments for these tables across different disks.
6. Examine the columns in the table to determine which fragmentation scheme would keep each scan thread equally busy for the decision-support queries.
   To see how the column values are distributed, create a distribution on the column with the UPDATE STATISTICS statement and examine the distribution with **dbschema**.

```
dbschema -d database -hd table
```


Considering physical fragmentation factors

When you fragment a table, the physical placement issues that pertain to tables apply to individual table fragments. Because each fragment resides in its own dbspace on a disk, you must address these issues separately for the fragments on each disk.

For details about placement issues that apply to tables, see Table performance considerations.

Fragmented and nonfragmented tables differ in the following ways:

- For fragmented tables, each fragment is placed in a separate, designated dbspace or multiple named fragments of the table are created within a single dbspace. For nonfragmented tables, the table can be placed in the default dbspace of the current database.

  Regardless of whether the table is fragmented or not, you should create a single chunk on each disk for each dbspace.

- Extent sizes for a fragmented table are usually smaller than the extent sizes for an equivalent nonfragmented table because fragments do not grow in increments as large as the entire table. For more information on how to estimate the space to allocate, see Estimating table size.
- In a fragmented table, the row pointer is not a unique unchanging pointer to the row on a disk. The database server uses the combination of fragment ID and row pointer internally, inside an index, to point to the row. These two fields are unique but can change over the life of the row. An application cannot access the fragment ID; therefore, you should use primary keys to access a specific row in a fragmented table. For more information, see the *IBM® Informix® Database Design and Implementation Guide*.
- An attached index or an index on a nonfragmented table uses 4 bytes for the row pointer. A detached index uses 8 bytes of disk space per key value for the fragment ID and row pointer combination. For more information about how to estimate space for an index, see Estimating index pages. For more information on attached indexes and detached indexes, see Strategy for fragmenting indexes.

Decision-support queries usually create and access large temporary files; placement of temporary dbspaces is a critical factor for performance. For more information about placement of temporary files, see Spreading temporary tables and sort files across multiple disks.

Distribution schemes

After you decide whether to fragment table rows, index keys, or both, and you decide how the rows and keys should be distributed over fragments, you can decide on a scheme to implement this distribution. Informix® supports random distribution among fragments and value-based distribution among fragments.

Random distribution among fragments

Round-robin fragmentation
: This type of fragmentation places rows one after another in fragments, rotating through the series of fragments to distribute the rows evenly.
For smart large objects, you can specify multiple sbspaces in the PUT clause of the CREATE TABLE or ALTER TABLE statement to distribute smart large objects in a round-robin distribution scheme so that the number of smart large objects in each space is approximately equal.

Value-based distribution among fragments

Expression-based fragmentation
: This type of fragmentation puts rows that contain specified values in the same fragment. You specify a *fragmentation expression* that defines criteria for assigning a set of rows to each fragment, either as a range rule or some arbitrary rule.
You can specify a *remainder fragment* that holds all rows that do not match the criteria for any other fragment, although a remainder fragment reduces the efficiency of the expression-based distribution scheme.

List-based fragmentation
: This type of fragmentation puts rows that contain specified values that match one of the specified values in a list of discrete values in the same fragment. For each fragment, you specify a list of one or more constant expressions as *fragment expressions* that correspond to one or more columns in the table. The column or set of columns from which the *fragment expressions* are calculated is called the *fragment key*.
You can optionally specify a *remainder fragment* that holds all rows that do not match the criteria for any other fragment. You can also optionally specify a NULL fragment that stores rows with missing data in the fragment key columns (because its fragment expression is NULL or IS NULL).

  The most important difference between fragmentation by list and fragmentation by expression is that every value in the list for each fragment must be unique among all the lists for fragments of the same table or index.

Interval-based fragmentation
: This type of fragmentation partitions data into fragments that are based on quantified values within a specific interval within the range of fragment key of a single numeric, DATE, or DATETIME column in the same fragment. You specify at least one range expression as the *fragment expression* that defines the upper limit of fragment key values for each fragment, and an *interval expression* that specifies the size of the range of system-defined fragments that the database server creates automatically.
You can optionally define a NULL fragment to store rows with missing data in the fragment key column, but no *remainder fragment* is supported or needed. The database server automatically creates a new fragment to store rows with non-NULL fragment key values outside the range of any existing fragment. The fragments that you define with range expressions are called *range fragments*, and the system-defined fragments that the database server creates at runtime are called *interval fragments*. This type of distribution scheme is sometimes called a *range interval* distribution strategy.

- Choosing a distribution scheme
  When choosing a distribution scheme, you must consider the ease of data balancing, whether you want fragments to be eliminated, and the effect of the data skip

feature.

- Designing an expression-based distribution scheme
  The first step in designing an expression-based distribution scheme is to determine the distribution of data in the table, particularly the distribution of values for the column on which you base the fragmentation expression.
- Suggestions for improving fragmentation
  You can improve fragmentation for optimal performance in decision-support and OLTP queries.

Related concepts:
Strategy for fragmenting indexes
Strategy for fragmenting temporary tables
Distribution schemes that eliminate fragments
Improve the performance of operations that attach and detach fragments
Monitoring fragment use
Specify temporary tables in the DBSPACETEMP configuration parameter
Related tasks:
Planning a fragmentation strategy
Related information:
List fragmentation clause
Interval fragment clause
Fragmentation: Storage distribution strategies

Choosing a distribution scheme

When choosing a distribution scheme, you must consider the ease of data balancing, whether you want fragments to be eliminated, and the effect of the data skip feature.

Table 1 compares round-robin and expression-based distribution schemes.

Table 1. Distribution-Scheme Comparisons

Distribution Scheme	Ease of Data Balancing	Fragment Elimination	Data Skip
Round-robin	Automatic. Data is balanced over time.	The database server cannot eliminate fragments.	You cannot determine if the integrity of the transaction is compromised when you use the data-skip feature. However, you can insert into a table fragmented by round-robin.
Expression-based	Requires knowledge of the data distribution.	If expressions on one or two columns are used, the database server can eliminate fragments for queries that have either range or equality expressions.	You can determine whether the integrity of a transaction has been compromised when you use the data-skip feature. You cannot insert rows if the appropriate fragment for those rows is down.

The distribution scheme that you choose depends on the following factors:

- The features in Table 1 of which you want to take advantage
- Whether or not your queries tend to scan the entire table
- Whether or not you know the distribution of data to be added
- Whether or not your applications tend to delete many rows
- Whether or not you cycle your data through the table

Basically, the round-robin scheme provides the easiest and surest way of balancing data. However, with round-robin distribution, you have no information about the fragment in which a row is located, and the database server cannot eliminate fragments.

In general, round-robin is the correct choice only when all the following conditions apply:

- Your queries tend to scan the entire table.
- You do not know the distribution of data to be added.
- Your applications tend not to delete many rows. (If they do, load balancing can be degraded.)

An expression-based scheme might be the best choice to fragment the data if any of the following conditions apply:

- Your application calls for numerous decision-support queries that scan specific portions of the table.
- You know what the data distribution is.
- You plan to cycle data through a database.

If you plan to add and delete large amounts of data periodically, based on the value of a column such as date, you can use that column in the distribution scheme. You can then use the alter fragment attach and alter fragment detach statements to cycle the data through the table.

The ALTER FRAGMENT ATTACH and DETACH statements provide the following advantages over bulk loads and deletes:

- The rest of the table fragments are available for other users to access. Only the fragment that you attach or detach is not available to other users.
- With the performance enhancements, the execution of an ALTER FRAGMENT ATTACH or DETACH statement is much faster than a bulk load or mass delete.

For more information, see Improve the performance of operations that attach and detach fragments.

In some cases, an appropriate index scheme can circumvent the performance problems of a particular distribution scheme. For more information, see Strategy for fragmenting indexes.

Designing an expression-based distribution scheme

The first step in designing an expression-based distribution scheme is to determine the distribution of data in the table, particularly the distribution of values for the column on which you base the fragmentation expression.

To obtain this information, run the UPDATE STATISTICS statement for the table and then use the **dbschema** utility to examine the distribution.

After you know the data distribution, you can design a fragmentation rule that distributes data across fragments as required to meet your fragmentation goal. If your primary goal is to improve performance, your fragment expression should generate an even distribution of rows across fragments.

If your primary fragmentation goal is improved concurrency, analyze the queries that access the table. If certain rows are accessed at a higher rate than others, you can compensate by opting for an uneven distribution of data over the fragments that you create.

Try not to use columns that are subject to frequent updates in the distribution expression. Such updates can cause rows to move from one fragment to another (that is, be deleted from one and added to another), and this activity increases CPU and I/O overhead.

Try to create nonoverlapping regions based on a single column with no REMAINDER fragment for the best fragment-elimination characteristics. The database server eliminates fragments from query plans whenever the query optimizer can determine that the values selected by the WHERE clause do not reside on those fragments, based on the expression-based fragmentation rule by which you assign rows to fragments. For more information, see Distribution schemes that eliminate fragments.

Suggestions for improving fragmentation

You can improve fragmentation for optimal performance in decision-support and OLTP queries.

The following suggestions are guidelines for fragmenting tables and indexes:

- For optimal performance in decision-support queries, fragment the table to increase parallelism, but do not fragment the indexes. Detach the indexes, and place them in a separate dbspace.
- For best performance in OLTP queries, use fragmented indexes to reduce contention between sessions. You can often fragment an index by its key value, which means the OLTP query only has to look at one fragment to find the location of the row.
  If the key value does not reduce contention, as when every user looks at the same set of key values (for instance, a date range), consider fragmenting the index on another value used in the WHERE clause. To cut down on fragment administration, consider not fragmenting some indexes, especially if you cannot find a good fragmentation expression to reduce contention.
- Use round-robin fragmentation on data when the table is read sequentially by decision-support queries. Round-robin fragmentation is a good method for spreading data evenly across disks when no column in the table can be used for an expression-based fragmentation scheme. However, in most DSS queries, all fragments are read.
- To reduce the total number of required dbspaces and decrease the time needed for searches, you can store multiple named fragments within the same dbspace.
- If you are using expressions, create them so that I/O requests, rather than quantities of data, are balanced across disks. For example, if the majority of your queries access only a portion of data in the table, set up your fragmentation expression to spread active portions of the table across disks, even if this arrangement results in an uneven distribution of rows.
- Keep fragmentation expressions simple. Fragmentation expressions can be as complex as you want. However, complex expressions take more time to evaluate and might prevent fragments from being eliminated from queries.
- Arrange fragmentation expressions so that the most restrictive condition for each dbspace is tested within the expression first. When the database server tests a value against the criteria for a given fragment, evaluation stops when a condition for that fragment tests false. Thus, if the condition that is most likely to be false is placed first, fewer conditions need to be evaluated before the database server moves to the next fragment. For example, in the following expression, the database server tests all six of the inequality conditions when it attempts to insert a row with a value of `25`:

  ```
  x >= 1 and x <= 10 in dbspace1,
  x > 10 and x <= 20 in dbspace2,
  x > 20 and x <= 30 in dbspace3
  ```

  By comparison, only four conditions in the following expression need to be tested: the first inequality for **dbspace1** `(x <= 10)`, the first for **dbspace2** `(x <= 20)`, and both conditions for **dbspace3**:

  ```
  x <= 10 and x >= 1 in dbspace1,
  x <= 20 and x > 10 in dbspace2,
  x <= 30 and x > 20 in dbspace3
  ```

- Avoid any expression that requires a data-type conversion. Type conversions increase the time to evaluate the expression. For instance, a DATE data type is implicitly converted to INTEGER for comparison purposes.
- Do not fragment on columns that change frequently unless you are willing to incur the administration costs. For example, if you fragment on a date column and older rows are deleted, the fragment with the oldest dates tends to empty, and the fragment with the recent dates tends to fill up. Eventually you must drop the old fragment and add a new fragment for newer orders.
- Do not fragment every table. Identify the critical tables that are accessed most frequently. Put only one fragment for a table on a disk.
- Do not fragment small tables. Fragmenting a small table across many disks might not be worth the overhead of starting all the scan threads to access the fragments. Also, balance the number of fragments with the number of processors on your system.
- When you define a fragmentation strategy on an unfragmented table, check the next-extent size to ensure that you are not allocating large amounts of disk space for each fragment.

Strategy for fragmenting indexes

When you fragment a table, the indexes that are associated with that table are fragmented implicitly, according to the distribution scheme that you use, except for the round-robin fragmentation scheme when automatic location is enabled. Indexes on tables that use the round-robin distribution scheme are not fragmented when the AUTOLOCATE configuration parameter or environment option is set to a positive integer. You can use the FRAGMENT BY clause of the CREATE INDEX statement to fragment the index on any table.

Each index of a fragmented table occupies its own tblspace with its own extents.

You can fragment the index with either of the following strategies:

- Same fragmentation strategy as the table
- Different fragmentation strategy from the table

- Attached indexes
  An *attached index* is an index that implicitly follows the table fragmentation strategy (distribution scheme and set of dbspaces in which the fragments are located). When you create an index on a fragmented table, the index is an attached index, unless you use the round-robin distribution scheme and automatic location is enabled. Indexes on tables that use the round-robin distribution scheme are not fragmented when the AUTOLOCATE configuration parameter or environment option is set to a positive integer.
- Detached indexes
  A *detached index* is an index with a separate fragmentation strategy that you set up explicitly with the CREATE INDEX statement.
- Restrictions on indexes for fragmented tables
  If the database server scans a fragmented index, multiple index fragments must be scanned and the results merged together. (The exception is if the index is fragmented according to some index-key range rule, and the scan does not cross a fragment boundary.) Because of this requirement, performance on index scans might suffer if the index is fragmented.

Related concepts:
Distribution schemes
Strategy for fragmenting temporary tables
Distribution schemes that eliminate fragments
Improve the performance of operations that attach and detach fragments
Monitoring fragment use
Related tasks:
Planning a fragmentation strategy

Attached indexes

An *attached index* is an index that implicitly follows the table fragmentation strategy (distribution scheme and set of dbspaces in which the fragments are located). When you create an index on a fragmented table, the index is an attached index, unless you use the round-robin distribution scheme and automatic location is enabled. Indexes on tables that use the round-robin distribution scheme are not fragmented when the AUTOLOCATE configuration parameter or environment option is set to a positive integer.

To create an attached index, do not specify a fragmentation strategy or storage option in the CREATE INDEX statement, as in the following sample SQL statements:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN dbsbspace1,
      (a >=5 AND a < 10) IN dbspace2
      ...
   ;

CREATE INDEX idx1 ON tb1(a);
```

For fragmented tables that use expression-based or round-robin distribution schemes, you can also create multiple partitions of a table or index within a single dbspace. This enables you to reduce the number of required dbspaces, thereby simplifying the management of dbspaces.

To create an attached index with partitions, include the partition name in your SQL statements, as shown in this example:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
     PARTITION part1 (a >=0 AND a < 5) IN dbs1,
     PARTITION part2 (a >=5 AND a < 10) IN dbs1
          ...
 ;

    CREATE INDEX idx1 ON tb1(a);
```

You can use "PARTITION BY EXPRESSION" instead of "FRAGMENT BY EXPRESSION" in CREATE TABLE, CREATE INDEX, and ALTER FRAGMENT ON INDEX statements as shown in this example:

```
ALTER FRAGMENT ON INDEX idx1 INIT PARTITION BY EXPRESSION
           PARTITION part1   (a <= 10) IN dbs1,
           PARTITION part2   (a <= 20) IN dbs1,
            PARTITION part3  (a <= 30) IN dbs1;
```

Use ALTER FRAGMENT syntax to change fragmented indexes that do not have partitions into indexes that have partitions. The syntax below shows how you might convert a fragmented index into an index that contains partitions:

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
CREATE INDEX ind1 ON t1 (c1) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3

ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
     PARTITION part_1 (c1=10) IN dbs1, PARTITION part_2 (c1=20) IN dbs1,
```

```
    PARTITION part_3 (c1=30) IN dbs1,
```

Creating a table or index containing partitions improves performance by enabling the database server to search more quickly and by reducing the required number of dbspaces.

The database server fragments the attached index according to the same distribution scheme as the table by using the same rule for index keys as for table data. As a result, attached indexes have the following physical characteristics:

- The number of index fragments is the same as the number of data fragments.
- Each attached index fragment resides in the same dbspace as the corresponding table data, but in a separate tblspace.
- An attached index or an index on a nonfragmented table uses 4 bytes for the row pointer for each index entry. For more information about how to estimate space for an index, see Estimating index pages.

Informix® does not support forest of trees attached indexes.

Detached indexes

A *detached index* is an index with a separate fragmentation strategy that you set up explicitly with the CREATE INDEX statement.

The following sample SQL statements create a detached index:

```
CREATE TABLE tb1 (a int)
      FRAGMENT BY EXPRESSION
         (a <= 10) IN tabdbspc1,
         (a <= 20) IN tabdbspc2,
         (a <= 30) IN tabdbspc3;

CREATE INDEX idx1 ON tb1 (a)
      FRAGMENT BY EXPRESSION
         (a <= 10) IN idxdbspc1,
         (a <= 20) IN idxdbspc2,
         (a <= 30) IN idxdbspc3;
```

This example illustrates a common fragmentation strategy, to fragment indexes in the same way as the tables, but specify different dbspaces for the index fragments. This fragmentation strategy of putting the index fragments in different dbspaces from the table can improve the performance of operations such as backup, recovery, and so forth.

By default, all new indexes that the CREATE INDEX statement creates are detached and stored in separate tablespaces from the data unless the deprecated IN TABLE syntax is specified.

To create a detached index with partitions, include the partition name in your SQL statements, as shown in this example:

```
CREATE TABLE tb1 (a int)
      FRAGMENT BY EXPRESSION
            PARTITION part1 (a <= 10) IN dbs1,
            PARTITION part2 (a <= 20) IN dbs2,
            PARTITION part3 (a <= 30) IN dbs3;

    CREATE INDEX idx1 ON tb1 (a)
          FRAGMENT BY EXPRESSION
            PARTITION part1   (a <= 10) IN dbs1,
            PARTITION part2   (a <= 20) IN dbs2,
            PARTITION part3   (a <= 30) IN dbs3;
```

You can use the `PARTITION BY EXPRESSION` keywords instead of the `FRAGMENT BY EXPRESSION` keywords in the CREATE TABLE, CREATE INDEX, and ALTER FRAGMENT ON INDEX statements.

If you do not want to fragment the index, you can put the entire index in a separate dbspace.

You can fragment the index for any table by expression. However, you cannot explicitly create a round-robin fragmentation scheme for an index. Whenever you fragment a table using a round-robin fragmentation scheme, convert all indexes that accompany the table to detached indexes for the best performance.

Detached indexes have the following physical characteristics:

- Each detached index fragment resides in a different tblspace from the corresponding table data. Therefore, the data and index pages cannot be interleaved within the tblspace.
- Detached index fragments have their own extents and *tblspace IDs*. The tblspace ID is also known as the *fragment* ID and *partition number*. A detached index uses 8 bytes of disk space per index entry for the fragment ID and row pointer combination. For more information on how to estimate space for an index, see Estimating index pages.

Forest of trees indexes are detached indexes. They cannot be attached indexes.

The database server stores the location of each table and index fragment, along with other related information, in the **sysfragments** system catalog table. You can view the **sysfragments** system catalog table to access information about fragmented tables and indexes, including the following :

- The value in the **partn** column is the partition number or fragment id of the table or index fragment. The partition number for a detached index is different from the partition number of the corresponding table fragment.
- The value in the **strategy** column is the distribution scheme used in the fragmentation strategy.

For a complete description of column values that the **sysfragments** system catalog table contains, see the *IBM® Informix® Guide to SQL: Reference*. For information about how to use **sysfragments** to monitor your fragments, see Monitoring fragment use.

Restrictions on indexes for fragmented tables

If the database server scans a fragmented index, multiple index fragments must be scanned and the results merged together. (The exception is if the index is fragmented according to some index-key range rule, and the scan does not cross a fragment boundary.) Because of this requirement, performance on index scans might suffer if the index is fragmented.

Because of these performance considerations, the database server places the following restrictions on indexes:

- You cannot fragment indexes by round-robin.
- You cannot fragment unique indexes by an expression that contains columns that are not in the index key.

For example, the following statement is not valid:

```
CREATE UNIQUE INDEX ia on tab1(col1)
   FRAGMENT BY EXPRESSION
      col2<10 in dbsp1,
      col2>=10 AND col2<100 in dbsp2,
      col2>100 in dbsp3;
```


Strategy for fragmenting temporary tables

You can fragment an explicit temporary table across dbspaces that reside on different disks.

You can create a temporary, fragmented table with the TEMP TABLE clause of the CREATE TABLE statement. However, you cannot alter the fragmentation strategy of fragmented temporary tables (as you can with permanent tables). The database server deletes the fragments that are created for a temporary table at the same time that it deletes the temporary table.

You can define your own fragmentation strategy for an explicit temporary table, or you can let the database server dynamically determine the fragmentation strategy.

For more information about explicit and implicit temporary tables, see your *IBM® Informix Administrator's Guide*.

Related concepts:
Distribution schemes
Strategy for fragmenting indexes
Distribution schemes that eliminate fragments
Improve the performance of operations that attach and detach fragments
Monitoring fragment use
Related tasks:
Planning a fragmentation strategy

Distribution schemes that eliminate fragments

Fragment elimination is a database server feature that reduces the number of fragments involved in a database operation. This capability can improve performance significantly and reduce contention for the disks on which fragments reside.

Fragment elimination improves both response time for a given query and concurrency between queries. Because the database server does not need to read in unnecessary fragments, I/O for a query is reduced. Activity in the LRU queues is also reduced.

If you use an appropriate distribution scheme, the database server can eliminate fragments from the following database operations:

- The fetch portion of the SELECT, INSERT, delete or update statements in SQL
  The database server can eliminate fragments when these SQL statements are optimized, before the actual search.

- Nested-loop joins
  When the database server obtains the key value from the outer table, it can eliminate fragments to search on the inner table.

Whether the database server can eliminate fragments from a search depends on two factors:

- The distribution scheme in the fragmentation strategy of the table that is being searched
- The form of the query expression (the expression in the WHERE clause of a SELECT, INSERT, delete or update statement)

- Fragmentation expressions for fragment elimination
  Some operators in expressions result in automatic fragment elimination.
- Query expressions for fragment elimination
  A query expression (the expression in the WHERE clause) can consist of simple expressions, not simple expressions, and multiple expressions.
- Effectiveness of fragment elimination
  The database server cannot eliminate fragments when you fragment a table with a round-robin distribution scheme. Furthermore, not all expression-based distribution schemes give you the same fragment-elimination behavior.

Related concepts:
Distribution schemes
Strategy for fragmenting indexes
Strategy for fragmenting temporary tables

Improve the performance of operations that attach and detach fragments
Monitoring fragment use
Related tasks:
Planning a fragmentation strategy

Fragmentation expressions for fragment elimination

Some operators in expressions result in automatic fragment elimination.

When the fragmentation strategy is defined with any of the following operators, fragment elimination can occur for a query on the table.

```
IN
=
<
>
<=
>=
AND
OR
NOT
IS NULL (only when not combined with other expressions using AND or OR operators)
```

If the fragmentation expression uses any of the following operators, fragment elimination does not occur for queries on the table.

```
!=
IS NOT NULL
```

For examples of fragmentation expressions that allow fragment elimination, see Effectiveness of fragment elimination.

Query expressions for fragment elimination

A query expression (the expression in the WHERE clause) can consist of simple expressions, not simple expressions, and multiple expressions.

The database server considers only simple expressions or multiple simple expressions combined with certain operators for fragment elimination.

A simple expression consists of the following parts:

```
column operator value
```

Simple Expression Part
Description

column
Is a single column name
The database server supports fragment elimination on all column types except columns that are defined with the NCHAR, NVARCHAR, BYTE, and TEXT data types.

operator
Must be an equality or range operator

value
Must be a literal or a host variable

The following examples show simple expressions:

```
name = "Fred"
date < "08/25/2008"
value >= :my_val
```

The following examples are not simple expressions:

```
unitcost * count > 4500
price <= avg(price)
result + 3 > :limit
```

The database server considers two types of simple expressions for fragment elimination, based on the operator:

- Range expressions
- Equality expressions

- Range expressions in query
  The database server can handle one or two column fragment elimination on queries with any combination of five relational operators in the WHERE clause.
- Equality expressions in query
  The database server can handle one or multiple column fragment elimination on queries with a combination of equality operators in the WHERE clause.

Range expressions in query

The database server can handle one or two column fragment elimination on queries with any combination of five relational operators in the WHERE clause.

Range expressions use the following relational operators:

```
<
>
<=
>=
!=
```

The database server can also eliminate fragments when these range expressions are combined with the following operators:

```
AND, OR, NOT
IS NULL, IS NOT NULL
MATCH, LIKE
```

If the range expression contains MATCH or LIKE, the database server can also eliminate fragments if the string does not begin with a wildcard character. The following examples show query expressions that can take advantage of fragment elimination:

```
columna MATCH "ab*"
columna LIKE "ab%" OR columnb LIKE "ab*"
```


Equality expressions in query

The database server can handle one or multiple column fragment elimination on queries with a combination of equality operators in the WHERE clause.

Equality expressions use the following equality operators:

```
=, IN
```

The database server can also eliminate fragments when these equality expressions are combined with the following operators:

```
AND, OR
```


Effectiveness of fragment elimination

The database server cannot eliminate fragments when you fragment a table with a round-robin distribution scheme. Furthermore, not all expression-based distribution schemes give you the same fragment-elimination behavior.

The following table summarizes the fragment-elimination behavior for different combinations of expression-based distribution schemes and query expressions. Partitions in fragmented tables do not affect the fragment-elimination behavior shown in the following table.

Table 1. Fragment elimination for different types of expression-based distribution schemes and query expressions

Type of Query (WHERE clause) Expression	Nonoverlapping Fragments on a Single Column	Overlapping or Non-contiguous Fragments on a Single Column	Nonoverlapping Fragments on Multiple Columns
Range expression	Fragments can be eliminated.	Fragments cannot be eliminated.	Fragments cannot be eliminated.
Equality expression	Fragments can be eliminated.	Fragments can be eliminated.	Fragments can be eliminated.

This table shows that the distribution schemes enable fragment elimination, but the effectiveness of fragment elimination is determined by the WHERE clause of the specified query.

For example, consider a table fragmented with the following expression:

```
FRAGMENT BY EXPRESSION
100 < column_a AND column_b < 0 IN dbsp1,
100 >= column_a AND column_b < 0 IN dbsp2,
column_b >= 0 IN dbsp3
```

The database server cannot eliminate any fragments from the search if the WHERE clause has the following expression:

```
column_a = 5 OR column_b = -50
```

However, the database server can eliminate the fragment in dbspace **dbsp3** if the WHERE clause has the following expression:

```
column_b = -50
```

Furthermore, the database server can eliminate the two fragments in dbspaces **dbsp2** and **dbsp3** if the WHERE clause has the following expression:

```
column_a = 5 AND column_b = -50
```

Partitions in fragmented tables do not affect fragment-elimination behavior.

- Nonoverlapping fragments on a single column
  A fragmentation rule that creates nonoverlapping fragments on a single column is the preferred fragmentation rule from a fragment-elimination standpoint.
- Overlapping fragments on a single column
  The fragments on a single column can be overlapping and noncontiguous. You can use any range, MOD function, or arbitrary rule that is based on a single column.
- Nonoverlapping fragments, multiple columns
  The database server uses an arbitrary rule to define nonoverlapping fragments based on multiple columns.

Nonoverlapping fragments on a single column

A fragmentation rule that creates nonoverlapping fragments on a single column is the preferred fragmentation rule from a fragment-elimination standpoint.

The advantage of this type of distribution scheme is that the database server can eliminate fragments for queries with range expressions as well as queries with equality expressions. You should meet these conditions when you design your fragmentation rule. Figure 1 gives an example of this type of fragmentation rule.
Figure 1. Example of nonoverlapping fragments on a single column

```
...
FRAGMENT BY EXPRESSION
a<=8 OR a IN (9,10) IN dbsp1,
10<a AND a<=20 IN dbsp2,
a IN (21,22,23) IN dbsp3,
a>23 IN dbsp4;
```

You can create nonoverlapping fragments using a range rule or an arbitrary rule based on a single column. You can use relational operators, as well as AND, IN, OR, or BETWEEN. Be careful when you use the BETWEEN operator. When the database server parses the BETWEEN keyword, it includes the end points that you specify in the range of values. Avoid using a REMAINDER clause in your expression. If you use a REMAINDER clause, the database server cannot always eliminate the remainder fragment.

Overlapping fragments on a single column

The fragments on a single column can be overlapping and noncontiguous. You can use any range, MOD function, or arbitrary rule that is based on a single column.

The only restriction for this category of fragmentation rule is that you base the fragmentation rule on a single column.

Figure 1 shows an example of this type of fragmentation rule.
Figure 1. Example of overlapping fragments on a single column

```
...
FRAGMENT BY EXPRESSION
a<=8 OR a IN (9,10,21,22,23) IN dbsp1,
a>10 IN dbsp2;
```

If you use this type of distribution scheme, the database server can eliminate fragments on an equality search but not a range search. This distribution scheme can still be useful because all INSERT and many UPDATE operations perform equality searches.

This alternative is acceptable if you cannot use an expression that creates nonoverlapping fragments with contiguous values. For example, in cases where a table is growing over time, you might want to use a MOD function rule to keep the fragments of similar size. Expression-based distribution schemes that use MOD function rules fall into this category because the values in each fragment are not contiguous.

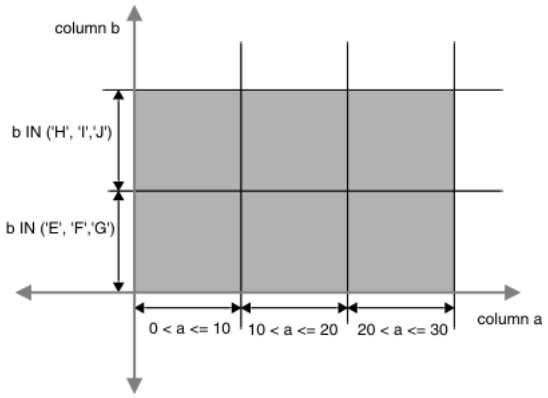
Nonoverlapping fragments, multiple columns

The database server uses an arbitrary rule to define nonoverlapping fragments based on multiple columns.

The following figures show an example of nonoverlapping fragments on two columns.
Figure 1. Example of nonoverlapping fragments on two columns

```
...
FRAGMENT BY EXPRESSION
0<a AND a<=10 AND b IN ('E', 'F', 'G') IN dbsp1,
0<a AND a<=10 AND b IN ('H', 'I', 'J') IN dbsp2,
10<a AND a<=20 AND b IN ('E', 'F', 'G') IN dbsp3,
10<a AND a<=20 AND b IN ('H', 'I', 'J') IN dbsp4,
20<a AND a<=30 AND b IN ('E', 'F', 'G') IN dbsp5,
20<a AND a<=30 AND b IN ('H', 'I', 'J') IN dbsp6;
```

Figure 2. Schematic example of nonoverlapping fragments on two columns



If you use this type of distribution scheme, the database server can eliminate fragments on an equality search but not a range search. This capability can still be useful because all INSERT operations and many UPDATE operations perform equality searches. Avoid using a REMAINDER clause in the expression. If you use a REMAINDER clause, the database server cannot always eliminate the remainder fragment.

This alternative is acceptable if you cannot obtain sufficient granularity using an expression based on a single column.

Improve the performance of operations that attach and detach fragments

When you use ALTER FRAGMENT ATTACH and DETACH statements to add or remove a large amount of data in a very large table, you can take steps to improve the performance of the ATTACH and DETACH operations.

The ALTER FRAGMENT DETACH statement provides a way to delete a segment of the table data rapidly. Similarly, the ALTER FRAGMENT ATTACH statement provides a way to load large amounts of data incrementally into an existing table by taking advantage of the fragmentation technology. However, the ALTER FRAGMENT ATTACH and ALTER FRAGMENT DETACH statements can take a long time to execute when the database server rebuilds indexes on the surviving table.

The database server provides performance optimizations for the ATTACH and DETACH operations of the ALTER FRAGMENT statement that reuse the indexes on the surviving tables. By eliminating the index build during the ATTACH or DETACH operation,

- this reduces the time required for the ALTER FRAGMENT ATTACH and ALTER FRAGMENT DETACH statements to execute,
- and improves the availability of the table.

The ALTER FRAGMENT operation requires exclusive access and exclusive locks on all of the tables involved in the operation. When you use the FORCE_DDL_EXEC environment option to specify a time limit for the database server to force out any transactions in other sessions that have opened (or that hold locks on) the tables involved in an ALTER FRAGMENT ON TABLE operation, also use the SET LOCK MODE TO WAIT statement to specify that number of seconds as the limit for waiting.

If the database server is unable to get exclusive access and exclusive locks on the table because of DDL transactions in concurrent sessions, the server will start rolling back the transactions that are open or that have locks on the table, until the specified time limit is reached. You might want to enable the FORCE_DDL_EXEC option and issue the SET LOCK MODE TO WAIT statement on a busy system, perhaps one that runs 24 hours a day, if you do not want to wait for transactions in concurrent sessions to close before you can alter a fragment.

- Improving ALTER FRAGMENT ATTACH performance
  You can take advantage of the performance optimizations for the ALTER FRAGMENT ATTACH statement if your database meets certain requirements.
- Improving ALTER FRAGMENT DETACH performance
  You can improve the performance of ALTER FRAGMENT DETACH statements by formulating appropriate distribution schemes for your table and index fragments and by eliminating the index build during the execution of ALTER FRAGMENT DETACH statements.
- Forcing out transactions when altering table fragments
  You can enable the server to force out transactions that have opened or hold locks on the target table of an ALTER FRAGMENT ON TABLE operation in a logging database. Users holding the DBA access privilege can do this by enabling the FORCE_DDL_EXEC session environment option of the SET ENVIRONMENT statement.

Related concepts:
Distribution schemes
Strategy for fragmenting indexes
Strategy for fragmenting temporary tables
Distribution schemes that eliminate fragments
Monitoring fragment use
Related tasks:
Planning a fragmentation strategy

Improving ALTER FRAGMENT ATTACH performance

You can take advantage of the performance optimizations for the ALTER FRAGMENT ATTACH statement if your database meets certain requirements.

To take advantage of the performance optimization, you must meet all of the following requirements:

- Formulate appropriate distribution schemes for your table and index fragments.
- Ensure that no data movement occurs between the resultant partitions due to fragment expressions.

- Update statistics for all the participating tables.
- Make the indexes on the attached tables unique if the index on the surviving table is unique.

Important: Only logging databases can benefit from the performance improvements for the ALTER FRAGMENT ATTACH statement. Without logging, the database server works with multiple copies of the same table to ensure recoverability of the data when a failure occurs. This requirement prevents reuse of the existing index fragments.

- Distribution schemes for reusing indexes
  You can use one of three distribution schemes that allow the attach operation of the ALTER FRAGMENT statement to reuse existing indexes.
- Ensuring no data movement when you attach a fragment
  You can ensure there is no data movement when you attach a fragment by establishing identical check constraint expressions and verifying that fragment expressions are not overlapping.
- Indexes on attached tables
  The database server tries to reuse the indexes on the attached tables as fragments of the resultant index. However, the corresponding index on the attached table might not exist or might not be usable due to disk-format mismatches. In these cases, it might be faster to build an index on the attached tables rather than to build the entire index on the resultant table.

Distribution schemes for reusing indexes

You can use one of three distribution schemes that allow the attach operation of the ALTER FRAGMENT statement to reuse existing indexes.

These distributions schemes are:

- Fragmenting the index in the same way as the table
- Fragmenting the index with the same set of fragment expressions as the table
- Attaching unfragmented tables to form a fragmented table

- Fragmenting the index in the same way as the table
  You fragment an index in the same way as the table when you create an index without specifying a fragmentation strategy.
- Fragmenting the index with the same distribution scheme as the table
  You fragment an index with the same distribution scheme as the table when you create an index that uses the same fragment expressions as the table.
- Attaching unfragmented tables together
  You can take advantage of the performance benefits of the ALTER FRAGMENT ATTACH operation when you combine two unfragmented tables into one fragmented table.

Fragmenting the index in the same way as the table

You fragment an index in the same way as the table when you create an index without specifying a fragmentation strategy.

A fragmentation strategy is the distribution scheme and set of dbspaces in which the fragments are located. For details, see Planning a fragmentation strategy.

Example of Fragmenting the Index in the Same Way as the Table

Suppose you create a fragmented table and index with the following SQL statements:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;

CREATE INDEX idx1 ON tb1(a);
```

Suppose you then create another table that is not fragmented, and you subsequently decide to attach it to the fragmented table.

```
CREATE TABLE tb2 (a int, CHECK (a >=10 AND a<15))
   IN db3;

CREATE INDEX idx2 ON tb2(a)
   IN db3;

ALTER FRAGMENT ON TABLE tb1
      ATTACH
         tb2 AS (a >= 10 and a<15) AFTER db2;
```

This attach operation can take advantage of the existing index **idx2** if no data movement occurs between the existing and the new table fragments. If no data movement occurs:

- The database server reuses index **idx2** and converts it to a fragment of index **idx1**.
- The index **idx1** remains as an index with the same fragmentation strategy as the table **tb1**.

If the database server discovers that one or more rows in the table **tb2** belong to preexisting fragments of the table **tb1**, the database server:

- Drops and rebuilds the index **idx1** to include the rows that were originally in tables **tb1** and **tb2**
- Drops the index **idx2**

For more information about how to ensure no data movement between the existing and the new table fragments, see Ensuring no data movement when you attach a fragment.

Fragmenting the index with the same distribution scheme as the table

You fragment an index with the same distribution scheme as the table when you create an index that uses the same fragment expressions as the table.

The database server determines if the fragment expressions are identical, based on the equivalency of the expression tree instead of the algebraic equivalence. For example, consider the following two expressions:

```
(col1 >= 5)
(col1 = 5 OR col1 > 5)
```

Although these two expressions are algebraically equivalent, they are not identical expressions.

Example of Fragmenting the Index with the Same Distribution Scheme as the Table

Suppose you create two fragmented tables and indexes with the following SQL statements:

```
CREATE TABLE tb1 (a INT)
   FRAGMENT BY EXPRESSION
      (a <= 10) IN tabdbspc1,
      (a <= 20) IN tabdbspc2,
      (a <= 30) IN tabdbspc3;
CREATE INDEX idx1 ON tb1 (a)
   FRAGMENT BY EXPRESSION
      (a <= 10) IN idxdbspc1,
      (a <= 20) IN idxdbspc2,
      (a <= 30) IN idxdbspc3;

CREATE TABLE tb2 (a INT CHECK a> 30 AND a<= 40)
   IN tabdbspc4;
CREATE INDEX idx2 ON tb2(a)
   IN idxdbspc4;
```

Suppose you then attach table **tb2** to table **tb1** with the following sample SQL statement:

```
ALTER FRAGMENT ON TABLE tb1
   ATTACH tb2 AS (a <= 40);
```

The database server can eliminate the rebuild of index **idx1** for this attach operation for the following reasons:

- The fragmentation expression for index **idx1** is identical to the fragmentation expression for table **tb1**. The database server:
  - Expands the fragmentation of the index **idx1** to the dbspace **idxdbspc4**
  - Converts index **idx2** to a fragment of index **idx1**
- No rows move from one fragment to another because the CHECK constraint is identical to the resulting fragmentation expression of the attached table.
  For more information about how to ensure no data movement between the existing and the new table fragments, see Ensuring no data movement when you attach a fragment.

Attaching unfragmented tables together

You can take advantage of the performance benefits of the ALTER FRAGMENT ATTACH operation when you combine two unfragmented tables into one fragmented table.

For example, suppose you create two unfragmented tables and indexes with the following SQL statements:

```
CREATE TABLE tb1(a int) IN db1;
   CREATE INDEX idx1 ON tb1(a) in db1;
CREATE TABLE tb2(a int) IN db2;
   CREATE INDEX idx2 ON tb2(a) in db2;
```

You might want to combine these two unfragmented tables with the following sample distribution scheme:

```
ALTER FRAGMENT ON TABLE tb1
   ATTACH
      tb1 AS (a <= 100),
      tb2 AS (a > 100);
```

If no data migrates between the fragments of **tb1** and **tb2**, the database server redefines index **idx1** with the following fragmentation strategy:

```
CREATE INDEX idx1 ON tb1(a) F
   FRAGMENT BY EXPRESSION
      a <= 100 IN db1,
      a > 100 IN db2;
```

Important: This behavior results in a different fragmentation strategy for the index prior to version 7.3 and version 9.2 of the database server. In earlier versions, the ALTER FRAGMENT ATTACH statement creates an unfragmented detached index in the dbspace **db1**.

Ensuring no data movement when you attach a fragment

You can ensure there is no data movement when you attach a fragment by establishing identical check constraint expressions and verifying that fragment expressions are not overlapping.

To ensure that no data movement occurs when you attach a fragment:

1. Establish a check constraint on the attached table that is identical to the fragment expression that it will assume after the ALTER FRAGMENT ATTACH operation.
2. Define the fragments with nonoverlapping expressions.

For example, you might create a fragmented table and index with the following SQL statements:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;

CREATE INDEX idx1 ON tb1(a);
```

Suppose you create another table that is not fragmented, and you subsequently decide to attach it to the fragmented table.

```
CREATE TABLE tb2 (a int, check (a >=10 and a<15))
   IN db3;

CREATE INDEX idx2 ON tb2(a)
   IN db3;

ALTER FRAGMENT ON TABLE tb1
   ATTACH
      tb2 AS (a >= 10 AND a<15) AFTER db2;
```

This ALTER FRAGMENT ATTACH operation takes advantage of the existing index **idx2** because the following steps were performed in the example to prevent data movement between the existing and the new table fragment:

- The check constraint expression in the CREATE TABLE **tb2** statement is identical to the fragment expression for table **tb2** in the ALTER FRAGMENT ATTACH statement.
- The fragment expressions specified in the CREATE TABLE **tb1** and the ALTER FRAGMENT ATTACH statements are not overlapping.

Therefore, the database server preserves index **idx2** in dbspace **db3** and converts it into a fragment of index **idx1**. The index **idx1** remains as an index with the same fragmentation strategy as the table **tb1**.

Indexes on attached tables

The database server tries to reuse the indexes on the attached tables as fragments of the resultant index. However, the corresponding index on the attached table might not exist or might not be usable due to disk-format mismatches. In these cases, it might be faster to build an index on the attached tables rather than to build the entire index on the resultant table.

Informix® estimates the cost to create the whole index on the resultant table. The server then compares this cost to the cost of building the individual index fragments for the attached tables and chooses the index build with the least cost.

Automatically Gathered Statistics for New Indexes

When the CREATE INDEX statement runs successfully, with or without the ONLINE keyword, Informix automatically gathers the following statistics for the newly created index:

- Index-level statistics, equivalent to the statistics gathered in the UPDATE STATISTICS operation in LOW mode, for all types of indexes, including B-tree, Virtual Index Interface, and functional indexes.
- Column-distribution statistics, equivalent to the distribution generated in the UPDATE STATISTICS operation in HIGH mode, for a non-opaque leading indexed column of an ordinary B-tree index. The resolution of the HIGH mode is 1.0 for a table size that is less than 1 million rows and 0.5 for higher table sizes. Tables with more than 1 million rows have a better resolution because they have more bins for statistics.

The automatically gathered distribution statistics are available to the query optimizer when it designs query plans for the table on which the new index was created.

Run UPDATE STATISTICS Before Attaching Tables

To ensure that cost estimates are correct, you should execute the UPDATE STATISTICS statement on all of the participating tables before you attach the tables. The LOW mode of the UPDATE STATISTICS statement is sufficient to derive the appropriate statistics for the optimizer to determine cost estimates for rebuilding indexes.

For more information about using the UPDATE STATISTICS statement, see the *IBM® Informix Guide to SQL: Syntax*.

- Example for situation when corresponding index does not exist
  When a table does not have an index on a column that can serve as the fragment of the resultant index, the database server estimates the cost of building the index fragment for the column, compares this cost to rebuilding the entire index for all fragments on the resultant table, and chooses the index build with the least cost.
- Example for situation when index on table is not usable
  When the index on a table is not usable, the database server estimates the cost of building the index fragment, compares this cost to rebuilding the entire index for all fragments on the resultant table, and chooses the index build with the least cost.

Example for situation when corresponding index does not exist

When a table does not have an index on a column that can serve as the fragment of the resultant index, the database server estimates the cost of building the index fragment for the column, compares this cost to rebuilding the entire index for all fragments on the resultant table, and chooses the index build with the least cost.

Suppose you create a fragmented table and index with the following SQL statements:

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;
CREATE INDEX idx1 ON tb1(a);
```

Suppose you then create two more tables that are not fragmented, and you subsequently decide to attach them to the fragmented table.

```
CREATE TABLE tb2 (a int, b int,
   CHECK (a >=10 and a<15)) IN db3;
CREATE INDEX idx2 ON tb2(a) IN db3;
CREATE TABLE tb3 (a int, b int,
   CHECK (a >= 15 and a<20)) IN db4;
CREATE INDEX idx3 ON tb3(b) IN db4;

ALTER FRAGMENT ON TABLE tb1
   ATTACH tb2 AS (a >= 10 and a<15) tb3 AS (a >= 15 and a<20);
```

The three CREATE INDEX statements automatically calculate distribution statistics for the leading column of each index in HIGH mode, as well as index statistics and table statistics in LOW mode.

The only time the `UPDATE STATISTICS LOW FOR TABLE` statement is required is after a CREATE INDEX statement in a situation in which the table has other preexisting indexes, as shown in this example:

```
CREATE TABLE tb1(col1 int, col2 int);
CREATE INDEX index idx1 on tb1(col1);
  (equivalent to update stats low on table tb1)
LOAD from tb1.unl insert into tb1; (load some data)
CREATE INDEX idx2 on tb1(col2);
```

The statement `CREATE INDEX idx2 on tb1(col2)` is NOT completely equivalent to `UPDATE STATISTICS LOW FOR TABLE tb1`, because the CREATE INDEX statement does not update index- level statistics for the preexisting index called `idx1`.

In the preceding example, table **tb3** does not have an index on column **a** that can serve as the fragment of the resultant index **idx1**. The database server estimates the cost of building the index fragment for column **a** on the consumed table **tb3** and compares this cost to rebuilding the entire index for all fragments on the resultant table. The database server chooses the index build with the least cost.

Example for situation when index on table is not usable

When the index on a table is not usable, the database server estimates the cost of building the index fragment, compares this cost to rebuilding the entire index for all fragments on the resultant table, and chooses the index build with the least cost.

Suppose you create tables and indexes as in the previous section, but the index on the third table specifies a dbspace that the first table also uses. The following SQL statements show this scenario:

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;
CREATE INDEX idx1 ON tb1(a);
CREATE TABLE tb2 (a int, b int, check (a >=10 and a<15))
   IN db3;
CREATE INDEX idx2 ON tb2(a)
   IN db3;

CREATE TABLE tb3 (a int, b int, check (a >= 15 and a<20))
   IN db4;
CREATE INDEX idx3 ON tb3(a)
   IN db2 ;
```

This example creates the index **idx3** on table **tb3** in the dbspace **db2**. As a result, index **idx3** is not usable because index **idx1** already has a fragment in the dbspace **db2,** and the fragmentation strategy does not allow more than one fragment to be specified in a given dbspace.

Again, the database server estimates the cost of building the index fragment for column **a** on the consumed table **tb3** and compares this cost to rebuilding the entire index **idx1** for all fragments on the resultant table. Then the database server chooses the index build with the least cost.

Improving ALTER FRAGMENT DETACH performance

You can improve the performance of ALTER FRAGMENT DETACH statements by formulating appropriate distribution schemes for your table and index fragments and by eliminating the index build during the execution of ALTER FRAGMENT DETACH statements.

To eliminate the index build during execution of the ALTER FRAGMENT DETACH statement, use one of the following fragmentation strategies:

- Fragment the index in the same way as the table.
- Fragment the index with the same distribution scheme as the table.

Important: Only logging databases can benefit from the performance improvements for the ALTER FRAGMENT DETACH statement. Without logging, the database server works with multiple copies of the same table to ensure recoverability of the data when a failure occurs. This requirement prevents reuse of the existing index fragments.

- Fragmenting the index in the same way as the table
  You fragment an index in the same way that you fragment the table when you create a fragmented table and subsequently create an index without specifying a fragmentation strategy, unless the distribution scheme is round-robin and automatic location is enabled. Indexes on tables that use the round-robin distribution scheme are not fragmented when the AUTOLOCATE configuration parameter or environment option is set to a positive integer.
- Fragmenting the index using same distribution scheme as the table
  You fragment an index with the same distribution scheme as the table when you create the index that uses the same fragment expressions as the table.

Fragmenting the index in the same way as the table

You fragment an index in the same way that you fragment the table when you create a fragmented table and subsequently create an index without specifying a fragmentation strategy, unless the distribution scheme is round-robin and automatic location is enabled. Indexes on tables that use the round-robin distribution scheme are not fragmented when the AUTOLOCATE configuration parameter or environment option is set to a positive integer.

For example, suppose you create a fragmented table and index with the following SQL statements:

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2,
      (a >=10 AND a <15) IN db3;
CREATE INDEX idx1 ON tb1(a);
```

The database server fragments the index keys into dbspaces **db1**, **db2**, and **db3** with the same column **a** value ranges as the table because the CREATE INDEX statement does not specify a fragmentation strategy.

Suppose you then decide to detach the data in the third fragment with the following SQL statement:

```
ALTER FRAGMENT ON TABLE tb1
   DETACH db3 tb3;
```

Because the fragmentation strategy of the index is the same as the table, the ALTER FRAGMENT DETACH statement does not rebuild the index after the detach operation. The database server drops the fragment of the index in dbspace **db3**, updates the system catalog tables, and eliminates the index build.

Fragmenting the index using same distribution scheme as the table

You fragment an index with the same distribution scheme as the table when you create the index that uses the same fragment expressions as the table.

A common fragmentation strategy is to fragment indexes in the same way as the tables but to specify different dbspaces for the index fragments. This fragmentation strategy of putting the index fragments into different dbspaces from the table can improve the performance of operations such as backup and recovery.

For example, suppose you create a fragmented table and index with the following SQL statements:

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2,
      (a >=10 AND a <15) IN db3;

CREATE INDEX idx1 on tb1(a)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a< 5) IN db4,
      (a >=5 AND a< 10) IN db5,
      (a >=10 AND a<15) IN db6;
```

Suppose that you then decide to detach the data in the third fragment with the following SQL statement:

```
ALTER FRAGMENT ON TABLE tb1
   DETACH db3 tb3;
```

Because the distribution scheme of the index is the same as the table, the ALTER FRAGMENT DETACH statement does not rebuild the index after the detach operation. The database server drops the fragment of the index in dbspace **db3**, updates the system catalog tables, and eliminates the index build.

Forcing out transactions when altering table fragments

You can enable the server to force out transactions that have opened or hold locks on the target table of an ALTER FRAGMENT ON TABLE operation in a logging database. Users holding the DBA access privilege can do this by enabling the FORCE_DDL_EXEC session environment option of the SET ENVIRONMENT statement.

You might want to do this on a busy system, perhaps one that runs 24 hours a day, if you do not want to wait for sessions to close before you alter a fragment.

Be aware, however, that by forcing out concurrent transactions to avoid waiting for locks to be released, the database server closes the Update cursors and rolls back the transactions of other users.

Prerequisites:

- You must be user **informix** or hold DBA access privileges on the database.
- The table must be in a database that supports transaction logging.

To force out concurrent transactions of other sessions when altering a table fragment:

1. Set the FORCE_DDL_EXEC environment option of the SET ENVIRONMENT statement to one of the following values:
   - `ON`, `on`, `'1'`, or `"1"` to enable the server to force out transactions that are open or have a lock on the table when an ALTER FRAGMENT ON TABLE statement is issued, until the server obtains a lock and exclusive access to the table.
   - A positive integer that represents an amount of time in seconds. The numeric value enables the server to force out transactions until the server gets exclusive access and exclusive locks on the table, or until the specified time limit. If the server cannot force out transactions by the specified number of seconds, the server stops attempting to force out the transactions, and the ALTER FRAGMENT statement waits for the locks to be released when the concurrent transactions are committed or rolled back.

   For example, to enable the FORCE_DDL_EXEC environment option to operate for 100 seconds when an ALTER FRAGMENT ON TABLE statement is issued, specify:

   ```
   SET ENVIRONMENT FORCE_DDL_EXEC '100';
   ```

2. Set the lock mode to wait to ensure that the server will wait a specified amount of time before forcing out any transactions.
   For example, to set the lock mode to wait for 20 seconds, specify:

   ```
   SET LOCK MODE TO WAIT "20";
   ```

   For more information, see Setting the lock mode to wait.

3. Run an ALTER FRAGMENT ON TABLE statement, for example, to attach, detach, modify, add, or drop the fragment.

The following SQL statements perform these actions:

- enable the FORCE_DDL_EXEC session environment option for 100 seconds,
- set the database server to wait up to 25 seconds for locks to be released,
- and change the interval size and storage location of range fragment **p2** of table **tabF**:

```
SET ENVIRONMENT FORCE_DDL_EXEC '100';
SET LOCK MODE TO WAIT 25;
ALTER FRAGMENT ON TABLE tabF MODIFY
   PARTITION p2 TO PARTITION p2 VALUES < 500 IN dbs0;
```

Attention:
While the ALTER FRAGMENT statement above is running, other transactions that attempt to access rows in table **tabF** are at risk of being forced out, if their Update cursor holds locks on rows in fragment **p2**.

After a transaction is rolled back because the FORCE_DDL_EXEC session environment option is enabled by another session, the database server returns this error to the session whose transaction failed:

```
-458  Long transaction aborted.
```

The concurrent transaction failing with error `-458` was not necessarily "long," but it had not yet been committed after opening or holding locks on the same table that the ALTER FRAGMENT statement in this example was modifying.
After you complete an ALTER FRAGMENT ON TABLE operation with the FORCE_DDL_EXEC session environment option enabled, you can turn the FORCE_DDL_EXEC session environment option off. For example, specify:

```
SET ENVIRONMENT FORCE_DDL_EXEC OFF;
```

Related information:
FORCE_DDL_EXEC session environment option

Monitoring fragment use

Once you determine a fragmentation strategy, you can monitor fragmentation.

You can monitor fragmentation in the following ways:

- Run individual **onstat** utility commands to capture information about specific aspects of a running query.
- Run a SET EXPLAIN statement before you run a query to write the query plan to an output file.

- Monitoring fragmentation with the onstat -g ppf command
  With the **onstat -g ppf** command, you can view partition information and monitor the I/O activity to verify your strategy and determine whether the I/O is balanced across fragments.

- Monitoring fragmentation with SET EXPLAIN output
  When the table is fragmented, the output of the SET EXPLAIN ON statement shows which table or index the database server scans to execute the query.

Related concepts:
Distribution schemes
Strategy for fragmenting indexes
Strategy for fragmenting temporary tables
Distribution schemes that eliminate fragments
Improve the performance of operations that attach and detach fragments
Related tasks:
Planning a fragmentation strategy

Monitoring fragmentation with the onstat -g ppf command

With the **onstat -g ppf** command, you can view partition information and monitor the I/O activity to verify your strategy and determine whether the I/O is balanced across fragments.

The **onstat -g ppf** output includes the number of read-and-write requests sent to each fragment that is currently open. Because a request can trigger multiple I/O operations, these requests do not indicate how many individual disk I/O operations occur, but you can get a good idea of the I/O activity from the displayed columns.

The `brfd` column in the output displays the number of buffer reads in pages. (Each buffer can contain one page.) This information is useful if you need to monitor the time a query takes to execute. Typically query execution time has a strong dependency on the number of required buffer reads. If the size of client-server buffering is small and your database contains TEXT data, query execution time can involve significantly more buffer reads, because the server reads the prior TEXT data.

The **onstat -g ppf** output by itself does not identify the table in which a fragment is located. To determine the table for the fragment, join the `partnum` column in the output to the **partnum** column in the **sysfragments** system catalog table. The **sysfragments** table displays the associated **table id**. You can also find the table name for the fragment by joining the **table id** column in **sysfragments** to the **table id** column in **systables**.

To determine the table name in **onstat -g ppf** output:

1. Obtain the value in the **partnum** field of the **onstat -g ppf** output.
2. Join the **tabid** column in the **sysfragments** system catalog table with the **tabid** column in the **systables** system catalog table to obtain the table name from **systables**.
   Use the **partnum** field value that you obtain in step 1 in the SELECT statement.

```
SELECT a.tabname FROM systables a, sysfragments b
   WHERE a.tabid = b.tabid
      AND partn = partnum_value;
```


Monitoring fragmentation with SET EXPLAIN output

When the table is fragmented, the output of the SET EXPLAIN ON statement shows which table or index the database server scans to execute the query.

The SET EXPLAIN output identifies the fragments with a fragment number. The fragment numbers are the same as those contained in the **partn** column in the **sysfragments** system catalog table.

The following example of partial SET EXPLAIN output shows a query that takes advantage of fragment elimination and scans two fragments in table **t1**:

```
QUERY:
------
SELECT * FROM t1 WHERE c1 > 12

Estimated Cost: 3
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN (Serial, fragments: 1, 2)

   Filters: informix.t1.c1 > 12
```

If the optimizer must scan all fragments (that is, if it is unable to eliminate any fragment from consideration), the SET EXPLAIN output displays fragments: `ALL`. In addition, if the optimizer eliminates all the fragments from consideration (that is, none of the fragments contain the queried information), the SET EXPLAIN output displays fragments: `NONE`.

For information about how the database server eliminates a fragment from consideration, see Distribution schemes that eliminate fragments.

For more information about the SET EXPLAIN ON statement, see Report that shows the query plan chosen by the optimizer.

Queries and the query optimizer

These topics describe query plans, explain how the database server manages query optimization, and discuss factors that you can use to influence the query plan. These topics also describe performance considerations for SPL routines, the UDR cache, and triggers.

The parallel database query (PDQ) features in the database server provide the largest potential performance improvements for a query. Parallel database query (PDQ) describes PDQ and the Memory Grant Manager (MGM) and explains how to control resource use by queries.

PDQ provides the most substantial performance gains if you fragment your tables as described in Fragmentation guidelines.

Improving individual query performance explains how to improve the performance of specific queries.

Data warehouse queries and performance issues related to dimensional databases are described in the *IBM Informix Data Warehouse Guide*.

1. The query plan
   The query optimizer evaluates the different ways in which a query might be performed and determines the best way to select the requested data. During this evaluation, the optimizer formulates a *query plan* to fetch the data rows that are required to process a query.
2. Factors that affect the query plan
   When the optimizer determines the query plan, it assigns a cost to each possible plan and then chooses the plan with the lowest cost. The optimizer analyzes several factors to determine the cost of each query plan.
3. Time costs of a query
   You can adjust a few, but not all, of the response-time effects of actions that the database server performs when processing a query.
4. Optimization when SQL is within an SPL routine
   If an SPL routine contains SQL statements, the database server optimizes and executes the SQL within the SPL routine.
5. Trigger execution
   A *trigger* is a database object that automatically executes one or more SQL statements (the *triggered action*) when a specified data manipulation language operation (the *triggering event*) occurs. You can define one or more triggers on a table to execute after a SELECT, INSERT, UPDATE or DELETE triggering event.

Related information:
Performance tuning dimensional databases

The query plan

The query optimizer evaluates the different ways in which a query might be performed and determines the best way to select the requested data. During this evaluation, the optimizer formulates a *query plan* to fetch the data rows that are required to process a query.

For example, when evaluating the different ways in which a query might be performed, the optimizer must determine whether indexes should be used. If the query includes a join, the optimizer must determine the join plan (hash or nested loop) and the order in which tables are evaluated or joined.

The following topics describe the components of a query plan and show examples of query plans.

- The access plan
  The way that the optimizer chooses to read a table is called an *access plan*. The simplest method to access a table is to read it sequentially, which is called a *table scan*. The optimizer chooses a table scan when most of the table must be read or the table does not have an index that is useful for the query.
- The join plan
  When a query contains more than one table, Informix® joins the tables using filters in the query. The way that the optimizer chooses to join the tables is the *join plan*.
- Example of query-plan execution
  This topic contains an example of a query with a SELECT statement that calls for a three-way join and describes one possible query plan.
- Query plans that include an index self-join path
  An *index self-join* is a type of index scan that you can think of as a union of many small index scans, each one with a single unique combination of lead-key columns and filters on non-lead-key columns.
- Query plan evaluation
  The optimizer considers all query plans by analyzing factors such as disk I/O and CPU costs.
- Report that shows the query plan chosen by the optimizer
  Any user who runs a query can use the SET EXPLAIN statement or the EXPLAIN directive to display the query plan that the optimizer chooses.
- Sample query plan reports
  The topics in this section describe sample query plans that you might want to display when analyzing the performance of different kinds of queries.
- XML query plans in IBM Data Studio
  IBM® Data Studio consists of a set of tools to use for administration, data modeling, and building queries from data that comes from data servers. The EXPLAIN_SQL routine prepares a query and returns a query plan in XML. The IBM Data Studio Administration Edition can use the EXPLAIN_SQL routine to obtain a query plan in XML format, interpret the XML, and render the plan visually.

Next topic: Factors that affect the query plan

The access plan

The way that the optimizer chooses to read a table is called an *access plan*. The simplest method to access a table is to read it sequentially, which is called a *table scan*. The optimizer chooses a table scan when most of the table must be read or the table does not have an index that is useful for the query.

The optimizer can also choose to access the table by an index. If the column in the index is the same as a column in a filter of the query, the optimizer can use the index to retrieve only the rows that the query requires. The optimizer can use a *key-only index scan* if the columns requested are within one index on the table. The database server retrieves the needed data from the index and does not access the associated table.
Important: The optimizer does not choose a key-only scan for a VARCHAR column. If you want to take advantage of key-only scans, use the ALTER TABLE with the MODIFY clause to change the column to a CHAR data type.
The optimizer compares the cost of each plan to determine the best one. The database server derives cost from estimates of the number of I/O operations required, calculations to produce the results, rows accessed, sorting, and so forth.

The join plan

When a query contains more than one table, Informix® joins the tables using filters in the query. The way that the optimizer chooses to join the tables is the *join plan*.

In the following query, the customer and orders table are joined by the `customer.customer_num = orders.customer_num` filter:

```
SELECT * from customer, orders
WHERE customer.customer_num = orders.customer_num
AND customer.lname = "Higgins";
```

The join method can be a nested-loop join or a hash join.

Because of the nature of hash joins, an application with isolation level set to Repeatable Read might temporarily lock all the records in tables that are involved in the join, including records that fail to qualify the join. This situation leads to decreased concurrency among connections. Conversely, nested-loop joins lock fewer records but provide reduced performance when a large number of rows are accessed. Thus, each join method has advantages and disadvantages.

- Nested-loop join
  In a nested-loop join, the database server scans the first, or *outer table*, and then joins each of the rows that pass table filters to the rows found in the second, or *inner table.*
- Hash join
  The optimizer usually uses a hash join when at least one of the two join tables does not have an index on the join column or when the database server must read a large number of rows from both tables. No index and no sorting is required when the database server performs a hash join.
- Join order
  The order that tables are joined in a query is extremely important. A poor join order can cause query performance to decline noticeably.

Nested-loop join

In a nested-loop join, the database server scans the first, or *outer table*, and then joins each of the rows that pass table filters to the rows found in the second, or *inner table.*

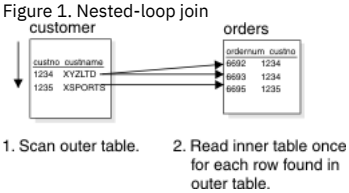
Figure 1 shows tables and rows, and the order they are read, for query:

```
SELECT * FROM customer, orders
WHERE customer.customer_num=orders.customer_num
AND order_date>"01/01/2007";
```

The database server accesses an outer table by an index or by a table scan. The database server applies any table filters first. For each row that satisfies the filters on the outer table, the database server reads the inner table to find a match.

The database server reads the inner table once for every row in the outer table that fulfills the table filters. Because of the potentially large number of times that the inner table can be read, the database server usually accesses the inner table by an index.

Figure 1. Nested-loop join



If the inner table does not have an index, the database server might construct an *autoindex* at the time of query execution. The optimizer might determine that the cost to construct an *autoindex* at the time of query execution is less than the cost to scan the inner table for each qualifying row in the outer table.

If the optimizer changes a subquery to a nested-loop join, it might use a variation of the nested-loop join, called a *semi join*. In a semi join, the database server reads the inner table only until it finds a match. In other words, for each row in the outer table, the inner table contributes at most one row. For more information on how the optimizer handles subqueries, see Query plans for subqueries.

Hash join

The optimizer usually uses a hash join when at least one of the two join tables does not have an index on the join column or when the database server must read a large number of rows from both tables. No index and no sorting is required when the database server performs a hash join.

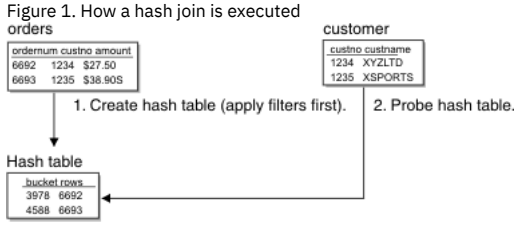
A hash join consists of two activities: first building the hash table (*build* phase) and then probing the hash table (*probe* phase). Figure 1 shows the hash join in detail.

In the build phase, the database server reads one table and, after it applies any filters, creates a hash table. Think of a hash table conceptually as a series of *buckets*, each with an address that is derived from the key value by applying a hash function. The database server does not sort keys in a particular hash bucket.

Smaller hash tables can fit in the virtual portion of database server shared memory. The database server stores larger hash files on disk in the dbspace specified by the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable.

In the probe phase, the database server reads the other table in the join and applies any filters. For each row that satisfies the filters on the table, the database server applies the hash function on the key and probes the hash table to find a match.

Figure 1. How a hash join is executed



Join order

The order that tables are joined in a query is extremely important. A poor join order can cause query performance to decline noticeably.

The following SELECT statement calls for a three-way join:

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
```

The optimizer can choose one of the following join orders:

- Join **customer** to **orders**. Join the result to **items**.
- Join **orders** to **customer**. Join the result to **items**.
- Join **customer** to **items**. Join the result to **orders**.
- Join **items** to **customer**. Join the result to **orders**.
- Join **orders** to **items**. Join the result to **customer**.
- Join **items** to **orders**. Join the result to **customer**.

For an example of how the database server executes a plan according to a specific join order, see Example of query-plan execution.

Example of query-plan execution

This topic contains an example of a query with a SELECT statement that calls for a three-way join and describes one possible query plan.

The following SELECT statement calls for a three-way join:

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
```

Assume also that no indexes are on any of the three tables. Suppose that the optimizer chooses the **customer-orders-items** path and the nested-loop join for both joins (in reality, the optimizer usually chooses a hash join for two tables without indexes on the join columns). Figure 1 shows the *query plan*, expressed in pseudocode. For information about interpreting query plan information, see Report that shows the query plan chosen by the optimizer.

Figure 1. A query plan in pseudocode

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept the row and send to user
            end if
         end for
      end if
   end for
end for
```

This procedure reads the following rows:

- All rows of the **customer** table once
- All rows of the **orders** table once for each row of the **customer** table
- All rows of the **items** table once for each row of the **customer-orders** pair

This example does not describe the only possible query plan. Another plan merely reverses the roles of **customer** and **orders**: for each row of **orders**, it reads all rows of **customer**, looking for a matching **customer_num**. It reads the same number of rows in a different order and produces the same set of rows in a different order. In this example, no difference exists in the amount of work that the two possible query plans need to do.

- Example of a join with column filters
  The presence of a *column filter* can change the query plan. A column filter is a WHERE expression that reduces the number of rows that a table contributes to a join.

- Example of a join with indexes
  The presence of indexes and constraints in query plans provides the optimizer with options that can greatly improve query-execution time.

Example of a join with column filters

The presence of a *column filter* can change the query plan. A column filter is a WHERE expression that reduces the number of rows that a table contributes to a join.

The following example shows the query described in Example of query-plan execution with a filter added:

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
      AND O.paid_date IS NULL
```

The expression `O.paid_date` IS NULL filters out some rows, reducing the number of rows that are used from the **orders** table. Consider a plan that starts by reading from **orders**. Figure 1 displays this sample plan in pseudocode.
Figure 1. Query plan that uses a column filter

```
for each row in the orders table do:
   read the row into O
   if O.paid_date is null then
      for each row in the customer table do:
         read the row into C
         if O.customer_num = C.customer_num then
            for each row in the items table do:
               read the row into I
               if I.order_num = O.order_num then
                  accept row and return to user
               end if
            end for
         end if
      end for
   end if
end for
```

Let *pdnull* represent the number of rows in **orders** that pass the filter. It is the value of **COUNT(*)** that results from the following query:

```
SELECT COUNT(*) FROM orders WHERE paid_date IS NULL
```

If one customer exists for every order, the plan in Figure 1 reads the following rows:

- All rows of the **orders** table once
- All rows of the **customer** table, *pdnull* times
- All rows of the **items** table, *pdnull* times

Figure 2 shows an alternative execution plan that reads from the **customer** table first.
Figure 2. The alternative query plan in pseudocode

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.paid_date is null and
           O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept row and return to user
            end if
         end for
      end if
   end for
```

Because the filter is not applied in the first step that Figure 2 shows, this plan reads the following rows:

- All rows of the **customer** table once
- All rows of the **orders** table once for every row of **customer**
- All rows of the **items** table, *pdnull* times

The query plans in Figure 1 and Figure 2 produce the same output in a different sequence. They differ in that one reads a table *pdnull* times, and the other reads a table SELECT COUNT`(*)` FROM `customer` times. By choosing the appropriate plan, the optimizer can save thousands of disk accesses in a real application.

Example of a join with indexes

The presence of indexes and constraints in query plans provides the optimizer with options that can greatly improve query-execution time.

This topic shows the outline of a query plan that differs from query shown in Example of a join with column filters, because it is constructed using indexes.
Figure 1. Query plan with indexes

```
for each row in the customer table do:
   read the row into C
   look up C.customer_num in index on orders.customer_num
   for each matching row in the orders index do:
      read the table row for O
      if O.paid_date is null then
         look up O.order_num in index on items.order_num
         for each matching row in the items index do:
            read the row for I
            construct output row and return to user
         end for
      end if
   end for
end for
```

The keys in an index are sorted so that when the database server finds the first matching entry, it can read any other rows with identical keys without further searching, because they are located in physically adjacent positions. This query plan reads only the following rows:

- All rows of the **customer** table once
- All rows of the **orders** table once (because each order is associated with only one customer)
- Only rows in the **items** table that match *pdnull* rows from the **customer-orders** pairs

This query plan achieves a great reduction in cost compared with plans that do not use indexes. An inverse plan, reading **orders** first and looking up rows in the **customer** table by its index, is also feasible by the same reasoning.

The physical order of rows in a table also affects the cost of index use. To the degree that a table is ordered relative to an index, the overhead of accessing multiple table rows in index order is reduced. For example, if the **orders** table rows are physically ordered according to the customer number, multiple retrievals of orders for a given customer would proceed more rapidly than if the table were ordered randomly.

In some cases, using an index might incur additional costs. For more information, see Index lookup costs.

Query plans that include an index self-join path

An *index self-join* is a type of index scan that you can think of as a union of many small index scans, each one with a single unique combination of lead-key columns and filters on non-lead-key columns.

The union of small index scans results in an access path that uses only subsets of the full range of a composite index. The table is logically joined to itself, and the more selective non-leading index keys are applied as index-bound filters to each unique combination of the leading key values.

An index self-join is beneficial for situations in which:

- The lead key of an index has many duplicates, and
- Predicates on the lead key are not selective, but predicates on the non-leading index keys are selective.

The query in Figure 1 shows the SET EXPLAIN output for a query plan that includes an index self-join path.

Figure 1. SET EXPLAIN output for a query with an index self-join path

```
QUERY:
------
SELECT a.c1,a.c2,a.c3 FROM tab1 a WHERE (a.c3 >= 100103) AND
     (a.c3 <= 100104) AND (a.c1 >= 'PICKED      ') AND
     (a.c1 <= 'RGA2        ') AND (a.c2 >= 1) AND (a.c2 <= 7)
     ORDER BY 1, 2, 3

Estimated Cost: 155
Estimated # of Rows Returned: 1
  1) informix.a: INDEX PATH
    (1) Index Keys: c1 c2 c3 c4 c5   (Key-Only)   (Serial, fragments: ALL)
        Index Self Join Keys (c1 c2 )
          Lower bound: informix.a.c1 >= 'PICKED      ' AND (informix.a.c2 >= 1 )
          Upper bound: informix.a.c1 <= 'RGA2        ' AND (informix.a.c2 <= 7 )
        Lower Index Filter: (informix.a.c1 = informix.a.c1 AND
          informix.a.c2 = informix.a.c2 ) AND informix.a.c3 >= 100103
        Upper Index Filter: informix.a.c3 <= 100104
        Index Key Filters:  (informix.a.c2 <= 7 ) AND
                            (informix.a.c2 >= 1 )
```

In Figure 1, an index exists on columns c1, c2, c3, c4, and c5. The optimizer chooses c1 and c2 as lead keys, which implies that columns c1 and c2 have many duplicates. Column c3 has few duplicates and thus the predicates on column c3 (`c3 >= 100103` and `c3 <= 100104`) have good selectivity.

As Figure 1 shows, an index self-join path is a self-join of two index scans using the same index. The first index scan retrieves each unique value for lead key columns, which are c1 and c2. The unique value of c1 and c2 is then used to probe the second index scan, which also uses predicates on column c3. Because predicates on column c3 have good selectivity:

- The index scan on the inner side of the nested-loop join is very efficient, retrieving only the few rows that satisfy the c3 predicates.
- The index scan does not retrieve extra rows.

Thus, for each unique value of c1 and c2, an efficient index scan on c1, c2 and c3 occurs.

The following lines in the example indicate that the optimizer has chosen an index self join path for this table, with columns c1 and c2 as the lead keys for the index self-join path:

```
Index Self Join Keys (c1 c2 )
          Lower bound: informix.a.c1 >= 'PICKED      ' AND (informix.a.c2 >= 1 )
```

```
        Upper bound: informix.a.c1 <= 'RGA2        ' AND (informix.a.c2 <= 7 )
```

The example shows the bounds for columns c1 and c2, which you can conceive of as the bounds for the index scan to retrieve the qualified leading keys of the index.

The following information in the example shows the self-join:

```
(informix.a.c1 = informix.a.c1 AND informix.a.c2 = informix.a.c2 )
```

This information represents the inner index scan. For lead key columns c1 and c2 the self- join predicate is used, indicating the value of c1 and c2 comes from the outer index scan. The predicates on column c3 serve as an index filter that makes the inner index scan efficient.

Regular index scans do not use filters on column c3 to position the index scan, because the lead key columns c1 and c2 do not have equality predicates.

The INDEX_SJ directive forces an index self-join path using the specified index, or choosing the least costly index in a list of indexes, even if data distribution statistics are not available for the leading index key columns. The AVOID_INDEX_SJ directive prevents a self-join path for the specified index or indexes. Also see Access-method directives and the *IBM® Informix® Guide to SQL: Syntax*.

Query plan evaluation

The optimizer considers all query plans by analyzing factors such as disk I/O and CPU costs.

The optimizer constructs all feasible plans simultaneously using a bottom-up, breadth-first search strategy. That is, the optimizer first constructs all possible join pairs. It eliminates the more expensive pair of any *redundant* pair. (Redundant pairs are join pairs that contain the same tables and produce the same set of rows as another join pair.)

For example, if neither join specifies an ordered set of rows by using the ORDER BY or GROUP BY clauses of the SELECT statement, the join pair (A x B) is redundant with respect to (B x A).

If the query uses additional tables, the optimizer joins each remaining pair to a new table to form all possible join triplets, eliminating the more expensive of redundant triplets and so on for each additional table to be joined. When a non-redundant set of possible join combinations has been generated, the optimizer selects the plan that appears to have the lowest execution cost.

Report that shows the query plan chosen by the optimizer

Any user who runs a query can use the SET EXPLAIN statement or the EXPLAIN directive to display the query plan that the optimizer chooses.

For information about how to specify the directives, see EXPLAIN directives. The user enters the SET EXPLAIN ON statement or the SET EXPLAIN ON AVOID_EXECUTE statement before the SQL statement for the query, as the following example shows.

```
SET EXPLAIN ON AVOID_EXECUTE;
SELECT * FROM customer, orders
WHERE customer.customer_num = orders.customer_num
   AND customer.lname = "Higgins";
```

If a user does not have any access to SQL code source, the Database Administrator can set dynamically the SET EXPLAIN using the **onmode -Y** command.

After the database server executes the SET EXPLAIN ON statement or sets dynamically the SET EXPLAIN with **onmode -Y** command, the server writes an explanation of each query plan to a file for subsequent queries that the user enters.

- The explain output file
  The SET EXPLAIN statement enables or disables recording measurements of queries in the current session, including the plan of the query optimizer, an estimate of the number of rows returned, and the relative cost of the query. The measurements appear in an output file.
- Query statistics section provides performance debugging information
  If the EXPLAIN_STAT configuration parameter is enabled, a Query Statistics section appears in the explain output file that the SET EXPLAIN statement of SQL and the **onmode -Y *session_id*** command displays.

Related concepts:
The explain output file
Query statistics section provides performance debugging information
Report that shows the query plan chosen by the optimizer
Enabling external directives
Related information:
SET EXPLAIN statement
Using the FILE TO option
Default name and location of the explain output file on UNIX
Default name and location of the output file on Windows
onmode -Y: Dynamically change SET EXPLAIN
onmode and Y arguments: Change query plan measurements for a session (SQL administration API)

The explain output file

The SET EXPLAIN statement enables or disables recording measurements of queries in the current session, including the plan of the query optimizer, an estimate of the number of rows returned, and the relative cost of the query. The measurements appear in an output file.

When you run the **onmode -Y** command to turn on dynamic SET EXPLAIN, the output is displayed in a new explain output file. If a file from the SET EXPLAIN statement exists, the database server stops using it, and instead uses the file created by **onmode -Y** until the administrator turns off dynamic SET EXPLAIN for the session.

The output file specifies if external directives are in effect.

The following codes in the Query Statistics section of the explain output file provide information about external tables:

- `xlcnv` identifies an operation that is loading data from an external table and inserting the data into a base table. Here `x` = external table, `l` = loading, and `cnv` = converter
- `xucnv` identifies an operation that is unloading data from an external table and inserting the data into a base table. Here `x` = external table, `u` = unloading, and `cnv` = converter

The Query Statistics section of the explain output file is a useful resource for debugging performance problems. See Query statistics section provides performance debugging information.

Related concepts:
Report that shows the query plan chosen by the optimizer
Query statistics section provides performance debugging information
Enabling external directives
Related information:
SET EXPLAIN statement
Using the FILE TO option
Default name and location of the explain output file on UNIX
Default name and location of the output file on Windows
onmode -Y: Dynamically change SET EXPLAIN
onmode and Y arguments: Change query plan measurements for a session (SQL administration API)

Query statistics section provides performance debugging information

If the EXPLAIN_STAT configuration parameter is enabled, a Query Statistics section appears in the explain output file that the SET EXPLAIN statement of SQL and the **onmode -Y *session_id*** command displays.

The Query Statistics section of the explain output file shows the estimated number of rows that the query plan expects to return, the actual number of returned rows, and other information about the query. You can use this information, which provides an indication of the overall flow of the query plan and how many rows flow through each stage of the query, to debug performance problems.

The following example shows query statistics in SET EXPLAIN output. If the estimated and actual number of rows scanned or joined are quite different, the statistics on those tables might be old and should be updated.

Figure 1. Query statistics in SET EXPLAIN output

```
select * from tab1, tab2 where tab1.c1 = tab2.c1 and tab1.c3 between 0 and 15

Estimated Cost: 104
Estimated # of Rows Returned: 69

  1) zelaine.tab2: SEQUENTIAL SCAN

  2) zelaine.tab1: INDEX PATH

    (1) Index Keys: c1 c3   (Serial, fragments: ALL)
        Lower Index Filter: (zelaine.tab1.c1 = zelaine.tab2.c1
                             AND zelaine.tab1.c3 >= 0 )
        Upper Index Filter: zelaine.tab1.c3 <= 15
NESTED LOOP JOIN

Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                tab2
  t2                tab1

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     50         50        50         00:00:00   4

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     67         69        4          00:00:00   2

  type     rows_prod  est_rows  time       est_cost
  ------------------------------------------------
  nljoin   67         70        00:00:00   104
```

Related concepts:
The explain output file
Report that shows the query plan chosen by the optimizer

Sample query plan reports
Enabling external directives
Related information:
SET EXPLAIN statement
Using the FILE TO option
Default name and location of the explain output file on UNIX
Default name and location of the output file on Windows
onmode -Y: Dynamically change SET EXPLAIN
onmode and Y arguments: Change query plan measurements for a session (SQL administration API)

Sample query plan reports

The topics in this section describe sample query plans that you might want to display when analyzing the performance of different kinds of queries.

- Single-table query
  This topic shows sample SET EXPLAIN output for a simple query and a complex query on a single table.
- Multitable query
  This topic shows sample SET EXPLAIN output for a multiple-table query.
- Key-first scan
  This topic shows a sample query that uses a *key-first scan*, which is an index scan that uses keys other than those listed as lower and upper index filters.
- Query plans for subqueries
  The optimizer can change a subquery to a join automatically if the join provides a lower cost.
- Query plans for collection-derived tables
  A collection-derived table is a special method that the database server uses to process a query on a collection. To use a collection-derived table, a query must contain the TABLE keyword in the FROM clause of an SQL statement.

Related concepts:
Query statistics section provides performance debugging information

Single-table query

This topic shows sample SET EXPLAIN output for a simple query and a complex query on a single table.

Figure 1 shows SET EXPLAIN output for a simple query.
Figure 1. Partial SET EXPLAIN output for a simple query

```
QUERY:
------
SELECT fname, lname, company FROM customer

Estimated Cost: 2
Estimated # of Rows Returned: 28

  1) virginia.customer: SEQUENTIAL SCAN
```

Figure 2 shows SET EXPLAIN output for a complex query on the **customer** table.

Figure 2. Partial SET EXPLAIN output for a complex query

```
QUERY:
------
SELECT fname, lname, company FROM customer
WHERE company MATCHES 'Sport*' AND
    customer_num BETWEEN 110 AND 115
ORDER BY lname

Estimated Cost: 1
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

  1) virginia.customer: INDEX PATH

        Filters: virginia.customer.company MATCHES 'Sport*'

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: virginia.customer.customer_num >= 110
        Upper Index Filter: virginia.customer.customer_num <= 115
```

The following output lines in Figure 2 show the scope of the index scan for the second query:

- Lower Index Filter: virginia.customer.customer_num >= 110
  Start the index scan with the index key value of `110`.
- Upper Index Filter: virginia.customer.customer_num <= 115
  Stop the index scan with the index key value of `115`.

Multitable query

This topic shows sample SET EXPLAIN output for a multiple-table query.

Figure 1. Partial SET EXPLAIN output for a multi-table query

```
QUERY:
------
SELECT C.customer_num, O.order_num, SUM (I.total_price)
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
GROUP BY C.customer_num, O.order_num

Estimated Cost: 78
Estimated # of Rows Returned: 1
Temporary Files Required For: Group By

  1) virginia.o: SEQUENTIAL SCAN

  2) virginia.c: INDEX PATH

    (1) Index Keys: customer_num   (Key-Only)  (Serial, fragments: ALL)
        Lower Index Filter:
          virginia.c.customer_num = virginia.o.customer_num
NESTED LOOP JOIN

  3) virginia.i: INDEX PATH

    (1) Index Keys: order_num   (Serial, fragments: ALL)
        Lower Index Filter: virginia.o.order_num = virginia.i.order_num
NESTED LOOP JOIN
```

The SET EXPLAIN output lists the order in which the database server accesses the tables and the access plan to read each table. The plan in Figure 1 indicates that the database server is to perform the following actions:

1. The database server is to read the **orders** table first.
   Because no filter exists on the **orders** table, the database server must read all rows. Reading the table in physical order is the least expensive approach.
2. For each row of **orders**, the database server is to search for matching rows in the **customer** table.
   The search uses the index on **customer_num**. The notation `Key-Only` means that only the index need be read for the **customer** table because only the **c.customer_num** column is used in the join and the output, and the column is an index key.
3. For each row of **orders** that has a matching **customer_num**, the database server is to search for a match in the **items** table using the index on **order_num**.

Key-first scan

This topic shows a sample query that uses a *key-first scan*, which is an index scan that uses keys other than those listed as lower and upper index filters.

Figure 1. Partial SET EXPLAIN output for a key-first scan

```
create index idx1 on tab1(c1, c2);

select * from tab1 where (c1 > 0) and ( (c2 = 1) or (c2 = 2))
Estimated Cost: 4
Estimated # of Rows Returned: 1

1) pubs.tab1: INDEX PATH

    (1) Index Keys: c1 c2 (Key-First) (Serial, fragments: ALL)
                Lower Index Filter: pubs.tab1.c1 > 0
                Index Key Filters:  (pubs.tab1.c2 = 1 OR pubs.tab1.c2 = 2)
```

Even though in this example the database server must eventually read the row data to return the query results, it attempts to reduce the number of possible rows by applying additional key filters first. The database server uses the index to apply the additional filter, `c2 = 1 OR c2 = 2`, before it reads the row data.

Query plans for subqueries

The optimizer can change a subquery to a join automatically if the join provides a lower cost.

For example, Figure 1 sample output of the SET EXPLAIN ON statement shows that the optimizer changes the table in the subquery to be the inner table in a join.
Figure 1. Partial SET EXPLAIN output for a flattened subquery

```
QUERY:
------
SELECT company, fname, lname, phone
FROM customer c
```

```
WHERE EXISTS(
   SELECT customer_num FROM cust_calls u
      WHERE c.customer_num = u.customer_num)

Estimated Cost: 6
Estimated # of Rows Returned: 7

  1) virginia.c: SEQUENTIAL SCAN

  2) virginia.u: INDEX PATH  (First Row)

    (1) Index Keys: customer_num call_dtime   (Key-Only)
                                              (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num
NESTED LOOP JOIN  (Semi Join)
```

For more information about the SET EXPLAIN ON statement, see Report that shows the query plan chosen by the optimizer.

When the optimizer changes a subquery to a join, it can use several variations of the access plan and the join plan:

- First-row scan
  A first-row scan is a variation of a table scan. When the database server finds one match, the table scan halts.
- Skip-duplicate-index scan
  The skip-duplicate-index scan is a variation of an index scan. The database server does not scan duplicates.
- Semi join
  The semi join is a variation of a nested-loop join. The database server halts the inner-table scan when the first match is found. For more information about a semi join, see Nested-loop join.

Query plans for collection-derived tables

A collection-derived table is a special method that the database server uses to process a query on a collection. To use a collection-derived table, a query must contain the TABLE keyword in the FROM clause of an SQL statement.

For more information about how to use collection-derived tables in an SQL statement, see the *IBM® Informix Guide to SQL: Syntax*.

Although the database does not actually create a table for the collection, it processes the data as if it were a table. Collection-derived tables allow developers to use fewer cursors and host variables to access a collection, in some cases.

These SQL statements create a collection column called **children**:

```
CREATE ROW TYPE person(name CHAR(255), id INT);
CREATE TABLE parents(name CHAR(255),
id INT,
children LIST(person NOT NULL));
```

The following query creates a collection-derived table for the **children** column and treats the elements of this collection as rows in a table:

```
SELECT name, id
FROM TABLE(MUTLISET(SELECT children
FROM parents
WHERE parents.id
= 1001)) c_table(name, id);
```

Alternatively, you can specify a collection-derived table in the FROM clause, as shown in this example:

```
SELECT name, id
FROM (SELECT children
FROM parents
WHERE parents.id
= 1001) c_table(name, id);
```

- Example showing how the database server completes the query
  Informix® performs several steps when completing a query for collection-derived tables.
- Derived tables folded into parent queries
  You can improve the performance of collection-derived tables by using SQL to fold derived tables in simple queries into a parent query instead of into query results that are put into a temporary table.

Example showing how the database server completes the query

Informix® performs several steps when completing a query for collection-derived tables.

When completing a query, the database server performs the steps shown in this example:

1. Scans the **parent** table to find the row where `parents.id = 1001`
   This operation is listed as a SEQUENTIAL SCAN in the SET EXPLAIN output that Figure 1 shows.
2. Reads the value of the collection column called **children**.

3. Scans the single collection and returns the value of **name** and **id** to the application.
   This operation is listed as a COLLECTION SCAN in the SET EXPLAIN output that Figure 1 shows.

Figure 1. Query plan that uses a collection-derived table

```
QUERY:
------
SELECT name, id
FROM (SELECT children
FROM parents
WHERE parents.id
= 1001) c_table(name, id);

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) lsuto.c_table: COLLECTION SCAN
    Subquery:
    ---------
    Estimated Cost: 1
    Estimated # of Rows Returned: 1

      1) lsuto.parents: SEQUENTIAL SCAN

            Filters: lsuto.parents.id = 1001
```


Derived tables folded into parent queries

You can improve the performance of collection-derived tables by using SQL to fold derived tables in simple queries into a parent query instead of into query results that are put into a temporary table.

Use SQL like that in this example:

```
select * from ((select col1, col2 from tab1)) as vtab(c1,c2)
```

However, if the query is complex because it involves aggregates, ORDER BY operations, or the UNION operation, the server creates a temporary table.

The database server folds derived tables in a manner that is similar to the way the server folds views through the IFX_FOLDVIEW configuration parameter (described in Enable view folding to improve query performance). When the IFX_FOLDVIEW configuration parameter is enabled, views are folded into a parent query. The views are not folded into query results that are put into a temporary table.

The following examples show derived tables folded into the main query.

Figure 1. Query plan that uses a derived table folded into the parent query

```
select * from ((select vcol0, tab1.col1 from
        table(multiset(select col2 from tab2 where col2 > 50 ))
        vtab2(vcol0),tab1 )) vtab1(vcol1,vcol2)
        where vcol1 = vcol2

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) informix.tab2: SEQUENTIAL SCAN

        Filters: informix.tab2.col2 > 50

  2) informix.tab1: SEQUENTIAL SCAN

        Filters:
        Table Scan Filters: informix.tab1.col1 > 50

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.tab2.col2 = informix.tab1.col1
```

Figure 2. Second query plan that uses a derived table folded into the parent query

```
select * from (select col1 from tab1 where col1 = 100) as vtab1(c1)
left join (select col1 from tab2 where col1 = 10) as vtab2(vc1)
on  vtab1.c1 =  vtab2.vc1

Estimated Cost: 5
Estimated # of Rows Returned: 1

      1) informix.tab1: SEQUENTIAL SCAN

            Filters: informix.tab1.col1 = 100

      2) informix.tab2: AUTOINDEX PATH

        (1) Index Keys: col1   (Key-Only)
            Lower Index Filter: informix.tab1.col1 = informix.tab2.col1
            Index Key Filters:  (informix.tab2.col1 = 10 )

    ON-Filters:(informix.tab1.col1 = informix.tab2.col1
            AND informix.tab2.col1 = 10 )
    NESTED LOOP JOIN(LEFT OUTER JOIN)
```

The following example shows a complex query involving the UNION operation. Here, a temporary table has been created.

Figure 3. Complex derived-table query that creates a temporary table

```
select * from (select col1 from tab1 union select col2 from tab2 )
as vtab(vcol1)  where vcol1 < 50

Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) (Temp Table For Collection Subquery): SEQUENTIAL SCAN
```


XML query plans in IBM Data Studio

IBM® Data Studio consists of a set of tools to use for administration, data modeling, and building queries from data that comes from data servers. The EXPLAIN_SQL routine prepares a query and returns a query plan in XML. The IBM Data Studio Administration Edition can use the EXPLAIN_SQL routine to obtain a query plan in XML format, interpret the XML, and render the plan visually.

If you plan to use IBM Data Studio to obtain Visual Explain output, you must create and specify a default sbspace name for the SBSPACENAME configuration parameter in your onconfig file. The EXPLAIN_SQL routine creates BLOBs in this sbspace.

For information about using IBM Data Studio, see IBM Data Studio documentation.

Factors that affect the query plan

When the optimizer determines the query plan, it assigns a cost to each possible plan and then chooses the plan with the lowest cost. The optimizer analyzes several factors to determine the cost of each query plan.

Some of the factors that the optimizer uses to determine the cost of each query plan are:

- The number of I/O requests that are associated with each file system access
- The CPU work that is required to determine which rows meet the query predicate
- The resources that are required to sort or group the data
- The amount of memory available for the query (specified by the DS_TOTAL_MEMORY and DS_MAX_QUERIES parameters)

To calculate the cost of each possible query plan, the optimizer:

- Uses a set of statistics that describes the nature and physical characteristics of the table data and indexes
- Examines the query filters
- Examines the indexes that can be used in the plan
- Uses the cost of moving data to perform joins locally or remotely for distributed queries

For queries that access remote tables in cross-server operations, certain characteristics can significantly degrade performance relative to the corresponding DML operations on tables and views in the local database. Query specifications that can potentially limit performance with remote tables include the following specifications:

- ANSI LEFT OUTER JOIN syntax
- Derived tables based on remote tables
- TEMP tables as materialized views that reference remote tables.

Limitations on remote views
: Reoptimization can occur with multiple executions of queries involving remote views. The optimizer does not pick up the query plans from statement cache even if the statement cache is enabled.

- Statistics held for the table and index
  The accuracy with which the query optimizer can assess the execution cost of a query plan depends on the information available to the optimizer. Use the UPDATE STATISTICS statement to maintain simple statistics about a table and its associated indexes. Updated statistics provide the query optimizer with information that can minimize the amount of time required to perform queries on that table.
- Filters in the query
  The query optimizer bases query-cost estimates on the number of rows to be retrieved from each table. In turn, the estimated number of rows is based on the *selectivity* of each conditional expression that is used within the WHERE clause. A conditional expression that is used to select rows is termed a *filter*.
- Indexes for evaluating a filter
  The query optimizer notes whether an index can be used to evaluate a filter. For this purpose, an indexed column is any single column with an index or the first column named in a composite index.
- Effect of PDQ on the query plan
  When the parallel database query (PDQ) feature is turned on, the optimizer can choose to execute a query in parallel. This can improve performance dramatically when the database server processes queries that decision-support applications initiate.
- Effect of OPTCOMPIND on the query plan
  The OPTCOMPIND setting influences the access plan that the optimizer chooses for single and multiple-table queries. You can change the value of OPTCOMPIND within a session for different kinds of queries.
- Effect of available memory on the query plan
  Informix® constrains the amount of memory that a parallel query can use based on the values of the DS_TOTAL_MEMORY and DS_MAX_QUERIES configuration parameters. If the amount of memory available for the query is too low to execute a hash join, the database server uses a nested-loop join instead.

Previous topic: The query plan

Next topic: Time costs of a query

Statistics held for the table and index

The accuracy with which the query optimizer can assess the execution cost of a query plan depends on the information available to the optimizer. Use the UPDATE STATISTICS statement to maintain simple statistics about a table and its associated indexes. Updated statistics provide the query optimizer with information that can minimize the amount of time required to perform queries on that table.

The database server starts a statistical profile of a table when the table is created, and the profile is refreshed when you issue the UPDATE STATISTICS statement. The query optimizer does not recalculate the profile for tables automatically. In some cases, gathering the statistics might take longer than executing the query.

To ensure that the optimizer selects a query plan that best reflects the current state of your tables, run UPDATE STATISTICS at regular intervals. For guidelines, see Update statistics when they are not generated automatically.

The optimizer uses the following system catalog information as it creates a query plan:

- The number of rows in a table, as of the most recent UPDATE STATISTICS statement
- Whether a column is constrained to be unique
- The distribution of column values, when requested with the MEDIUM or HIGH keyword in the UPDATE STATISTICS statement
  For more information about data distributions, see Creating data distributions.
- The number of disk pages that contain row data

The optimizer also uses the following system catalog information about indexes:

- The indexes that exist on a table, including the columns that they index, whether they are ascending or descending, and whether they are clustered
- The depth of the index structure (a measure of the amount of work that is needed to perform an index lookup)
- The number of disk pages that index entries occupy
- The number of unique entries in an index, which can be used to estimate the number of rows that an equality filter returns
- Second-largest and second-smallest key values in an indexed column

Only the second-largest and second-smallest key values are noted, because the extreme values might have a special meaning that is not related to the rest of the data in the column. The database server assumes that key values are distributed evenly between the second largest and second smallest. Only the initial 4 bytes of these keys are stored. If you create a distribution for a column associated with an index, the optimizer uses that distribution when it estimates the number of rows that match a query.

For more information about system catalog tables, see the *IBM® Informix® Guide to SQL: Reference*.

Filters in the query

The query optimizer bases query-cost estimates on the number of rows to be retrieved from each table. In turn, the estimated number of rows is based on the *selectivity* of each conditional expression that is used within the WHERE clause. A conditional expression that is used to select rows is termed a *filter*.

The selectivity is a value between 0 and 1 that indicates the proportion of rows within the table that the filter can pass. A selective filter, one that passes few rows, has a selectivity near 0, and a filter that passes almost all rows has a selectivity near 1. For guidelines on filters, see Improve filter selectivity.

The optimizer can use data distributions to calculate selectivity for the filters in a query. However, in the absence of data distributions, the database server calculates selectivity for filters of different types based on table indexes. The following table lists some of the selectivity values that the optimizer assigns to filters of different types. Selectivity that is calculated using data distributions is even more accurate than the selectivity that this table shows.

In the table:

- *indexed-col* is the first or only column in an index.
- *2nd-max*, *2nd-min* are the second-largest and second-smallest key values in indexed column.
- The plus sign (+) means logical union (= the Boolean OR operator) and the multiplication symbol (x) means logical intersection (= the Boolean AND operator).

Table 1. Selectivity values that the optimizer assigns to filters of different types

Filter Expression	Selectivity (F)
indexed-col = *literal-value* *indexed-col* = *host-variable* *indexed-col* IS NULL	F = 1/(number of distinct keys in index)
tab1.indexed-col = *tab2.indexed-col*	F = 1/(number of distinct keys in the larger index)
indexed-col > *literal-value*	F = (2nd-max - *literal-value*)/(2nd-max - 2nd-min)
indexed-col < *literal-value*	F = (*literal-value* - 2nd-min)/(2nd-max - 2nd-min)
any-col IS NULL *any-col* = *any-expression*	F = 1/10
any-col > *any-expression* *any-col* < *any-expression*	F = 1/3
any-col MATCHES *any-expression* *any-col* LIKE *any-expression*	F = 1/5
EXISTS *subquery*	F = 1 if *subquery* estimated to return >0 rows, else 0
NOT *expression*	F = 1 - F(*expression*)
expr1 AND *expr2*	F = F(*expr1*) x F(*expr2*)

Filter Expression	Selectivity (F)
expr1 OR *expr2*	F = F(*expr1*) + F(*expr2*) - (F(*expr1*) x F(*expr2*))
any-col IN *list*	Treated as *any-col* = $item_1$ OR . . . OR *any-col* = $item_n$.
any-col relop ANY *subquery*	Treated as any-col relop $value_1$ OR . . . OR *any-col relop* $value_n$ for estimated size of subquery *n*. Here *relop* is any relational operator, such as <, >, >=, <=.

Indexes for evaluating a filter

The query optimizer notes whether an index can be used to evaluate a filter. For this purpose, an indexed column is any single column with an index or the first column named in a composite index.

If the values contained in the index are all that is required, the database server does not read the rows. It is faster to omit the page lookups for data pages whenever the database server can read values directly from the index.

The optimizer can choose an index for any one of the following cases:

- When the column is indexed and a value to be compared is a literal, a host variable, or an uncorrelated subquery
  The database server can locate relevant rows in the table by first finding the row in an appropriate index. If an appropriate index is not available, the database server must scan each table in its entirety.
- When the column is indexed and the value to be compared is a column in another table (a join expression)
  The database server can use the index to find matching values. The following join expression shows such an example:

  ```
  WHERE customer.customer_num = orders.customer_num
  ```

  If rows of **customer** are read first, values of **customer_num** can be applied to an index on **orders.customer_num**.
- When processing an ORDER BY clause
  If all the columns in the clause appear in the required sequence within a single index, the database server can use the index to read the rows in their ordered sequence, thus avoiding a sort.
- When processing a GROUP BY clause
  If all the columns in the clause appear in one index, the database server can read groups with equal keys from the index without requiring additional processing after the rows are retrieved from their tables.

Effect of PDQ on the query plan

When the parallel database query (PDQ) feature is turned on, the optimizer can choose to execute a query in parallel. This can improve performance dramatically when the database server processes queries that decision-support applications initiate.

For more information, see Parallel database query (PDQ).

Effect of OPTCOMPIND on the query plan

The OPTCOMPIND setting influences the access plan that the optimizer chooses for single and multiple-table queries. You can change the value of OPTCOMPIND within a session for different kinds of queries.

To change the value of OPTCOMPIND within a session, use the SET ENVIRONMENT OPTCOMPIND command, not the OPTCOMPIND configuration parameter. For more information about using this command, see Setting the value of OPTCOMPIND within a session.

- Single-table query
  For single-table scans, when OPTCOMPIND is set to `0` or `1` and the current transaction isolation level is Repeatable Read, the optimizer considers two types of access plans.
- Multitable query
  For join plans, the OPTCOMPIND setting influences the access plan for a specific ordered pair of tables.

Single-table query

For single-table scans, when OPTCOMPIND is set to `0` or `1` and the current transaction isolation level is Repeatable Read, the optimizer considers two types of access plans.

If:

- An index is available, the optimizer uses it to access the table.
- No index is available, the optimizer considers scanning the table in physical order.

When OPTCOMPIND is not set in the database server configuration, its value defaults to 2. When OPTCOMPIND is set to 2 or 1 and the current isolation level is not Repeatable Read, the optimizer chooses the least expensive plan to access the table.

Multitable query

For join plans, the OPTCOMPIND setting influences the access plan for a specific ordered pair of tables.

Set OPTCOMPIND to 0 if you want the database server to select a join method exactly as it did in previous versions of the database server. This option ensures compatibility with previous versions.

If OPTCOMPIND is set to 0 or set to 1 and the current transaction isolation level is Repeatable Read, the optimizer gives preference to the nested-loop join.
Important: When OPTCOMPIND is set to 0, the optimizer does not choose a hash join .
If OPTCOMPIND is set to 2 or set to 1 and the transaction isolation level is not Repeatable Read, the optimizer chooses the least expensive query plan from among those previously listed and gives no preference to the nested-loop join.

Effect of available memory on the query plan

Informix® constrains the amount of memory that a parallel query can use based on the values of the DS_TOTAL_MEMORY and DS_MAX_QUERIES configuration parameters. If the amount of memory available for the query is too low to execute a hash join, the database server uses a nested-loop join instead.

For more information about parallel queries and the DS_TOTAL_MEMORY and DS_MAX_QUERIES parameters, see Parallel database query (PDQ).

Time costs of a query

You can adjust a few, but not all, of the response-time effects of actions that the database server performs when processing a query.

The following costs can be reduced by optimal query construction and appropriate indexes:

- Sort time
- Data mismatches
- In-place ALTER TABLE
- Index lookups

For information about how to optimize specific queries, see Improving individual query performance.

- Memory-activity costs
  The database server can process only data in memory. It must read rows into memory to evaluate those rows against the filters of a query. After the server finds rows that satisfy those filters, it prepares an output row in memory by assembling the selected columns.
- Sort-time costs
  A sort requires in-memory work as well as disk work. The in-memory work depends on the number of columns that are sorted, the width of the combined sort key, and the number of row combinations that pass the query filter. You can reduce the cost of sorting.
- Row-reading costs
  When the database server needs to examine a row that is not already in memory, it must read that row from disk. The database server does not read only one row; it reads the entire page that contains the row. If the row spans more than one page, it reads all of the pages.
- Sequential access costs
  Disk costs are lowest when the database server reads the rows of a table in physical order.
- Nonsequential access costs
  The disk-access time is much higher when a disk device reads table pages nonsequentially than when it reads that same table sequentially.
- Index lookup costs
  The database server incurs additional costs when it finds a row through an index. The index is stored on disk, and its pages must be read into memory with the data pages that contain the desired rows.
- In-place ALTER TABLE costs
  For certain conditions, the database server uses an in-place alter algorithm to modify each row when you execute an ALTER TABLE statement. After the alter table operation, the database server inserts rows using the latest definition. If your query accesses rows that are not yet converted to the new table definition, you might notice a slight degradation in the performance of your individual query, because the database server reformats each row in memory before it is returned.
- View costs
  A complex view could run more slowly than expected.
- Small-table costs
  A table is small if it occupies so few pages that it can be retained entirely in the page buffers. Operations on small tables are generally faster than operations on large tables.
- Data-mismatch costs
  An SQL statement can encounter additional costs when the data type of a column that is used in a condition differs from the definition of the column in the CREATE

TABLE statement.

- Encrypted-value costs
  An encrypted value uses more storage space than the corresponding plain-text value because all of the information needed to decrypt the value except the encryption key is stored with the value.
- GLS functionality costs
  Sorting or indexing certain data sets can degrade performance.
- Network-access costs
  Moving data over a network imposes delays in addition to those you encounter with direct disk access.

Previous topic: Factors that affect the query plan
Next topic: Optimization when SQL is within an SPL routine

Memory-activity costs

The database server can process only data in memory. It must read rows into memory to evaluate those rows against the filters of a query. After the server finds rows that satisfy those filters, it prepares an output row in memory by assembling the selected columns.

Most of these activities are performed quickly. Depending on the computer and its workload, the database server can perform hundreds or even thousands of comparisons each second. As a result, the time spent on in-memory work is usually a small part of the execution time.

Although some in-memory activities, such as sorting, take a significant amount of time, it takes much longer to read a row from disk than to examine a row that is already in memory.

Sort-time costs

A sort requires in-memory work as well as disk work. The in-memory work depends on the number of columns that are sorted, the width of the combined sort key, and the number of row combinations that pass the query filter. You can reduce the cost of sorting.

You can use the following formula to calculate the in-memory work that a sort operation requires:

$$W_m = (c * N_{fr}) + (w * N_{fr}\log_2(N_{fr}))$$

W_m
: is the in-memory work.

c
: is the number of columns to order and represents the costs to extract column values from the row and concatenate them into a sort key.

w
: is proportional to the width of the combined sort key in bytes and stands for the work to copy or compare one sort key. A numeric value for w depends strongly on the computer hardware in use.

N_{fr}
: is the number of rows that pass the query filter.

Sorting can involve writing information temporarily to disk if the amount of data to sort is large. You can direct the disk writes to occur in the operating-system file space or in a dbspace that the database server manages. For details, see Configure dbspaces for temporary tables and sort files.

The disk work depends on the number of disk pages where rows appear, the number of rows that meet the conditions of the query predicate, the number of rows that can be placed on a sorted page, and the number of merge operations that must be performed. Use the following formula to calculate the disk work that a sort operation requires:

$$W_d = p + (N_{fr}/N_{rp}) * 2 * (m - 1))$$

W_d
: is the disk work.

p
: is the number of disk pages.

N_{fr}
: is the number of rows that pass the filters.

N_{rp}
: is the number of rows that can be placed on a page.

m
: represents the number of *levels of merge* that the sort must use.

The factor m depends on the number of sort keys that can be held in memory. If there are no filters, N_{fr}/N_{rp} is equivalent to p.

When all the keys can be held in memory, m=1 and the disk work is equivalent to p. In other words, the rows are read and sorted in memory.

For moderate to large tables, rows are sorted in batches that fit in memory, and then the batches are merged. When m=2, the rows are read, sorted, and written in batches. Then the batches are read again and merged, resulting in disk work proportional to the following value:

$$W_d = p + (2 * (N_{fr}/N_{rp}))$$

The more specific the filters, the fewer the rows that are sorted. As the number of rows increases, and the amount of memory decreases, the amount of disk work increases.

To reduce the cost of sorting, use the following methods:

- Make your filters as specific (selective) as possible.
- Limit the projection list to the columns that are relevant to your problem.

Row-reading costs

When the database server needs to examine a row that is not already in memory, it must read that row from disk. The database server does not read only one row; it reads the entire page that contains the row. If the row spans more than one page, it reads all of the pages.

The actual cost of reading a page is variable and hard to predict. The actual cost is a combination of the factors shown in the following table.

Factor	Effect of Factor
Buffering	If the needed page is in a page buffer already, the cost to read is nearly zero.
Contention	If two or more applications require access to the disk hardware, I/O requests can be delayed.
Seek time	The slowest thing that a disk does is to *seek*; that is, to move the access arm to the track that holds the data. Seek time depends on the speed of the disk and the location of the disk arm when the operation starts. Seek time varies from zero to nearly a second.
Latency	The transfer cannot start until the beginning of the page rotates under the access arm. This *latency*, or rotational delay, depends on the speed of the disk and on the position of the disk when the operation starts. Latency can vary from zero to a few milliseconds.

The time cost of reading a page can vary from microseconds for a page that is already in a buffer, to a few milliseconds when contention is zero and the disk arm is already in position, to hundreds of milliseconds when the page is in contention and the disk arm is over a distant cylinder of the disk.

Sequential access costs

Disk costs are lowest when the database server reads the rows of a table in physical order.

When the first row on a page is requested, the disk page is read into a buffer page. After the page is read in, it does not need not to be read again; requests for subsequent rows on that page are filled from the buffer until all the rows on that page are processed. When one page is exhausted, the page for the next set of rows must be read in.

When you use unbuffered devices for dbspaces, and the table is organized properly, the disk pages of consecutive rows are placed in consecutive locations on the disk. This arrangement allows the access arm to move very little when it reads sequentially. In addition, latency costs are usually lower when pages are read sequentially.

Related information:
Read-ahead operations

Nonsequential access costs

The disk-access time is much higher when a disk device reads table pages nonsequentially than when it reads that same table sequentially.

Whenever a table is read in random order, additional disk accesses are required to read the rows in the required order. Disk costs are higher when the rows of a table are read in a sequence unrelated to physical order on disk. Because the pages are not read sequentially from the disk, both seek and rotational delays occur before each page can be read.

Nonsequential access often occurs when you use an index to locate rows. Although index entries are sequential, there is no guarantee that rows with adjacent index entries must reside on the same (or adjacent) data pages. In many cases, a separate disk access must be made to fetch the page for each row located through an index. If a table is larger than the page buffers, a page that contained a row previously read might be cleaned (removed from the buffer and written back to the disk) before a subsequent request for another row on that page can be processed. That page might have to be read in again.

Depending on the relative ordering of the table with respect to the index, you can sometimes retrieve pages that contain several needed rows. The degree to which the physical ordering of rows on disk corresponds to the order of entries in the index is called *clustering*. A highly clustered table is one in which the physical ordering on disk corresponds closely to the index.

Index lookup costs

The database server incurs additional costs when it finds a row through an index. The index is stored on disk, and its pages must be read into memory with the data pages that contain the desired rows.

An index lookup works down from the root page to a leaf page. The root page, because it is used so often, is almost always found in a page buffer. The odds of finding a leaf page in a buffer depend on the size of the index, the form of the query, and the frequency of column-value duplication. If each value occurs only once in the index and the query is a join, each row to be joined requires a nonsequential lookup into the index, followed by a nonsequential access to the associated row in the table.

- Reading duplicate values from an index
  Reading an index with duplicate entries incurs an additional cost over reading the table sequentially.
- Searching for NCHAR or NVARCHAR columns in an index
  A query using an index on an NCHAR or NVARCHAR scans the entire index, resulting in additional time costs.

Reading duplicate values from an index

Reading an index with duplicate entries incurs an additional cost over reading the table sequentially.

Each entry or set of entries with the same value must be located in the index. Then, for each entry in the index, a random access must be made to the table to read the associated row. However, if there are many duplicate rows per distinct index value, and the associated table is highly clustered, the added cost of joining through the index can be slight.

Searching for NCHAR or NVARCHAR columns in an index

A query using an index on an NCHAR or NVARCHAR scans the entire index, resulting in additional time costs.

Global Language Support (GLS) Only
: Indexes that are built on NCHAR or NVARCHAR columns are sorted using a locale-specific comparison value. For example, the Spanish double-l character (ll) might be treated as a single unique character instead of a pair of ls.

  In some locales, the comparison value is not based on the code-set order. The index build uses the locale-specific comparison value to store the key values in the index. As a result, a query using an index on an NCHAR or NVARCHAR scans the entire index because the database server searches the index in code-set order.

In-place ALTER TABLE costs

For certain conditions, the database server uses an in-place alter algorithm to modify each row when you execute an ALTER TABLE statement. After the alter table operation, the database server inserts rows using the latest definition. If your query accesses rows that are not yet converted to the new table definition, you might notice a slight degradation in the performance of your individual query, because the database server reformats each row in memory before it is returned.

For more information about the conditions and performance advantages when an in-place alter occurs, see Altering a table definition.

View costs

A complex view could run more slowly than expected.

You can create views of tables for a number of reasons:

- To limit the data that a user can access
- To reduce the time that it takes to write a complex query
- To hide the complexity of the query that a user needs to write

However, a query against a view might execute more slowly than expected when the complexity of the view definition causes a temporary table to be created to process the query. This temporary table is referred to as a *materialized view*. For example, you can create a view with a union to combine results from several SELECT statements.

The following sample SQL statement creates a view that includes unions:

```
CREATE VIEW view1 (col1, col2, col3, col4)
   AS
      SELECT a, b, c, d
         FROM tab1 WHERE
      UNION
      SELECT a2, b2, c2, d2
         FROM tab2 WHERE
...
      UNION
      SELECT an, bn, cn, dn
         FROM tabn WHERE
;
```

When you create a view that contains complex SELECT statements, the end user does not need to handle the complexity. The end user can just write a simple query, as the following example shows:

```
SELECT a, b, c, d
   FROM view1
```

```
      WHERE a < 10;
```

However, this query against **view1** might execute more slowly than expected because the database server creates a fragmented temporary table for the view before it executes the query.

Another situation when the query might execute more slowly than expected is if you use a view in an ANSI join. The complexity of the view definition might cause a temporary table to be created.

To determine if you have a query that must build a temporary table to process the view, execute the SET EXPLAIN statement. If you see `Temp Table For View` in the SET EXPLAIN output file, your query requires a temporary table to process the view.

Small-table costs

A table is small if it occupies so few pages that it can be retained entirely in the page buffers. Operations on small tables are generally faster than operations on large tables.

As an example, in the **stores_demo** database, the **state** table that relates abbreviations to names of states has a total size of fewer than 1000 bytes; it fits in no more than two pages. This table can be included in any query at little cost. No matter how this table is used, it costs no more than two disk accesses to retrieve this table from disk the first time that it is required.

Data-mismatch costs

An SQL statement can encounter additional costs when the data type of a column that is used in a condition differs from the definition of the column in the CREATE TABLE statement.

For example, the following query contains a condition that compares a column to a data type value that differs from the table definition:

```
CREATE TABLE table1 (a integer, );
SELECT * FROM table1
   WHERE a = '123';
```

The database server rewrites this query before execution to convert `123` to an integer. The SET EXPLAIN output shows the query in its adjusted format. This data conversion has no noticeable overhead.

The additional costs of a data mismatch are most severe when the query compares a character column with a noncharacter value and the length of the number is not equal to the length of the character column. For example, the following query contains a condition in the WHERE clause that equates a character column to an integer value because of missing quotation marks:

```
CREATE TABLE table2 (char_col char(3), );
SELECT * FROM table2
   WHERE char_col = 1;
```

This query finds all of the following values that are valid for **char_col**:

```
' 1'
'001'
'1'
```

These values are not necessarily clustered together in the index keys. Therefore, the index does not provide a fast and correct way to obtain the data. The SET EXPLAIN output shows a sequential scan for this situation.
Warning: The database server does not use an index when the SQL statement compares a character column with a noncharacter value that is not equal in length to the character column.

Encrypted-value costs

An encrypted value uses more storage space than the corresponding plain-text value because all of the information needed to decrypt the value except the encryption key is stored with the value.

Most encrypted data requires approximately 33 percent more storage space than unencrypted data. Omitting the hint used with the password can reduce encryption overhead by up to 50 bytes. If you are using encrypted values, you must make sure that you have sufficient space available for the values.

GLS functionality costs

Sorting or indexing certain data sets can degrade performance.

For information about the performance degradation that occurs from indexing some data sets, see Searching for NCHAR or NVARCHAR columns in an index.

If you do not need a non-ASCII collation sequence, use the CHAR and VARCHAR data types for character columns whenever possible. Because CHAR and VARCHAR data require simple value-based comparison, sorting and indexing these columns is less expensive than for non-ASCII data types (NCHAR or NVARCHAR, for example).

For more information about other character data types, see the *IBM® Informix® GLS User's Guide*.

Network-access costs

Moving data over a network imposes delays in addition to those you encounter with direct disk access.

Network delays can occur when the application sends a query or update request across the network to a database server on another computer. Although the database server performs the query on the remote host computer, that database server returns the output to the application over the network.

Data sent over a network consists of command messages and buffer-sized blocks of row data. Although the details can differ depending on the network and the computers, the database server network activity follows a simple model in which one computer, the *client*, sends a request to another computer, the *server*. The server replies with a block of data from a table.

Whenever data is exchanged over a network, delays are inevitable in the following situations:

- When the network is busy, the client must wait its turn to transmit. Such delays are usually less than a millisecond. However, on a heavily loaded network, these delays can increase exponentially to tenths of seconds and more.
- When the server is handling requests from more than one client, requests might be queued for a time that can range from milliseconds to seconds.
- When the server acts on the request, it incurs the time costs of disk access and in-memory operations that the preceding sections describe.

Transmission of the response is also subject to network delays.

Network access time is extremely variable. In the best case, when neither the network nor the server is busy, transmission and queuing delays are insignificant, and the server sends a row almost as quickly as a local database server might. Furthermore, when the client asks for a second row, the page is likely to be in the page buffers of the server.

Unfortunately, as network load increases, all these factors tend to worsen at the same time. Transmission delays rise in both directions, which increases the queue at the server. The delay between requests decreases the likelihood of a page remaining in the page buffer of the responder. Thus, network-access costs can change suddenly and quite dramatically.

If you use the SELECT FIRST *n* clause in a distributed query, you will still see only the requested amount of data. However, the local database server does not send the SELECT FIRST *n* clause to the remote site. Therefore, the remote site might return more data.

The optimizer that the database server uses assumes that access to a row over the network takes longer than access to a row in a local database. This estimate includes the cost of retrieving the row from disk and transmitting it across the network.

For information about actions that might improve performance across the network, see the following sections:

- Optimizer estimates of distributed queries
- MaxConnect for multiple connections UNIX
- Multiplexed connections and CPU utilization
- Network buffer pools

Optimization when SQL is within an SPL routine

If an SPL routine contains SQL statements, the database server optimizes and executes the SQL within the SPL routine.

The topics in this section contain information about how and when the database server optimizes and executes these routines.

- SQL optimization
  If an SPL routine contains SQL statements, at some point the query optimizer evaluates the possible query plans for SQL in the SPL routine and selects the query plan with the lowest cost. The database server puts the selected query plan for each SQL statement in an execution plan for the SPL routine.
- Execution of an SPL routine
  When the database server executes an SPL routine with the EXECUTE PROCEDURE statement, with the CALL statement, or within an SQL statement, the server performs several activities.
- SPL routine executable format stored in UDR cache
  The first time that a user executes an SPL routine, the database server stores the executable format and any query plans in the UDR cache in the virtual portion of shared memory.

Previous topic: Time costs of a query
Next topic: Trigger execution

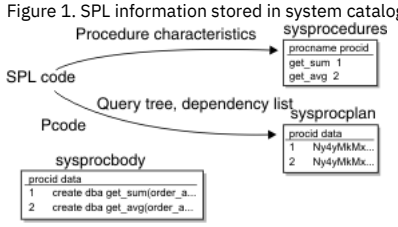
SQL optimization

If an SPL routine contains SQL statements, at some point the query optimizer evaluates the possible query plans for SQL in the SPL routine and selects the query plan with the lowest cost. The database server puts the selected query plan for each SQL statement in an execution plan for the SPL routine.

When you create an SPL routine with the CREATE PROCEDURE statement, the database server attempts to optimize the SQL statements within the SPL routine at that time. If the tables cannot be examined at compile time (because they do not exist or are not available), the creation does not fail. In this case, the database server optimizes the SQL statements the first time that the SPL routine executes.

The database server stores the optimized execution plan in the **sysprocplan** system catalog table for use by other processes. In addition, the database server stores information about the SPL routine (such as procedure name and owner) in the **sysprocedures** system catalog table and an ASCII version of the SPL routine in the **sysprocbody** system catalog table.

Figure 1 summarizes the information that the database server stores in system catalog tables during the compilation process.
Figure 1. SPL information stored in system catalog tables



- Displaying the execution plan
  When you execute an SPL routine, it is already optimized. You can display the query plan for each SQL statement contained in the SPL routine
- Automatic reoptimization
  In some situations, the database server reoptimizes an SQL statement the next time an SPL routine.
- Reoptimizing SPL routines
  You can run an SQL statement that reoptimizes an SPL routine to prevent automatic reoptimization.
- Optimization levels for SQL in SPL routines
  The current optimization level set in an SPL routine affects how the SPL routine is optimized.

Displaying the execution plan

When you execute an SPL routine, it is already optimized. You can display the query plan for each SQL statement contained in the SPL routine

To display the query plan, execute the SET EXPLAIN ON statement prior to one of the following SQL statements that always tries to optimize the SPL routine:

- CREATE PROCEDURE
- UPDATE STATISTICS FOR PROCEDURE
  For example, use the following statements to display the query plan for an SPL routine:

```
SET EXPLAIN ON;
UPDATE STATISTICS FOR PROCEDURE procname;
```


Automatic reoptimization

In some situations, the database server reoptimizes an SQL statement the next time an SPL routine.

If the AUTO_REPREPARE configuration parameter or the IFX_AUTO_REPREPARE session environment variable is disabled, the following error can result when prepared objects or SPL routines are executed after the schema of a table referenced by the prepared object or indirectly referenced by the SPL routine has been modified:

```
-710  Table <table-name> has been dropped, altered, or renamed.
```

The database server uses a dependency list to keep track of changes that would cause reoptimization the next time that an SPL routine executes.

The database server reoptimizes an SQL statement the next time an SPL routine executes after one of the following situations:

- Execution of any data definition language (DDL) statement (such as ALTER TABLE, DROP INDEX, and CREATE INDEX) that might alter the query plan
- Alteration of a table that is linked to another table with a referential constraint (in either direction)
- Execution of UPDATE STATISTICS FOR TABLE for any table involved in the query
  The UPDATE STATISTICS FOR TABLE statement changes the version number of the specified table in **systables.**
- Renaming a column, database, or index with the RENAME statement

Whenever the SPL routine is reoptimized, the database server updates the **sysprocplan** system catalog table with the reoptimized execution plan.

Reoptimizing SPL routines

You can run an SQL statement that reoptimizes an SPL routine to prevent automatic reoptimization.

If you do not want to incur the cost of automatic reoptimization when you first execute an SPL routine after one of the situations that Automatic reoptimization lists, execute the UPDATE STATISTICS statement with the FOR PROCEDURE clause immediately after the situation occurs. In this way, the SPL routine is reoptimized before any

users execute it.

To prevent unnecessary reoptimization of all SPL routines, ensure that you specify a specific procedure name in the FOR PROCEDURE clause.

```
UPDATE STATISTICS FOR PROCEDURE myroutine;
```

For guidelines to run UPDATE STATISTICS, see Update statistics when they are not generated automatically.

Optimization levels for SQL in SPL routines

The current optimization level set in an SPL routine affects how the SPL routine is optimized.

The algorithm that a SET OPTIMIZATION HIGH statement invokes is a sophisticated, cost-based strategy that examines all reasonable query plans and selects the best overall alternative. For large joins, this algorithm can incur more overhead than desired. In extreme cases, you can run out of memory.

The alternative algorithm that a SET OPTIMIZATION LOW statement invokes eliminates unlikely join strategies during the early stages, which reduces the time and resources spent during optimization. However, when you specify a low level of optimization, the optimal strategy might not be selected because it was eliminated from consideration during early stages of the algorithm.

For SPL routines that remain unchanged or change only slightly and that contain complex SELECT statements, you might want to set the SET OPTIMIZATION statement to `HIGH` when you create the SPL routine. This optimization level stores the best query plans for the SPL routine. Then set optimization to `LOW` before you execute the SPL routine. The SPL routine then uses the optimal query plans and runs at the more cost-effective rate if reoptimization occurs.

Execution of an SPL routine

When the database server executes an SPL routine with the EXECUTE PROCEDURE statement, with the CALL statement, or within an SQL statement, the server performs several activities.

The database server performs these activities:

- It reads the interpreter code from the system catalog tables and converts it from a compressed format to an executable format. If the SPL routine is in the UDR cache, the database server retrieves it from the cache and bypasses the conversion step.
- It executes any SPL statements that it encounters.
- When the database server encounters an SQL statement, it retrieves the query plan from the database and executes the statement. If the query plan has not been created, the database server optimizes the SQL statement before it executes.
- When the database server reaches the end of the SPL routine or when it encounters a RETURN statement, it returns any results to the client application. Unless the RETURN statement has a WITH RESUME clause, the SPL routine execution is complete.

SPL routine executable format stored in UDR cache

The first time that a user executes an SPL routine, the database server stores the executable format and any query plans in the UDR cache in the virtual portion of shared memory.

When another user executes an SPL routine, the database server first checks the UDR cache. SPL execution performance improves when the database server can execute the SPL routine from the UDR cache. The UDR cache also stores UDRs, user-defined aggregates, and extended data types definitions.

- Adjust the UDR cache
  The default number of SPL routines, UDRs, and other user-defined definitions in the UDR cache is 127. You can change the number of entries with the PC_POOLSIZE configuration parameter.

Related reference:
Configure and monitor memory caches

Adjust the UDR cache

The default number of SPL routines, UDRs, and other user-defined definitions in the UDR cache is 127. You can change the number of entries with the PC_POOLSIZE configuration parameter.

The database server uses a hashing algorithm to store and locate SPL routines in the UDR cache. You can modify the number of *buckets* in the UDR cache with the PC_HASHSIZE configuration parameter. For example, if the value of the PC_POOLSIZE configuration parameter is `100` and the value of the PC_HASHSIZE configuration parameter is `10`, each bucket can have up to 10 SPL routines and UDRs.

Too many buckets cause the database server to move out cached SPL routines when the bucket fills. Too few buckets increase the number of SPL routines in a bucket, and the database server must search though the SPL routines in a bucket to determine if the SPL routine that it needs is there.

When the number of entries in a bucket reaches 75 percent, the database server removes the least recently used SPL routines from the bucket (and hence from the UDR cache) until the number of SPL routines in the bucket is 50 percent of the maximum SPL routines in the bucket.

Monitor the UDR cache by running the **onstat -g prc** command. If the numbers in the **hits** fields are not evenly distributed among buckets, increase the value of the PC_HASHSIZE configuration parameter. Adjust the number of buckets to have the least number of high hit entries per bucket.

Important: PC_POOLSIZE and PC_HASHSIZE also control other memory caches for the database server (excluding the buffer pool, the SQL statement cache, the data distribution cache, and the data-dictionary cache). When you modify the size and number of hash buckets for SQL routines, you also modify the size and number of hash buckets for the other caches (such as the aggregate cache, oplcass, and typename cache).

Related information:

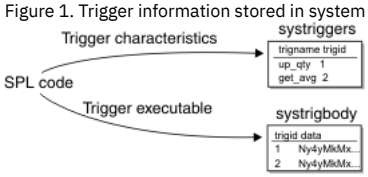
onstat -g prc command: Print sessions using UDR or SPL routines
PC_POOLSIZE configuration parameter
PC_HASHSIZE configuration parameter

Trigger execution

A *trigger* is a database object that automatically executes one or more SQL statements (the *triggered action*) when a specified data manipulation language operation (the *triggering event*) occurs. You can define one or more triggers on a table to execute after a SELECT, INSERT, UPDATE or DELETE triggering event.

You can also define INSTEAD OF triggers on a view. These triggers specify the SQL statements to be executed as triggered actions on the underlying table when a triggering INSERT, UPDATE or DELETE statement attempts to modify the view. These triggers are called INSTEAD OF triggers because only the triggered SQL action is executed; the triggering event is not executed. For more information about using triggers, see the *IBM® Informix® Guide to SQL: Tutorial* and information about the CREATE TRIGGER statement in the *IBM Informix Guide to SQL: Syntax*.

Figure 1. Trigger information stored in system catalog tables



When you use the CREATE TRIGGER statement to register a new trigger, the database server:

- Stores information about the trigger in the **systriggers** system catalog table.
- Stores the text of the statements that the trigger executes in the **systrigbody** system catalog table.

The **sysprocedures** system catalog table identifies trigger routines that can be invoked only as triggered actions.

Memory-resident tables of the **sysmaster** database indicate whether the table or view has triggers on it.

Whenever a SELECT, INSERT, UPDATE, or DELETE statement is issued, the database server checks to see if the statement is a *triggering event* that activates a trigger for the table and columns (or for the view) on which the DML statement operates. If the statement requires activating triggers, the database server retrieves the statement text of the triggered actions from the **systrigbody** table and runs the triggered DML statements or SPL routine before, during, or after the triggering events. For INSTEAD OF triggers on a view, the database server performs the triggered actions instead of the triggering events.

- Performance implications for triggers
  In many situations, triggers can improve performance slightly because of the reduction in the number of messages passed from the client to the database server.

Previous topic: Optimization when SQL is within an SPL routine

Performance implications for triggers

In many situations, triggers can improve performance slightly because of the reduction in the number of messages passed from the client to the database server.

For example, if the trigger fires five SQL statements, the client saves at least 10 messages passed between the client and database server (one to send the SQL statement and one for the reply after the database server executes the SQL statement). Triggers improve performance the most when they execute more SQL statements and the network speed is comparatively slow.

When the database server executes an SQL statement, it must perform the following actions:

- Determine if triggers must be fired
- Retrieve the triggers from **systriggers** and **systrigbody**

These operations cause only a slight performance impact that can be offset by the decreased number of messages passed between the client and the server.

However, triggers executed on SELECT statements have additional performance implications. The following sections explain these implications.

- SELECT triggers on tables in a table hierarchy
  When the database server executes a SELECT statement that includes a table that is involved in a table hierarchy, and the SELECT statement fires a SELECT trigger, performance might be slower if the SELECT statement that invokes the trigger involves a join, sort, or materialized view.
- SELECT triggers and row buffering
  The lack of buffering for SELECT statements that fire SELECT triggers might reduce performance slightly compared to an identical SELECT statement that does not fire a SELECT trigger.

SELECT triggers on tables in a table hierarchy

When the database server executes a SELECT statement that includes a table that is involved in a table hierarchy, and the SELECT statement fires a SELECT trigger, performance might be slower if the SELECT statement that invokes the trigger involves a join, sort, or materialized view.

In this case, the database server does not know which columns are affected in the table hierarchy, so it can execute the query differently. The following behaviors might occur:

- Key-only index scans are disabled on the table that is involved in a table hierarchy.
- If the database server needs to sort data selected from a table involved in a table hierarchy, it copies all of the columns in the SELECT list to the temporary table, not just the sort columns.
- If the database server uses the table included in the table hierarchy to build a hash table for a hash join with another table, it bypasses the early projection, meaning it uses all of the columns from the table to build the hash table, not just the columns in the join.
- If the SELECT statement contains a materialized view (meaning a temporary table must be built for the columns in a view) that contains columns from a table involved in a table hierarchy, all columns from the table are included in the temporary table, not just the columns actually contained in the view.

SELECT triggers and row buffering

The lack of buffering for SELECT statements that fire SELECT triggers might reduce performance slightly compared to an identical SELECT statement that does not fire a SELECT trigger.

In SELECT statements whose tables do not fire SELECT triggers, the database server sends more than one row back to the client and stores the rows in a buffer even though the client application requested only one row with a FETCH statement. However, for SELECT statements that contain one or more tables that fire a SELECT trigger, the database server sends only the requested row back to the client instead of a buffer full. The database server cannot return other rows to the client until the trigger action occurs.

Optimizer directives

Optimizer directives are comments that tell the query optimizer how to execute a query. You can use optimizer directives to improve query performance.

- What optimizer directives are
  Optimizer directives are specifications formatted as comments that provide information to the query optimizer about how to execute a query.
- Reasons to use optimizer directives
  In most cases, the optimizer chooses the fastest query plan. You can use optimizer directives when the optimizer does not choose the best query plan to perform a query, because of the complexity of the query, or because the query does not have enough information about the nature of the data. A poor query plan produces poor performance.
- Preparation for using directives
  In most cases, the optimizer chooses the fastest query plan. However, you can take steps to assist the optimizer and to prepare for using directives.
- Guidelines for using directives
  Guidelines for directives include frequently analyzing the effectiveness of the query and using negative directives.
- Types of optimizer directives that are supported in SQL statements
  Directives that are in SQL statements are embedded in queries. These directives include access-method directives, join-order directives, join-plan directives, and optimization-goal directives.
- Configuration parameters and environment variables for optimizer directives
  You can use the DIRECTIVES configuration parameter to turn on or off all directives that the database server encounters, and you can use the **IFX_DIRECTIVES** environment variable to override the setting of the DIRECTIVES configuration parameter.
- Optimizer directives and SPL routines
  Directives operate differently for a query in an SPL routine because a SELECT statement in an SPL routine is not necessarily optimized immediately before the database server executes it.
- Avoiding index or prepared object exceptions by forced reoptimization
  If the AUTO_REPREPARE configuration parameter and the IFX_AUTO_REPREPARE session environment variable are enabled, Informix automatically recompiles prepared statements and SPL routines after the schema of a referenced table is modified by a DDL statement. If the AUTO_REPREPARE configuration parameter or the IFX_AUTO_REPREPARE session environment variable is disabled, you can take steps to prevent errors.
- External optimizer directives
  If you are user **informix**, you can create, save, and delete external directives.

What optimizer directives are

Optimizer directives are specifications formatted as comments that provide information to the query optimizer about how to execute a query.

You can use two kinds of optimizer directives:

- Optimizer directives in the form of instructions that are embedded in queries (For more information, see Optimizer directives that are embedded in queries.

- External optimizer directives that you create and save for use as temporary workaround solutions to problems when you do not want to change SQL statements in queries. (For more information, see External optimizer directives.)

- Optimizer directives that are embedded in queries
  Optimizer directives embedded in queries are comments in a SELECT statement that provide information to the query optimizer on how to execute a query. You can also place directives in UPDATE and DELETE statements, instructing the optimizer how to access the data.
- External optimizer directives
  External optimizer directives are optimizer directives that an administrator can create and store in the **sysdirectives** catalog table. The administrator can then use an ONCONFIG variable to make the directives available.

Related concepts:
Reasons to use optimizer directives
Preparation for using directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Optimizer directives that are embedded in queries

Optimizer directives embedded in queries are comments in a SELECT statement that provide information to the query optimizer on how to execute a query. You can also place directives in UPDATE and DELETE statements, instructing the optimizer how to access the data.

Optimizer directives can either be explicit directions (for example, "use this index" or "access this table first"), or they can eliminate possible query plans (for example, "do not read this table sequentially" or "do not perform a nested-loop join").

External optimizer directives

External optimizer directives are optimizer directives that an administrator can create and store in the **sysdirectives** catalog table. The administrator can then use an ONCONFIG variable to make the directives available.

Client users also specify an environment variable and can choose to use these optimizer directives in queries in situations when they do not want to insert comments in SQL statements.

External directives are useful when it is not feasible to rewrite a query for a short-term solution to a problem, for example, when a query starts to perform poorly. Rewriting the query by changing the SQL statement is preferable for long-term solutions to problems.

External directives are for occasional use only. The number of directives stored in the **sysdirectives** catalog should not exceed 50. A typical enterprise only needs 0 to 9 directives.

Reasons to use optimizer directives

In most cases, the optimizer chooses the fastest query plan. You can use optimizer directives when the optimizer does not choose the best query plan to perform a query, because of the complexity of the query, or because the query does not have enough information about the nature of the data. A poor query plan produces poor performance.

Before you decide when to use optimizer directives, you should understand what makes a good query plan.

The optimizer creates a query plan based on costs of using different table-access paths, join orders, and join plans.

Some query plan guidelines are:

- Do not use an index when the database server must read a large portion of the table. For example, the following query might read most of the **customer** table:

  ```
  SELECT * FROM customer WHERE STATE <> "ALASKA";
  ```

  Assuming the customers are evenly spread among all 50 states, you might estimate that the database server must read 98 percent of the table. It is more efficient to read the table sequentially than to traverse an index (and subsequently the data pages) when the database server must read most of the rows.

- When you choose between indexes to access a table, use an index that can rule out the most rows. For example, consider the following query:

  ```
  SELECT * FROM customer
  WHERE state = "NEW YORK" AND order_date = "01/20/97"
  ```

  Assuming that 200,000 customers live in New York and only 1000 customers ordered on any one day, the optimizer most likely chooses an index on **order_date** rather than an index on **state** to perform the query.

- Place small tables or tables with restrictive filters early in the query plan. For example, consider the following query:

```
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
      AND
   customer.state = "NEVADA";
```

  In this example, if you read the **customer** table first, you can rule out most of the rows by applying the filter that chooses all rows in which `state = "NEVADA"`.

  By ruling out rows in the **customer** table, the database server does not read as many rows in the **orders** table (which might be significantly larger than the **customer** table).

- Choose a hash join when neither column in the join filter has an index.
  In the previous example, if **customer.customer_num** and **orders.customer_num** are not indexed, a hash join is probably the best join plan.

- Choose nested-loop joins if:
  - The number of rows retrieved from the outer table after the database server applies any table filters is small, and the inner table has an index that can be used to perform the join.
  - The index on the outermost table can be used to return rows in the order of the ORDER BY clause, eliminating the need for a sort.

For information about query plans, see The query plan. For more information about directives, see

- Preparation for using directives
- Guidelines for using directives
- Types of optimizer directives that are supported in SQL statements

Related concepts:
What optimizer directives are
Preparation for using directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Preparation for using directives

In most cases, the optimizer chooses the fastest query plan. However, you can take steps to assist the optimizer and to prepare for using directives.

To prepare for using directives, make sure that you perform the following tasks:

- Run UPDATE STATISTICS.
  Without accurate statistics, the optimizer cannot choose the appropriate query plan. Run UPDATE STATISTICS any time that the data in the tables changes significantly (many new rows are added, updated, or deleted). For more information, see Update the statistics for the number of rows.

- Create distributions.
  One of the first things that you should try when you attempt to improve a slow query is to create distributions on columns involved in a query. Distributions provide the most accurate information to the optimizer about the nature of the data in the table. Run UPDATE STATISTICS HIGH on columns involved in the query filters to see if performance improves. For more information, see Creating data distributions.

In some cases, the query optimizer does not choose the best query plan because of the complexity of the query or because (even with distributions) it does not have enough information about the nature of the data. In these cases, you can attempt to improve performance for a particular query by using directives.

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Guidelines for using directives

Guidelines for directives include frequently analyzing the effectiveness of the query and using negative directives.

Consider the following guidelines:

- Examine the effectiveness of a particular directive frequently to make sure it continues to operate effectively. Imagine a query in a production program with several directives that force an optimal query plan. Some days later, users add, update, or delete a large number of rows, which changes the nature of the data so much that the once optimal query plan is no longer effective. This example illustrates how you must use directives with care.

- Use negative directives (such as AVOID_NL, AVOID_FULL, and so on) whenever possible. When you exclude a behavior that degrades performance, you rely on the optimizer to use the next-best choice rather than attempt to force a path that might not be optimal.

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Preparation for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Types of optimizer directives that are supported in SQL statements

Directives that are in SQL statements are embedded in queries. These directives include access-method directives, join-order directives, join-plan directives, and optimization-goal directives.

Include the directives in the SQL statement as a comment that occurs immediately after the SELECT, UPDATE, or DELETE keyword. The first character in a directive is always a plus (+) sign. In the following query, the ORDERED directive specifies that the tables should be joined in the same order as they are listed in the FROM clause. The AVOID_FULL directive specifies that the optimizer should discard any plans that include a full table scan on the listed table (**employee**).

```
SELECT --+ORDERED, AVOID_FULL(e) * FROM employee e, department d
> 50000;
```

For a complete syntax description for directives, see the *IBM® Informix Guide to SQL: Syntax*.

To influence the choice of a query plan that the optimizer makes, you can alter the following aspects of a query:

- Access method
- Join order
- Join method
- Optimization goal
- Star-join directives

You can also use EXPLAIN directives instead of the SET EXPLAIN statement to show the query plan. The following sections describe these aspects in detail.

- Access-method directives
  The database server uses an access method to access a table. The server can either read the table sequentially via a full table scan or use any one of the indexes on the table. Access-method directives influence the access method.
- Join-order directives
  The join-order directive ORDERED tells the optimizer to join tables in the order that the SELECT statement lists them.
- Join-method directives
  The join-method directives influence how the database server joins two tables in a query.
- Optimization-goal directives
  In some queries, you might want to find only the first few rows in the result of a query. Or, you might know that all rows must be accessed and returned from the query. You can use the optimization-goal directives to find the first row that satisfies the query or all rows that satisfy the query.
- Star-join directives
  Star-join directives can specify how the query optimizer joins two or more tables, among which one or more dimension tables have foreign-key dependencies on one or more fact tables.
- EXPLAIN directives
  You can use the EXPLAIN directives to display the query plan that the optimizer chooses, and you can specify to display the query plan without running the query.
- Example of directives that can alter a query plan
  Directives can alter the query plan. You can use particular directives to force the optimizer to choose a particular type of query plan, for example one that uses hash joins and the order of tables as they appear in the query.

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Preparation for using directives
Guidelines for using directives
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Access-method directives

The database server uses an access method to access a table. The server can either read the table sequentially via a full table scan or use any one of the indexes on the table. Access-method directives influence the access method.

The following table lists the directives that influence the access method:

Access-Method Directive	Description
INDEX	Tells the optimizer to use the index specified to access the table. If the directive lists more than one index, the optimizer chooses the index that yields the least cost.
AVOID_INDEX	Tells the optimizer not use any of the indexes listed. You can use this directive with the AVOID_FULL directive.
INDEX_SJ	Forces an index self-join path using the specified index, or choosing the least costly index in a list of indexes, even if data distribution statistics are not available for the leading index key columns of the index. For information about index self-join paths, see Query plans that include an index self-join path.
AVOID_INDEX_SJ	Tells the optimizer not to use an index self-join path for the specified index or indexes.
FULL	Tells the optimizer to perform a full table scan.
AVOID_FULL	Tells the optimizer not to perform a full table scan on the listed table. You can use this directive with the AVOID_INDEX directive.
INDEX_ALL or MULTI_INDEX	Access the table by using the specified indexes for a multi-index scan. The INDEX_ALL and MULTI_INDEX keywords are synonyms.
AVOID_MULTI_INDEX	Tells the optimizer not to consider a multi-index scan path for the specified table.

In some cases, forcing an access method can change the join method that the optimizer chooses. For example, if you exclude the use of an index with the AVOID_INDEX directive, the optimizer might choose a hash join instead of a nested-loop join.

The optimizer considers an index self-join path only if all of the following conditions are met:

- The index does not have functional keys, user-defined types, built-in opaque types, or non-B-tree indexes
- Data distribution statistics are available for the index key column under consideration
- The number of rows in the table is at least 10 times the number of unique combinations of all possible lead-key column values.

If all of these conditions are met, the optimizer estimates the cost of an index self-join path and compares it with the costs of alternative access methods. The optimizer then picks the best access method for the table. For more information about the access-method directives and some examples of their usage, see the *IBM® Informix® Guide to SQL: Syntax*.

Join-order directives

The join-order directive ORDERED tells the optimizer to join tables in the order that the SELECT statement lists them.

- Effect of join order on join plan
  By specifying the join order, you might affect more than just how tables are joined.
- Join order when you use views
  The ORDERED directive that is inside a view or is in a query that contains a view affect the join order.

Effect of join order on join plan

By specifying the join order, you might affect more than just how tables are joined.

For example, consider the following query:

```
SELECT --+ORDERED, AVOID_FULL(e)
* FROM employee e, department d
WHERE e.dept_no = d.dept_no AND e.salary > 5000
```

In this example, the optimizer chooses to join the tables with a hash join. However, if you arrange the order so that the second table is **employee** (and must be accessed by an index), the hash join is not feasible.

```
SELECT --+ORDERED, AVOID_FULL(e)
* FROM department d, employee e
WHERE e.dept_no = d.dept_no AND e.salary > 5000;
```

The optimizer chooses a nested-loop join in this case.

Join order when you use views

The ORDERED directive that is inside a view or is in a query that contains a view affect the join order.

Two cases can affect join order when you use views:

- The ORDERED directive is inside the view.
  The ORDERED directive inside a view affects the join order of only the tables inside the view. The tables in the view must be joined contiguously. Consider the following view and query:

```
CREATE VIEW emp_job_view as
   SELECT {+ORDERED}
   emp.job_num, job.job_name
   FROM emp, job
   WHERE emp.job_num = job.job_num;

SELECT * from dept, emp_job_view,  project
   WHERE dept.dept_no = project.dept_num
   AND emp_job_view.job_num = project.job_num;
```

  The ORDERED directive specifies that the **emp** table come before the job table. The directive does not affect the order of the **dept** and **project** table. Therefore, all possible join orders are as follows:

  - **emp**, **job**, **dept**, **project**
  - **emp**, **job**, **project**, **dept**
  - **project**, **emp**, **job**, **dept**
  - **dept**, **emp**, **job**, **project**
  - **dept**, **project**, **emp**, **job**
  - **project**, **dept**, **emp**, **job**
- The ORDERED directive is in a query that contains a view.
  If an ORDERED directive appears in a query that contains a view, the join order of the tables in the query are the same as they are listed in the SELECT statement. The tables within the view are joined as they are listed within the view.

  In the following query, the join order is **dept**, **project**, **emp**, **job**:

```
CREATE VIEW emp_job_view AS
   SELECT
   emp.job_num, job.job_name
   FROM emp, job
   WHERE emp.job_num = job.job_num;
SELECT {+ORDERED}
   * FROM dept, project, emp_job_view
   WHERE dept.dept_no = project.dept_num
   AND emp_job_view.job_num = project.job_num;
```

  An exception to this rule is when the view cannot be folded into the query, as in the following example:

```
CREATE VIEW emp_job_view2 AS
   SELECT DISTINCT
   emp.job_num, job.job_name
   FROM emp,job
   WHERE emp.job_num = job.job_num;
```

  In this example, the database server executes the query and puts the result in a temporary table. The order of tables in this query is **dept**, **project**, **temp_table**.

Join-method directives

The join-method directives influence how the database server joins two tables in a query.

The following directives influence the join method between two tables:

- USE_NL
  Use the listed tables as the inner table in a nested-loop join.
- USE_HASH
  Access the listed tables with a hash join. You can also choose whether the table is used to create the hash table or to probe the hash table.
- AVOID_NL
  Do not use the listed tables as the inner table in a nested-loop join. A table listed with this directive can still participate in a nested-loop join as an outer table.
- AVOID_HASH
  Do not access the listed tables with a hash join. Optionally, you can allow a hash join but restrict the table from being the one that is probed or the table from which the hash table is built.

You can specify the keyword /BUILD after the name of a table in a USE_HASH or AVOID_HASH optimizer directives:

- With USE_HASH directives, the /BUILD keyword tells the optimizer to use the specified table to build the hash table.
- With AVOID_HASH, the /BUILD keyword tells the optimizer to avoid using the specified table to build the hash table.

You can specify the keyword /PROBE after the name of a table in a USE_HASH or AVOID_HASH optimizer directives:

- With USE_HASH directives, the /PROBE keyword tells the optimizer to use the specified table to probe the hash table.
- With AVOID_HASH directives, the /PROBE keyword tells the optimizer to avoid using the specified table to probe the hash table.

Optimization-goal directives

In some queries, you might want to find only the first few rows in the result of a query. Or, you might know that all rows must be accessed and returned from the query. You can use the optimization-goal directives to find the first row that satisfies the query or all rows that satisfy the query.

For example, you might want to find only the first few rows in the result of a query, because the Informix® ESQL/C program opens a cursor for the query and performs a FETCH to find only the first row.

Use the optimization-goal directives to optimize the query for either one of these cases:

- FIRST_ROWS
  Choose a plan that optimizes the process of finding only the first row that satisfies the query.
- ALL_ROWS
  Choose a plan that optimizes the process of finding all rows (the default behavior) that satisfy the query.

If you use the FIRST_ROWS directive, the optimizer might abandon a query plan that contains activities that are time-consuming up front. For example, a hash join might take too much time to create the hash table. If only a few rows must be returned, the optimizer might choose a nested-loop join instead.

In the following example, assume that the database has an index on **employee.dept_no** but not on **department.dept_no**. Without directives, the optimizer chooses a hash join.

```
SELECT *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```

However, with the FIRST_ROWS directive, the optimizer chooses a nested-loop join because of the high initial overhead required to create the hash table.

```
SELECT {+first_rows} *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```


Star-join directives

Star-join directives can specify how the query optimizer joins two or more tables, among which one or more dimension tables have foreign-key dependencies on one or more fact tables.

The following directives can influence the join plan for tables that logically participate in a star schema or in a snowflake schema:

- FACT
  The optimizer considers a query plan in which the specified table is a fact table in a star-join execution plan.
- AVOID_FACT
  The optimizer does not consider a star-join execution plan that treats the specified table (or any of the tables in the list of tables) as a fact table.
- STAR_JOIN
  The optimizer favors a star-join execution plan, if available.
- AVOID_STAR_JOIN
  The optimizer chooses a query execution plan that is not a star-join plan.

These star-join directives have no effect unless the parallel database query feature (PDQ) is enabled.

Related information:
Star-Join Directives
Concepts of dimensional data modeling
Keys to join the fact table with the dimension tables
Use the snowflake schema for hierarchical dimension tables

EXPLAIN directives

You can use the EXPLAIN directives to display the query plan that the optimizer chooses, and you can specify to display the query plan without running the query.

You can use these directives:

- EXPLAIN
  Displays the query plan that the optimizer chooses.
- EXPLAIN AVOID_EXECUTE
  Displays the query plan that the optimizer chooses, but does not run the query.

When you want to display the query plan for one SQL statement only, use these EXPLAIN directives instead of the SET EXPLAIN ON or SET EXPLAIN ON AVOID_EXECUTE statements.

When you use AVOID_EXECUTE (either the directive or in the SET EXPLAIN statement), the query does not execute but displays the following message:

```
No rows returned.
```

Figure 1 shows sample output for a query that uses the EXPLAIN AVOID_EXECUTE directive.

Figure 1. Result of EXPLAIN AVOID_EXECUTE directives

```
QUERY:
------
select --+ explain avoid_execute
  l.customer_num, l.lname, l.company,
  l.phone, r.call_dtime, r.call_descr
from customer l, cust_calls r
where l.customer_num = r.customer_num

DIRECTIVES FOLLOWED:
EXPLAIN
AVOID_EXECUTE
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 7

  1) informix.r: SEQUENTIAL SCAN

  2) informix.l: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.l.customer_num = informix.r.customer_num
NESTED LOOP JOIN
```

The following table describes the pertinent output lines in Figure 1 that describe the chosen query plan.

Output Line in Figure 1	**Chosen Query Plan Description**
`DIRECTIVES FOLLOWED: EXPLAIN AVOID_EXECUTE`	Use the directives EXPLAIN and AVOID_EXECUTE to display the query plan and do not execute the query.
`Estimated # of Rows Returned: 7`	Estimate that this query returns seven rows.
`Estimated Cost: 7`	This estimated cost of `7` is a value that the optimizer uses to compare different query plans and select the one with the lowest cost.
`1) informix.r: SEQUENTIAL SCAN`	Use the **cust_calls r** table as the outer table and scan it to obtain each row.
`2) informix.l: INDEX PATH`	For each row in the outer table, use an index to obtain the matching row(s) in the inner table **customer l**.
`(1) Index Keys: customer_num (Serial, fragments: ALL)`	Use the index on the **customer_num** column, scan it serially, and scan all fragments (the **customer l** table consists of only one fragment).
`Lower Index Filter: informix.l.customer_num = informix.r.customer_num`	Start the index scan at the **customer_num** value from the outer table.

Example of directives that can alter a query plan

Directives can alter the query plan. You can use particular directives to force the optimizer to choose a particular type of query plan, for example one that uses hash joins and the order of tables as they appear in the query.

The following example shows how directives can alter the query plan.

Suppose you have the following query:

```
SELECT * FROM emp,job,dept
WHERE emp.location = 10
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER";
```

Assume that the following indexes exist:

```
ix1: emp(empno,jobno,deptno,location)
ix2: job(jobno)
ix3: dept(location)
```

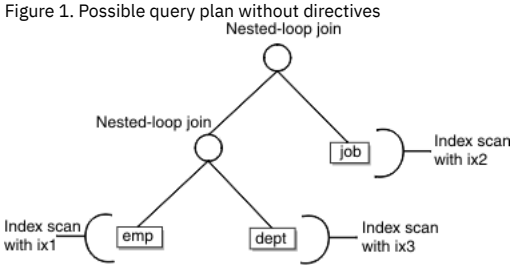
You run the query with SET EXPLAIN ON to display the query path that the optimizer uses.

```
QUERY:
------
SELECT * FROM emp,job,dept
WHERE emp.location = "DENVER"
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER"

Estimated Cost: 5
Estimated # of Rows Returned: 1

1) informix.emp: INDEX PATH

    Filters: informix.emp.location = 'DENVER'

    (1) Index Keys: empno jobno deptno location   (Key-Only)

2) informix.dept: INDEX PATH
```

```
    Filters: informix.dept.deptno = informix.emp.deptno

    (1) Index Keys: location
        Lower Index Filter: informix.dept.location = 'DENVER'
NESTED LOOP JOIN

3) informix.job: INDEX PATH

    (1) Index Keys: jobno   (Key-Only)
        Lower Index Filter: informix.job.jobno = informix.emp.jobno
NESTED LOOP JOIN
```

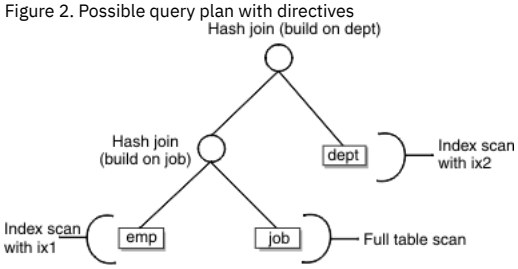
The diagram in Figure 1 shows a possible query plan for this query. The query plan has three levels of information: (1) a nested-loop join, (2) an index scan on one table and a nested-loop join, and (3) index scans on two other tables.

Figure 1. Possible query plan without directives



Perhaps you are concerned that using a nested-loop join might not be the fastest method to execute this query. You also think that the join order is not optimal. You can force the optimizer to choose a hash join and order the tables in the query plan according to their order in the query, so the optimizer uses the query plan that Figure 2 shows. This query plan that has three levels of information: (1) a hash join, (2) an index scan and a hash join, and (3) an index scan on two other tables.

Figure 2. Possible query plan with directives



To force the optimizer to choose the query plan that uses hash joins and the order of tables shown in the query, use the directives that the following partial SET EXPLAIN output shows:

```
QUERY:
------
SELECT {+ORDERED,
   INDEX(emp ix1),
   FULL(job),
   USE_HASH(job /BUILD),
   USE_HASH(dept /BUILD),
   INDEX(dept ix3)}
   * FROM emp,job,dept
   WHERE emp.location = 1
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER"

DIRECTIVES FOLLOWED:
ORDERED
INDEX ( emp ix1 )
FULL ( job )
USE_HASH ( job/BUILD )
USE_HASH ( dept/BUILD )
INDEX ( dept ix3 )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 1

1) informix.emp: INDEX PATH

    Filters: informix.emp.location = 'DENVER'

    (1) Index Keys: empno jobno deptno location   (Key-Only)

2) informix.job: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.emp.jobno = informix.job.jobno

3) informix.dept: INDEX PATH

    (1) Index Keys: location
        Lower Index Filter: informix.dept.location = 'DENVER'

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.emp.deptno = informix.dept.deptno
```

Configuration parameters and environment variables for optimizer directives

You can use the DIRECTIVES configuration parameter to turn on or off all directives that the database server encounters, and you can use the **IFX_DIRECTIVES** environment variable to override the setting of the DIRECTIVES configuration parameter.

If the DIRECTIVES configuration parameter is set to `1` (the default), the optimizer follows all directives. If the DIRECTIVES configuration parameter is set to `0`, the optimizer ignores all directives.

You can override the setting of DIRECTIVES. If the **IFX_DIRECTIVES** environment variable is set to `1` or `ON`, the optimizer follows directives for any SQL the client session executes. If **IFX_DIRECTIVES** is `0` or `OFF`, the optimizer ignores directives for any SQL in the client session.

Any directives in an SQL statement take precedence over the join plan that the OPTCOMPIND configuration parameter forces. For example, if a query includes the USE_HASH directive and OPTCOMPIND is set to `0` (nested-loop joins preferred over hash joins), the optimizer uses a hash join.

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Preparation for using directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Optimizer directives and SPL routines

Directives operate differently for a query in an SPL routine because a SELECT statement in an SPL routine is not necessarily optimized immediately before the database server executes it.

The optimizer creates a query plan for a SELECT statement in an SPL routine when the database server creates the SPL routine or during the execution of the UPDATE STATISTICS statement that include the `FOR FUNCTION`, `FOR PROCEDURE`, or `FOR ROUTINE` keywords.

The optimizer reads and applies directives at the time that it creates the query plan. Because it stores the query plan in a system catalog table, the SELECT statement is not reoptimized when it is executed. Therefore, settings of **IFX_DIRECTIVES** and DIRECTIVES affect SELECT statements inside an SPL routine when they are set at any of the following times:

- Before the CREATE PROCEDURE statement
- Before the UPDATE STATISTICS FOR ROUTINE statements that cause SQL data-manipulation statements in SPL routines to be optimized
- During certain circumstances when SELECT statements have variables supplied at runtime

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Preparation for using directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Avoiding index or prepared object exceptions by forced reoptimization
Related tasks:
External optimizer directives

Avoiding index or prepared object exceptions by forced reoptimization

If the AUTO_REPREPARE configuration parameter and the IFX_AUTO_REPREPARE session environment variable are enabled, Informix® automatically recompiles prepared statements and SPL routines after the schema of a referenced table is modified by a DDL statement. If the AUTO_REPREPARE configuration parameter or the IFX_AUTO_REPREPARE session environment variable is disabled, you can take steps to prevent errors.

If the AUTO_REPREPARE configuration parameter or the IFX_AUTO_REPREPARE session environment variable is disabled, the following error can result when prepared objects or SPL routines are executed after the schema of a table referenced by the prepared object or indirectly referenced by the SPL routine has been modified.

```
-710  Table <table-name> has been dropped, altered, or renamed.
```

This error can occur with explicitly prepared statements. These statements have the following form:

```
PREPARE statement_id FROM quoted_string;
```

After a statement has been prepared in the database server and before execution of the statement, a table to which the statement refers might have been renamed or altered, possibly changing the structure of the table. Problems can occur as a result.

Adding an index to the table after preparing the statement can also invalidate the statement. A subsequent OPEN command for a cursor fails if the cursor refers to the invalid prepared statement; the failure occurs even if the OPEN command has the WITH REOPTIMIZATION clause.

If an index was added after the statement was prepared, you must prepare the statement again and declare the cursor again. You cannot simply reopen the cursor if it was based on a prepared statement that is no longer valid.

This error can also occur with SPL routines. Before the database server executes a new SPL routine the first time, it optimizes the code (statements) in the SPL routine. Optimization makes the code depend on the structure of the tables that the procedure references. If the table structure changes after the procedure is optimized, but before it is executed, this error can occur.

Each SPL routine is optimized the first time that it is run (not when it is created). This behavior means that an SPL routine might succeed the first time it is run but fail later under virtually identical circumstances. The failure of an SPL routine can also be intermittent, because failure during one execution forces an internal warning to reoptimize the procedure before the next execution.

The database server keeps a list of tables that the SPL routine references explicitly. Whenever any one of these explicitly referenced tables is modified, the database server reoptimizes the procedure the next time the procedure is executed.

However, if the SPL routine depends on a table that is referenced only indirectly, the database server cannot detect the need to reoptimize the procedure after that table is changed. For example, a table can be referenced indirectly if the SPL routine invokes a trigger. If a table that is referenced by the trigger (but not directly by the SPL routine) is changed, the database server does not know that it should reoptimize the SPL routine before running it. When the procedure is run after the table has been changed, this error can occur.

Use one of two methods to recover from this error:

- Issue the UPDATE STATISTICS FOR PROCEDURE statement to force reoptimization of the procedure.
- Rerun the procedure.

To prevent this error, you can force reoptimization of the SPL routine. For example, to force reoptimization of an SPL routine called **procedure_name**, execute the following statement:

```
UPDATE STATISTICS FOR PROCEDURE procedure_name;
```

Note that the following UPDATE STATISTICS statement has the same effect:

```
UPDATE STATISTICS FOR ROUTINE procedure_name;
```

Important:
Keep in mind that in databases that use transaction logging, you must run the UPDATE STATISTICS statement in a transaction that does not contain any other SQL statements.

You can add this statement to your program in either of the following ways:

- Place the UPDATE STATISTICS statement after each DDL statement that changes the mode of an object.
- Place the UPDATE STATISTICS statement before each execution of the SPL routine.

For efficiency, you can put the UPDATE STATISTICS statement with the action that occurs less frequently in the program (change of object mode or execution of the procedure). In most cases, the action that occurs less frequently in the program is the change of object mode.
When you follow this method of recovering from this error, you must execute the UPDATE STATISTICS FOR PROCEDURE statement for each procedure that references the changed tables indirectly, unless the procedure also references the tables explicitly.

You can also recover from this error by simply rerunning the SPL routine. The first time that the stored procedure fails, the database server marks the procedure as needing reoptimization. The next time that you run the procedure, the database server reoptimizes the procedure before running it. However, running the SPL routine twice might not be practical or safe. A safer choice is to use the UPDATE STATISTICS FOR PROCEDURE statement to force reoptimization of the procedure.

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Preparation for using directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Related tasks:
External optimizer directives

External optimizer directives

If you are user **informix**, you can create, save, and delete external directives.

- Creating and saving external directives
  You can define external directives by creating *association records* that include query optimizer directives, and saving those records in the **sysdirectives** system catalog table. Association records associate a list of one or more optimizer directives with a specific query text. The database server can apply those optimizer directives to subsequent instances of the same query text.
- Enabling external directives
  After you create and save external directives, you must set the configuration parameter and environmental variable that enable the directives. The database server searches for a directive for a query only if the external directives are set on both the database server and the client.
- Deleting external directives
  When you no longer need an external directive, the DBA or user **informix** can use the DELETE statement of SQL to remove it from the **sysdirectives** system catalog table.

Related concepts:
What optimizer directives are
Reasons to use optimizer directives
Preparation for using directives
Guidelines for using directives
Types of optimizer directives that are supported in SQL statements
Configuration parameters and environment variables for optimizer directives
Optimizer directives and SPL routines
Avoiding index or prepared object exceptions by forced reoptimization

Creating and saving external directives

You can define external directives by creating *association records* that include query optimizer directives, and saving those records in the **sysdirectives** system catalog table. Association records associate a list of one or more optimizer directives with a specific query text. The database server can apply those optimizer directives to subsequent instances of the same query text.

Use the SAVE EXTERNAL DIRECTIVES statement to create the association record to use for the list of one or more query directives These directives are applied automatically to subsequent instances of the same query.

The following example shows a SAVE EXTERNAL DIRECTIVES statement that registers an association-record in the system catalog as a new row in the **sysdirectives** table that can be used as a query optimizer directive.

```
SAVE EXTERNAL DIRECTIVES {+INDEX(t1,i11)} ACTIVE FOR
   SELECT {+INDEX(t1, i2) } c1 FROM t1 WHERE c1=1;
```

The following data is stored in the association record that the SQL statement above defined:

```
id              16
query           select {+INDEX(t1, i2) } c1 from t1 where c1=1
directive       INDEX(t1,i11)
directivecode   BYTE value

active          1
hashcode        -589336273
```

Here `{+INDEX(t1,i11)}`, the external directive that followed the DIRECTIVES keyword, will be applied to future instances of the specified query, but the inline `{+INDEX(t1,i2)}` directive will be ignored.

The information in the external directives that immediately follow the DIRECTIVES keyword must be within comment indicators, just as the same directives would appear in SELECT, UPDATE, MERGE, and DELETE statements, except that blank characters, rather than comma (,) symbols, are the required separators if the list of external directives includes more than one directive.

Enabling external directives

After you create and save external directives, you must set the configuration parameter and environmental variable that enable the directives. The database server searches for a directive for a query only if the external directives are set on both the database server and the client.

Enable the directive by using a combination of the EXT_DIRECTIVES configuration parameter, which is in the ONCONFIG file, and the **IFX_EXTDIRECTIVES** client-side environment variable.

The EXT_DIRECTIVE values that you can use are:

Value	Explanation
`0` (default)	Off. The directive cannot be enabled, even if IFX_EXTDIRECTIVES is enabled.
`1`	On. The directive can be enabled for a session if IFX_EXTDIRECTIVES is enabled.
`2`	On. The directive can be used even if IFX_EXTDIRECTIVES is not enabled.

You can also use the EXTDIRECTIVES option of the SET ENVIRONMENT statement to enable or disable external directives during a session. What you specify with the EXTDIRECTIVES option overwrites the external directive setting that is specified in the EXT_DIRECTIVES configuration parameter in the ONCONFIG file.

To overwrite the value for enabling or disabling the external directive in the ONCONFIG file:

- To enable the external directives during a session, specify `1`, `on`, or `ON` as the value for SET ENVIRONMENT EXTDIRECTIVES.
- To disable the external directives during a session, specify `0`, `off`, or `OFF` as the value for SET ENVIRONMENT EXTDIRECTIVES.

To enable the default values specified in the EXT_DIRECTIVES configuration parameter and in the client-side IFX_EXTDIRECTIVES environment variable during a session, specify DEFAULT as the value for the EXTDIRECTIVES option of the SET ENVIRONMENT statement.

The explain output file specifies whether external directives are in effect.

Related concepts:
The explain output file
Query statistics section provides performance debugging information
Report that shows the query plan chosen by the optimizer

Related information:
SET EXPLAIN statement
Using the FILE TO option
Default name and location of the explain output file on UNIX
Default name and location of the output file on Windows
onmode -Y: Dynamically change SET EXPLAIN
onmode and Y arguments: Change query plan measurements for a session (SQL administration API)

Deleting external directives

When you no longer need an external directive, the DBA or user **informix** can use the DELETE statement of SQL to remove it from the **sysdirectives** system catalog table.

When external directives are enabled and the **sysdirectives** system catalog table is not empty,

- the database server compares every query with the query text of every ACTIVE external directive,
- and for queries executed by the DBA (or by user **informix**) with every TEST ONLY external directive.

The purpose of external directives is to improve the performance of queries that match the query string, but the use of such directives can potentially slow other queries, if the query execution optimizer must compare the query strings of a large number of active external directives with the text of every SELECT statement. For this reason, recommends that the DBA not allow the **sysdirectives** table to accumulate more than a few ACTIVE rows. (An alternative way to avoid unintended performance impact on other queries is to disable support for external directives by setting the EXT_DIRECTIVES configuration parameter to `0`. Setting the **IFX_EXTDIRECTIVES** client environment variable to `0` has the same effect.)

Parallel database query (PDQ)

You can manage how the database server performs PDQ and you can monitor the resources that the database server uses for PDQ.

- What PDQ is
  Parallel database query (PDQ) is a database server feature that can improve performance dramatically when the server processes queries that decision-support applications initiate. PDQ enables Informix to distribute the work for one aspect of a query among several processors. For example, if a query requires an aggregation, Informix can distribute the work for the aggregation among several processors.
- Structure of a PDQ query
  Each decision-support query has a primary thread. The database server can start additional threads to perform tasks for the query (for example, scans and sorts). Depending on the number of tables or fragments that a query must search and the resources that are available for a decision-support query, the database server assigns different components of a query to different threads.
- Database server operations that use PDQ
  Informix processes some types of SQL operations that the database server processes in parallel. However some situations limit the degree of parallelism that Informix can use.
- The Memory Grant Manager
  The Memory Grant Manager (MGM) is a database server component that coordinates the use of memory, CPU virtual processors (VPs), disk I/O, and scan threads among decision-support queries. The MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and MAX_PDQPRIORITY configuration parameters to determine the quantity of these PDQ resources that can be granted to a decision-support query.
- The allocation of resources for parallel database queries
  When you configure the database server, consider how the use of PDQ affects users of OLTP, decision-support (DSS) applications, and other applications. Then you can plan how to allocate resources for PDQ.
- Managing PDQ queries
  The database server administrator, the writer of an application, and the users all have a certain amount of control over the amount of resources that Informix allocates to processing a query. The database server administrator exerts control through the use of configuration parameters. The application developer or the user can exert control through an environment variable or SQL statement.
- Monitoring resources used for PDQ and DSS queries
  You can monitor the resources (shared memory and threads) that the Memory Grant Manager (MGM) has allocated for PDQ queries and the resources that PDQ and decision-support (DSS) queries currently use.

What PDQ is

Parallel database query (PDQ) is a database server feature that can improve performance dramatically when the server processes queries that decision-support applications initiate. PDQ enables Informix® to distribute the work for one aspect of a query among several processors. For example, if a query requires an aggregation, Informix can distribute the work for the aggregation among several processors.

PDQ also includes tools for resource management.

Another database server feature, *table fragmentation*, allows you to store the parts of a table on different disks. PDQ delivers maximum performance benefits when the data that you query is in fragmented tables. For information about how to use fragmentation for maximum performance, see Planning a fragmentation strategy.

Related concepts:
Database server operations that use PDQ
The allocation of resources for parallel database queries

Managing PDQ queries
Monitoring resources used for PDQ and DSS queries

Structure of a PDQ query

Each decision-support query has a primary thread. The database server can start additional threads to perform tasks for the query (for example, scans and sorts). Depending on the number of tables or fragments that a query must search and the resources that are available for a decision-support query, the database server assigns different components of a query to different threads.

The database server initiates these PDQ threads, which are listed as *secondary threads* in the SET EXPLAIN output.

Secondary threads are further classified as either *producers* or *consumers*, depending on their function. A producer thread supplies data to another thread. For example, a scan thread might read data from shared memory that corresponds to a given table and pass it along to a join thread. In this case, the scan thread is considered a producer, and the join thread is considered a consumer. The join thread, in turn, might pass data along to a sort thread. When doing so, the join thread is considered a producer, and the sort thread is considered a consumer.

Several producers can supply data to a single consumer. When this situation occurs, the database server sets up an internal mechanism, called an *exchange*, that synchronizes the transfer of data from those producers to the consumer. For instance, if a fragmented table is to be sorted, the optimizer typically calls for a separate scan thread for each fragment. Because of different I/O characteristics, the scan threads can be expected to complete at different times. An exchange is used to funnel the data produced by the various scan threads into one or more sort threads with a minimum of buffering. Depending on the complexity of the query, the optimizer might call for a multilayered hierarchy of producers, exchanges, and consumers. Generally speaking, consumer threads work in parallel with producer threads so that the amount of intermediate buffering that the exchanges perform remains negligible.

The database server creates these threads and exchanges automatically and transparently. They terminate automatically as they complete processing for a given query. The database server creates new threads and exchanges as needed for subsequent queries.

Database server operations that use PDQ

Informix® processes some types of SQL operations that the database server processes in parallel. However some situations limit the degree of parallelism that Informix can use.

In the topics on database server operations that use PDQ in this section, a *query* is any SELECT statement.

- Parallel update and delete operations
  Informix performs some types of update and delete operations in parallel.
- Parallel insert operations
  Informix performs some types of insert operations in parallel.
- Parallel index builds
  Index builds can take advantage of PDQ and can be parallelized. The database server performs both scans and sorts in parallel for index builds.
- Parallel user-defined routines
  If a query contains a user-defined routine (UDR) in an expression, the database server can execute a query in parallel when you turn on PDQ.
- Hold cursors that use PDQ
  When hold cursors that are created by declaring the WITH HOLD qualifier have no locks, PDQ is enabled.
- SQL operations that do not use PDQ
  The database server does not process some types of queries in parallel.
- Update statistics operations affected by PDQ
  An SQL UPDATE STATISTICS statement that is not processed in parallel, is affected by PDQ because it must allocate the memory used for sorting. Thus the behavior of the UPDATE STATISTICS statement is affected by the memory management associated with PDQ.
- SPL routines and triggers and PDQ
  Statements that involve SPL routines are not executed in parallel. However, statements within procedures are executed in parallel.
- Correlated and uncorrelated subqueries
  The database server does not use PDQ to process correlated subqueries. Only one thread at a time can execute a correlated subquery. While one thread executes a correlated subquery, other threads that request to execute the subquery are blocked until the first one completes.
- OUTER index joins and PDQ
  The database server reduces the PDQ priority of queries that contain OUTER index joins to LOW (if the priority is set to a higher value) for the duration of the query. If a subquery or a view contains OUTER index joins, the database server lowers the PDQ priority of only that subquery or view, not of the parent query or any other subquery.
- Remote tables used with PDQ
  Although the database server can process the data stored in a remote table in parallel, the data is communicated serially because the database server allocates a single thread to submit and receive the data from the remote table. The database server lowers the PDQ priority of queries that require access to a remote database to LOW.

Related concepts:
What PDQ is

Parallel update and delete operations

Informix® performs some types of update and delete operations in parallel.

The database server takes the following two steps to process UPDATE and DELETE statements:

1. Fetches the qualifying rows.
2. Applies the action of updating or deleting.

The database server performs the first step of an UPDATE or DELETE statement in parallel, with the following exceptions:

- The targeted table in a DELETE statement has a referential constraint that can cascade to a child table.
- The UPDATE or DELETE statement contains an OR clause and the optimizer chooses an OR index to process the OR filter.
- The UPDATE statement contains a subquery that the optimizer converts into a join.

Parallel insert operations

Informix® performs some types of insert operations in parallel.

The types of insert operations that the server performs in parallel are:

- SELECT...INTO TEMP inserts using explicit temporary tables.
- INSERT INTO...SELECT inserts using implicit temporary tables.

- Explicit inserts with SELECT...INTO TEMP statements
  The database server can insert rows in parallel into explicit temporary tables that you specify in SQL statements of the form SELECT....INTO TEMP.
- Implicit inserts with INSERT INTO...SELECT statements
  The database server can also insert rows in parallel into implicit tables that it creates when it processes SQL statements of the form INSERT INTO...SELECT.

Explicit inserts with SELECT...INTO TEMP statements

The database server can insert rows in parallel into explicit temporary tables that you specify in SQL statements of the form SELECT....INTO TEMP.

For example, the database server can perform the inserts in parallel into the temporary table, **temp_table**, as the following example shows:

```
SELECT * FROM table1 INTO TEMP temp_table
```

To perform parallel inserts into a temporary table:

1. Set PDQ priority > 0.
   You must meet this requirement for any query that you want the database server to perform in parallel.

2. Set DBSPACETEMP to a list of two or more dbspaces.
   This step is required because of the way that the database server performs the insert. To perform the insert in parallel, the database server first creates a fragmented temporary table. So that the database server knows where to store the fragments of the temporary table, you must specify a list of two or more dbspaces in the DBSPACETEMP configuration parameter or the **DBSPACETEMP** environment variable. In addition, you must set DBSPACETEMP to indicate storage space for the fragments before you execute the SELECT...INTO statement.

The database server performs the parallel insert by writing in parallel to each of the fragments in a round-robin fashion. Performance improves as you increase the number of fragments.

Related concepts:
Implicit inserts with INSERT INTO...SELECT statements

Implicit inserts with INSERT INTO...SELECT statements

The database server can also insert rows in parallel into implicit tables that it creates when it processes SQL statements of the form INSERT INTO...SELECT.

For example, the database server processes the following INSERT statement in parallel:

```
INSERT INTO target_table SELECT * FROM source_table
```

The target table can be either a permanent table or a temporary table.

The database server processes this type of INSERT statement in parallel only when the target tables meet the following criteria:

- The value of PDQ priority is greater than 0.
- The target table is fragmented into two or more dbspaces.
- The target table has no enabled referential constraints or triggers.
- The target table is not a remote table.
- In a database with logging, the target table does not contain filtering constraints.
- The target table does not contain columns of TEXT or BYTE data type.

The database server does not process parallel inserts that reference an SPL routine. For example, the database server never processes the following statement in parallel:

```
INSERT INTO table1 EXECUTE PROCEDURE ins_proc
```

Related tasks:
Explicit inserts with SELECT...INTO TEMP statements

Parallel index builds

Index builds can take advantage of PDQ and can be parallelized. The database server performs both scans and sorts in parallel for index builds.

The following operations initiate index builds:

- Create an index.
- Add a unique, primary key.
- Add a referential constraint.
- Enable a referential constraint.

When PDQ is in effect, the scans for index builds are controlled by the PDQ configuration parameters described in The allocation of resources for parallel database queries.

If you have a computer with multiple CPUs, the database server uses two sort threads to sort the index keys. The database server uses two sort threads during index builds without the user setting the **PSORT_NPROCS** environment variable.

Parallel user-defined routines

If a query contains a user-defined routine (UDR) in an expression, the database server can execute a query in parallel when you turn on PDQ.

The database server can perform the following parallel operations if the UDR is written and registered appropriately:

- Parallel scans
- Parallel comparisons with the UDR

For more information about how to enable parallel execution of UDRs, see Parallel UDRs.

Hold cursors that use PDQ

When hold cursors that are created by declaring the WITH HOLD qualifier have no locks, PDQ is enabled.

PDQ will be set for hold cursors in the following cases:

- Queries with Dirty Read or Committed Read isolation level, ANSI, and read-only cursor
- Queries with Dirty Read or Committed Read isolation level, NON-ANSI, non-updateable cursor

SQL operations that do not use PDQ

The database server does not process some types of queries in parallel.

For example, the server does not process the following types of queries in parallel:

- Queries started with an isolation level of Cursor Stability
  Subsequent changes to the isolation level do not affect the parallelism of queries already prepared. This situation results from the inherent nature of parallel scans, which scan several rows simultaneously.
- Queries that use a cursor declared as FOR UPDATE
- Queries in the `FOR EACH ROW` section of the Action clause of a *Select* trigger
- A DELETE or MERGE statement in the `FOR EACH ROW` section of the Action clause of a *Delete* trigger
- An INSERT or MERGE statement in the `FOR EACH ROW` section of the Action clause of an *Insert* trigger
- An UPDATE or MERGE statement in the `FOR EACH ROW` section of the Action clause of an *Update* trigger
- Data definition language (DDL) statements.
  For a complete list of the DDL statements of SQL that Informix® supports, see the *IBM® Informix Guide to SQL: Syntax*.

In addition, the database server does not process sequence objects in PDQ operations. If your SQL statement includes sequencing operations, such as expressions that include the **NEXTVAL** or **CURRVAL** operators, PDQ processing is unavailable for that statement.

Update statistics operations affected by PDQ

An SQL UPDATE STATISTICS statement that is not processed in parallel, is affected by PDQ because it must allocate the memory used for sorting. Thus the behavior of the UPDATE STATISTICS statement is affected by the memory management associated with PDQ.

The database server must allocate the memory that the UPDATE STATISTICS statement uses for sorting.

If you have an extremely large database and indexes are fragmented, UPDATE STATISTICS LOW can automatically run statements in parallel. For more information, see Update statistics in parallel on very large databases.

SPL routines and triggers and PDQ

Statements that involve SPL routines are not executed in parallel. However, statements within procedures are executed in parallel.

When the database server executes an SPL routine, it does not use PDQ to process non-related SQL statements contained in the procedure. Each SQL statement can be executed independently in parallel, however, using intraquery parallelism when possible. As a consequence, you should limit the use of procedure calls from within data manipulation language (DML) statements if you want to use the parallel-processing abilities of the database server. For a complete list of DML statements, see the *IBM® Informix Guide to SQL: Syntax*.

The database server uses intraquery parallelism to process the statements in the body of an SQL trigger in the same way that it processes statements in SPL routines. For restrictions on using PDQ for queries in some triggered actions of Select, Insert, and Update triggers, see SQL operations that do not use PDQ.

Correlated and uncorrelated subqueries

The database server does not use PDQ to process correlated subqueries. Only one thread at a time can execute a correlated subquery. While one thread executes a correlated subquery, other threads that request to execute the subquery are blocked until the first one completes.

For uncorrelated subqueries, only the first thread that makes the request actually executes the subquery. Other threads simply use the results of the subquery and can do so in parallel.

As a consequence, it is strongly recommended that, whenever possible, you use joins rather than subqueries to build queries so that the queries can take advantage of PDQ.

OUTER index joins and PDQ

The database server reduces the PDQ priority of queries that contain OUTER index joins to LOW (if the priority is set to a higher value) for the duration of the query. If a subquery or a view contains OUTER index joins, the database server lowers the PDQ priority of only that subquery or view, not of the parent query or any other subquery.

Remote tables used with PDQ

Although the database server can process the data stored in a remote table in parallel, the data is communicated serially because the database server allocates a single thread to submit and receive the data from the remote table. The database server lowers the PDQ priority of queries that require access to a remote database to LOW.

In this case, all local scans are parallel, but all local joins and remote access are nonparallel.

The Memory Grant Manager

The Memory Grant Manager (MGM) is a database server component that coordinates the use of memory, CPU virtual processors (VPs), disk I/O, and scan threads among decision-support queries. The MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and MAX_PDQPRIORITY configuration parameters to determine the quantity of these PDQ resources that can be granted to a decision-support query.

The MGM dynamically allocates the following resources for decision-support queries:

- The number of scan threads that are started for each decision-support query
- The number of threads that can be started for each query
- The amount of memory in the virtual portion of database server shared memory that the query can reserve

When your database server system has heavy OLTP use, and you find performance is degrading, you can use the MGM facilities to limit the resources that are committed to decision-support queries. During off-peak hours, you can designate a larger proportion of the resources to parallel processing, which achieves higher throughput for decision-support queries.

The MGM grants memory to a query for such activities as sorts, hash joins, and processing of GROUP BY clauses. The amount of memory that decision-support queries use cannot exceed DS_TOTAL_MEMORY.

The MGM grants memory to queries in *quantum* increments. To calculate the approximate size of the quantum, use the following formula:

```
memory quantum = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

For example, if DS_TOTAL_MEMORY is 12 MB and DS_MAX_QUERIES is 4, the quantum is 3 MB (12/4). Thus, with these values in effect, a quantum of memory equals 3 MB. The database server can adjust the size of the quantum dynamically when it grants memory. In general, memory is allocated more efficiently when quanta are smaller. You can often improve performance of concurrent queries by increasing DS_MAX_QUERIES to reduce the size of a quantum of memory.

To monitor resources that the MGM allocates, run the **onstat -g mgm** command. This command shows only the amount of memory that is used; it does not show the amount of memory that is granted.

The MGM also grants a maximum number of scan threads per query that is based on the values of the DS_MAX_SCANS and the DS_MAX_QUERIES parameters.

The following formula yields the maximum number of scan threads per query:

```
scan_threads = min (nfrags, DS_MAX_SCANS * (pdqpriority / 100)
   * (MAX_PDQPRIORITY / 100))
```

nfrags
: Is the number of fragments in the table with the largest number of fragments.

pdqpriority
: Is the value for PDQ priority that is set by either the PDQPRIORITY environment variable or the SQL statement SET PDQPRIORITY.

The PDQPRIORITY environment variable and the SQL statement SET PDQPRIORITY request a percentage of PDQ resources for a query. You can use the MAX_PDQPRIORITY configuration parameter to limit the percentage of the requested resources that a query can obtain and to limit the impact of decision-support queries on OLTP processing.

Related concepts:
Effect of configuration on memory utilization
Limiting the priority of decision-support queries
The DS_TOTAL_MEMORY configuration parameter and memory utilization
Related information:
onstat -g mgm command: Print MGM resource information

The allocation of resources for parallel database queries

When you configure the database server, consider how the use of PDQ affects users of OLTP, decision-support (DSS) applications, and other applications. Then you can plan how to allocate resources for PDQ.

When the database server uses PDQ to perform a query in parallel, it puts a heavy load on the operating system. In particular, PDQ exploits the following resources:

- Memory
- CPU VPs
- Disk I/O (to fragmented tables and temporary table space)
- Scan threads

You can control how the database server uses resources in the following ways:

- Limit the priority of parallel database queries.
- Adjust the amount of memory.
- Limit the number of scan threads.
- Limit the number of concurrent queries.

- Limiting the priority of decision-support queries
  You can limit the parallel processing resources that decision-support (DSS) queries consume by adjusting the values of **PDQPRIORITY** environment variable, the MAX_PDQPRIORITY configuration parameter, and other configuration parameters.
- Adjusting the amount of memory for DSS and PDQ queries
  You can estimate the amount of shared memory to allocate to decision-support (DSS) queries. Then, if necessary, you can adjust the value of the DS_TOTAL_MEMORY configuration parameter, which specifies the amount of memory available for PDQ queries.
- Limiting the number of concurrent scans
  The database server apportions some number of scans to a query according to its PDQ priority (among other factors). You can adjust the value of the DS_MAX_SCANS configuration parameter to limit the number of concurrent scans.
- Limiting the maximum number of PDQ queries
  You can adjust the maximum number of PDQ queries that can run concurrently by changing the value of the DS_MAX_QUERIES configuration parameter.

Related concepts:
What PDQ is

Limiting the priority of decision-support queries

You can limit the parallel processing resources that decision-support (DSS) queries consume by adjusting the values of **PDQPRIORITY** environment variable, the MAX_PDQPRIORITY configuration parameter, and other configuration parameters.

The default value for the PDQ priority of individual applications is `0`, which means that PDQ processing is not used. The database server uses this value unless one of the following actions overrides it:

- You set the **PDQPRIORITY** environment variable.
- The application uses the SET PDQPRIORITY statement.

The **PDQPRIORITY** environment variable and the MAX_PDQPRIORITY configuration parameter work together to control the amount of resources to allocate for parallel processing. Setting these correctly is critical for the effective operation of PDQ.

The MAX_PDQPRIORITY configuration parameter allows you to limit the parallel processing resources that DSS queries consume. Thus, the **PDQPRIORITY** environment variable sets a *reasonable* or *recommended* priority value, and MAX_PDQPRIORITY limits the resources that an application can claim.

The MAX_PDQPRIORITY configuration parameter specifies the maximum percentage of the requested resources that a query can obtain. For instance, if **PDQPRIORITY** is `80` and MAX_PDQPRIORITY is `50`, each active query receives an amount of memory equal to 40 percent of DS_TOTAL_MEMORY, rounded down to the nearest quantum. In this example, MAX_PDQPRIORITY effectively limits the number of concurrent decision-support queries to two. Subsequent queries must wait for earlier queries to finish before they acquire the resources that they need to run.

An application or user can use the DEFAULT tag of the SET PDQPRIORITY statement to use the value for PDQ priority if the value has been set by the **PDQPRIORITY** environment variable. DEFAULT is the symbolic equivalent of a `-1` value for PDQ priority.

You can use the **onmode** command-line utility to change the values of the following configuration parameters temporarily:

- Use **onmode -M** to change the value of DS_TOTAL_MEMORY.
- Use **onmode -Q** to change the value of DS_MAX_QUERIES.
- Use **onmode -D** to change the value of MAX_PDQPRIORITY.
- Use **onmode -S** to change the value of DS_MAX_SCANS.

These changes remain in effect only as long as the database server remains up and running. When the database server starts, it uses the values listed in the ONCONFIG file.

For more information about the preceding parameters, see Effect of configuration on memory utilization. For more information about **onmode**, see your *IBM® Informix Administrator's Reference*.

If you must change the values of the decision-support parameters on a regular basis (for example, to set MAX_PDQPRIORITY to `100` each night for processing reports), you can use a scheduled operating-system job to set the values. For information about creating scheduled jobs, see your operating-system manuals.

To obtain the best performance from the database server, choose values for the **PDQPRIORITY** environment variable and MAX_PDQPRIORITY parameter, observe the resulting behavior, and then adjust the values for these parameters. No well-defined rules exist for choosing these environment variable and parameter values. The following sections discuss strategies for setting **PDQPRIORITY** and MAX_PDQPRIORITY for specific needs.

- Limiting the value of the PDQ priority
  You can adjust the value of the MAX_PDQPRIORITY configuration parameter to adjust the PDQ priority and allocate more resources to either OLTP or decision-support processing.
- Maximizing OLTP throughput for queries
  At times, you might want to allocate resources to maximize the throughput for individual OLTP queries rather than for decision-support queries.
- Conserving resources when using PDQ
  If applications make little use of queries that require parallel sorts and parallel joins, consider using the `LOW` setting for PDQ priority.
- Allowing maximum use of parallel processing
  Set **PDQPRIORITY** and MAX_PDQPRIORITY to `100` if you want the database server to assign as many resources as possible to parallel processing.
- Determining the level of parallel processing
  You can use different numeric settings for **PDQPRIORITY** to experiment with the effects of parallelism on a single application.
- Limits on parallel operations associated with PDQ priority
  The database server reduces the PDQ priority of queries that contain outer joins to `LOW` (if set to a higher value) for the duration of the query. If a subquery or a view contains outer joins, the database server lowers the PDQ priority only of that subquery or view, not of the parent query or of any other subquery.
- Using SPL routines with PDQ queries
  The database server freezes the PDQ priority that is used to optimize SQL statements within SPL routines at the time of procedure creation or the last manual recompilation with the UPDATE STATISTICS statement. You can change the client value of **PDQPRIORITY**.

Related concepts:
The Memory Grant Manager

Limiting the value of the PDQ priority

You can adjust the value of the MAX_PDQPRIORITY configuration parameter to adjust the PDQ priority and allocate more resources to either OLTP or decision-support processing.

The MAX_PDQPRIORITY configuration parameter limits the PDQ priority that the database server grants when users either set the **PDQPRIORITY** environment variable or issue the SET PDQPRIORITY statement before they issue a query. When an application or an end user attempts to set a PDQ priority, the priority that is granted is multiplied by the value that MAX_PDQPRIORITY specifies.

Set the value of MAX_PDQPRIORITY lower when you want to allocate more resources to OLTP processing.

Set the value of MAX_PDQPRIORITY higher when you want to allocate more resources to decision-support processing.

The possible range of values is 0 to 100. This range represents the percent of resources that you can allocate to decision-support processing.

Related concepts:
Maximizing OLTP throughput for queries
Conserving resources when using PDQ
Allowing maximum use of parallel processing
Determining the level of parallel processing
Limits on parallel operations associated with PDQ priority
Using SPL routines with PDQ queries

Maximizing OLTP throughput for queries

At times, you might want to allocate resources to maximize the throughput for individual OLTP queries rather than for decision-support queries.

In this case, set MAX_ PDQPRIORITY to `0`, which limits the value of PDQ priority to `OFF`. A PDQ priority value of `OFF` does not prevent decision-support queries from running. Instead, it causes the queries to run without parallelization. In this configuration, response times for decision-support queries might be slow.

Related concepts:
Limiting the value of the PDQ priority
Conserving resources when using PDQ
Allowing maximum use of parallel processing
Determining the level of parallel processing
Limits on parallel operations associated with PDQ priority
Using SPL routines with PDQ queries

Conserving resources when using PDQ

If applications make little use of queries that require parallel sorts and parallel joins, consider using the `LOW` setting for PDQ priority.

If the database server is operating in a multiuser environment, you might set MAX_PDQPRIORITY to `1` to increase interquery performance at the cost of some intraquery parallelism. A trade-off exists between these two different types of parallelism because they compete for the same resources. As a compromise, you might set MAX_PDQPRIORITY to some intermediate value (perhaps `20` or `30`) and set **PDQPRIORITY** to `LOW`. The environment variable sets the default behavior to `LOW`, but the MAX_PDQPRIORITY configuration parameter allows individual applications to request more resources with the SET PDQPRIORITY statement.

Related concepts:
Limiting the value of the PDQ priority
Maximizing OLTP throughput for queries
Allowing maximum use of parallel processing
Determining the level of parallel processing
Limits on parallel operations associated with PDQ priority
Using SPL routines with PDQ queries

Allowing maximum use of parallel processing

Set **PDQPRIORITY** and MAX_PDQPRIORITY to `100` if you want the database server to assign as many resources as possible to parallel processing.

This setting is appropriate for times when parallel processing does not interfere with OLTP processing.

Related concepts:
Limiting the value of the PDQ priority
Maximizing OLTP throughput for queries
Conserving resources when using PDQ
Determining the level of parallel processing
Limits on parallel operations associated with PDQ priority
Using SPL routines with PDQ queries

Determining the level of parallel processing

You can use different numeric settings for **PDQPRIORITY** to experiment with the effects of parallelism on a single application.

For information about how to monitor parallel execution, see Monitoring resources used for PDQ and DSS queries.

Related concepts:
Limiting the value of the PDQ priority
Maximizing OLTP throughput for queries
Conserving resources when using PDQ
Allowing maximum use of parallel processing
Limits on parallel operations associated with PDQ priority
Using SPL routines with PDQ queries

Limits on parallel operations associated with PDQ priority

The database server reduces the PDQ priority of queries that contain outer joins to `LOW` (if set to a higher value) for the duration of the query. If a subquery or a view contains outer joins, the database server lowers the PDQ priority only of that subquery or view, not of the parent query or of any other subquery.

The database server lowers the PDQ priority of queries that require access to a remote database (same or different database server instance) to `LOW` if you set it to a higher value. In that case, all local scans are parallel, but all local joins and remote accesses are nonparallel.

Related concepts:
Limiting the value of the PDQ priority
Maximizing OLTP throughput for queries
Conserving resources when using PDQ
Allowing maximum use of parallel processing
Determining the level of parallel processing
Using SPL routines with PDQ queries

Using SPL routines with PDQ queries

The database server freezes the PDQ priority that is used to optimize SQL statements within SPL routines at the time of procedure creation or the last manual recompilation with the UPDATE STATISTICS statement. You can change the client value of **PDQPRIORITY**.

To change the client value of **PDQPRIORITY**, embed the SET PDQPRIORITY statement within the body of your SPL routine.

The PDQ priority value that the database server uses to optimize or reoptimize an SQL statement is the value that was set by a SET PDQPRIORITY statement, which must have been executed within the same procedure. If no such statement has been executed, the value that was in effect when the procedure was last compiled or created is used.

The PDQ priority value currently in effect outside a procedure is ignored within a procedure when it is executing.

It is suggested that you turn PDQ priority off when you enter a procedure and then turn it on again for specific statements. You can avoid tying up large amounts of memory for the procedure, and you can make sure that the crucial parts of the procedure use the appropriate PDQ priority, as the following example illustrates:

```
CREATE PROCEDURE my_proc (a INT, b INT, c INT)
   Returning INT, INT, INT;
SET PDQPRIORITY 0;
...
SET PDQPRIORITY 85;
SELECT ... (big complicated SELECT statement)
SET PDQPRIORITY 0;
...
;
```

Related concepts:
Limiting the value of the PDQ priority
Maximizing OLTP throughput for queries
Conserving resources when using PDQ
Allowing maximum use of parallel processing
Determining the level of parallel processing
Limits on parallel operations associated with PDQ priority

Adjusting the amount of memory for DSS and PDQ queries

You can estimate the amount of shared memory to allocate to decision-support (DSS) queries. Then, if necessary, you can adjust the value of the DS_TOTAL_MEMORY configuration parameter, which specifies the amount of memory available for PDQ queries.

Use the following formula as a starting point for estimating the amount of shared memory to allocate to DSS queries:

```
DS_TOTAL_MEMORY = p_mem - os_mem - rsdnt_mem -
                  (128 kilobytes * users) - other_mem
```

p_mem
represents the total physical memory that is available on the host computer.
os_mem

represents the size of the operating system, including the buffer cache.

resdnt_mem
represents the size of Informix® resident shared memory.

users
is the number of expected users (connections) specified in the NETTYPE configuration parameter.

other_mem
is the size of memory used for other (non-IBM® Informix) applications.

The value for DS_TOTAL_MEMORY that is derived from this formula serves only as a starting point. To arrive at a value that makes sense for your configuration, you must monitor paging and swapping. (Use the tools provided with your operating system to monitor paging and swapping.) When paging increases, decrease the value of DS_TOTAL_MEMORY so that processing the OLTP workload can proceed.

The amount of memory that is granted to a single parallel database query depends on many system factors, but in general, the amount of memory granted to a single parallel database query is proportional to the following formula:

```
memory_grant_basis = (DS_TOTAL_MEMORY/DS_MAX_QUERIES) *
                (PDQPRIORITY / 100) *
                (MAX_PDQPRIORITY / 100)
```

However, if the currently executing queries on all databases of the server instance require more memory than this estimate of the average allocation, another query might overflow to disk or might wait until concurrent queries completed execution and released sufficient memory resources for running the query. The following alternative formula estimates the PDQ memory available for a single query directly:

```
memory_for_single_query = DS_TOTAL_MEMORY  *
                (PDQPRIOIRTY / 100) *
                (MAX_PDQPRIORITY / 100)
```


Limiting the number of concurrent scans

The database server apportions some number of scans to a query according to its PDQ priority (among other factors). You can adjust the value of the DS_MAX_SCANS configuration parameter to limit the number of concurrent scans.

The DS_MAX_SCANS and MAX_PDQPRIORITY configuration parameters allow you to limit the resources that users can assign to a query, according to the following formula:

```
scan_threads = min (nfrags, (DS_MAX_SCANS * (pdqpriority / 100)
   * (MAX_PDQPRIORITY / 100) )
```

nfrags
is the number of fragments in the table with the largest number of fragments.

pdqpriority
is the PDQ priority value set by either the **PDQPRIORITY** environment variable or the SET PDQPRIORITY statement.

For example, suppose a large table contains 100 fragments. With no limit on the number of concurrent scans allowed, the database server would concurrently execute 100 scan threads to read this table. In addition, many users could initiate this query.

As the database server administrator, you set the DS_MAX_SCANS configuration parameter to a value lower than the number of fragments in this table to prevent the database server from being flooded with scan threads by multiple decision-support queries. You can set DS_MAX_SCANS to 20 to ensure that the database server concurrently executes a maximum of 20 scan threads for parallel scans. Furthermore, if multiple users initiate parallel database queries, each query receives only a percentage of the 20 scan threads, according to the PDQ priority assigned to the query and the value for the MAX_PDQPRIORITY configuration parameter that the database server administrator sets.

Limiting the maximum number of PDQ queries

You can adjust the maximum number of PDQ queries that can run concurrently by changing the value of the DS_MAX_QUERIES configuration parameter.

The DS_MAX_QUERIES configuration parameter limits the number of concurrent decision-support queries that can run.

To estimate the number of decision-support (DSS) queries that the database server can run concurrently, count each query that runs with PDQ priority set to 1 or greater as one full query.

The database server allocates less memory to queries that run with a lower priority, so you can assign lower-priority queries a PDQ priority value that is between 1 and 30, depending on the resource impact of the query. The total number of queries with PDQ priority values greater than 0 cannot exceed the value of DS_MAX_QUERIES.

Managing PDQ queries

The database server administrator, the writer of an application, and the users all have a certain amount of control over the amount of resources that Informix® allocates to processing a query. The database server administrator exerts control through the use of configuration parameters. The application developer or the user can exert control through an environment variable or SQL statement.

- Analyzing query plans with SET EXPLAIN output
  You can use SET EXPLAIN output to study the query plans of an application. The output of the SET EXPLAIN statement shows decisions that the query optimizer makes. It shows whether parallel scans are used, the maximum number of threads required to answer the query, and the type of join used for the query.
- Influencing the choice of a query plan
  The **OPTCOMPIND** environment variable and the OPTCOMPIND configuration parameter indicate the preferred join plan, thus assisting the optimizer in selecting the appropriate join method for parallel database queries. To influence the optimizer in its choice of a join plan, you can set the OPTCOMPIND configuration parameter.
- Setting the PDQ priority dynamically
  You can use the SET PDQPRIORITY statement to set PDQ priority dynamically within an application. The PDQ priority value can be any integer from -1 through 100.
- Enabling the database server to allocate PDQ memory
  You can set the IMPLICIT_PDQ session environment option of the SET ENVIRONMENT statement to enable the database server to calculate the amount of PDQ memory to allocate to queries during the current session. This potentially overrides the current **PDQPRIORITY** setting.
- User control of PDQ resources
  To indicate the PDQ priority of a query, you can set the **PDQPRIORITY** environment variable or execute the SET PDQPRIORITY statement prior to issuing a query. These PDQ priority options allow you to request a certain amount of parallel-processing resources for the query.
- DBA control of resources for PDQ and DSS queries
  To manage the total amount of resources that the database server allocates to parallel database and decision-support (DSS) queries, the database server administrator can set the environment variable and configuration parameters.

Related concepts:
What PDQ is

Analyzing query plans with SET EXPLAIN output

You can use SET EXPLAIN output to study the query plans of an application. The output of the SET EXPLAIN statement shows decisions that the query optimizer makes. It shows whether parallel scans are used, the maximum number of threads required to answer the query, and the type of join used for the query.

You can restructure a query or use OPTCOMPIND to change how the optimizer treats the query.

Influencing the choice of a query plan

The **OPTCOMPIND** environment variable and the OPTCOMPIND configuration parameter indicate the preferred join plan, thus assisting the optimizer in selecting the appropriate join method for parallel database queries. To influence the optimizer in its choice of a join plan, you can set the OPTCOMPIND configuration parameter.

The value that you assign to the OPTCOMPIND configuration parameter is referenced only when applications do not set the **OPTCOMPIND** environment variable.

Set OPTCOMPIND to `0` if you want the database server to select a join plan exactly as it did in versions of the database server prior to version 6.0. This option ensures compatibility with previous versions of the database server.

An application with an isolation mode of Repeatable Read can lock all records in a table when it performs a hash join. For this reason, you should set OPTCOMPIND to `1`.

If you want the optimizer to make the determination for you based on cost, regardless of the isolation level of applications, set OPTCOMPIND to `2`.

You can use the SET ENVIRONMENT OPTCOMPIND command to change the value of OPTCOMPIND within a session. For more information about using this command, see Setting the value of OPTCOMPIND within a session.

For more information about OPTCOMPIND and the different join plans, see The query plan.

Setting the PDQ priority dynamically

You can use the SET PDQPRIORITY statement to set PDQ priority dynamically within an application. The PDQ priority value can be any integer from -1 through 100.

The PDQ priority set with the SET PDQPRIORITY statement supersedes the **PDQPRIORITY** environment variable.

The DEFAULT tag for the SET PDQPRIORITY statement allows an application to revert to the value for PDQ priority as set by the environment variable, if any. For more information about the SET PDQPRIORITY statement, see the *IBM® Informix Guide to SQL: Syntax*.

Enabling the database server to allocate PDQ memory

You can set the IMPLICIT_PDQ session environment option of the SET ENVIRONMENT statement to enable the database server to calculate the amount of PDQ memory to allocate to queries during the current session. This potentially overrides the current **PDQPRIORITY** setting.

The maximum amount of memory that the database server can allocate, however, is limited by the physical memory available to your system, and by the settings of these parameters:

- The **PDQPRIORITY** environment variable
- The most recent SET PDQPRIORITY statement of SQL
- The MAX_PDQPRIORITY configuration parameter
- The DS_TOTAL_MEMORY configuration parameter
- The BOUND_IMPL_PDQ session environment variable.

The IMPLICIT_PDQ session environment option is available only on systems that support **PDQPRIORITY**.
By default, the IMPLICIT_PDQ session environment variable is set to `OFF`. When IMPLICIT_PDQ is set to OFF, the server does not override the current **PDQPRIORITY** setting.

To enable the database server to calculate memory allocations for queries and to distribute memory among query operators according to their needs, enter the following statement before you issue the query:

```
SET ENVIRONMENT IMPLICIT_PDQ ON;
```

If you instead set the IMPLICIT_PDQ value to an integer in the range from 1 to 100, the database server scales its estimate by the specified value. For example, the following example restricts memory allocation in subsequent queries of the session to half of the current **PDQPRIORITY** setting:

```
SET ENVIRONMENT IMPLICIT_PDQ '50';
```

If you set a low IMPLICIT_PDQ value, the amount of memory assigned to the query is proportionally reduced, which might increase the amount of query-operator overflow.

The IMPLICIT_PDQ functionality for a query requires at least LOW distribution statistics on all tables that the query accesses. If distribution statistics are missing for one or more tables that the query references, the IMPLICIT_PDQ setting has no effect on the resources available for query execution. This restriction also applies to star-join queries, which are not supported in the case of missing statistics.

Limiting PDQ resource allocation by setting BOUND_IMPL_PDQ

If IMPLICIT_PDQ is set to ON or to a numeric value, you can also use the BOUND_IMPL_PDQ session environment option of the SET ENVIRONMENT statement of SQL to specify that the allocated PDQ memory should be no greater than the current explicit PDQPRIORITY value or range. If the IMPLICIT_PDQ session environment setting is OFF, whether explicitly off by default, then the BOUND_IMPL_PDQ setting has no effect.

For example, the following statement forces the database server to use explicit PDQPRIORITY values as guidelines in allocating memory, if the IMPLICIT_PDQ session environment option has already been set:

```
SET ENVIRONMENT BOUND_IMPL_PDQ ON;
```

If the IMPLICIT_PDQ setting is an integer in the range from 1 to 100, the explicit PDQPRIORITY value is scaled by that setting as a percentage during the current session. When the BOUND_IMPL_PDQ session environment option is set to ON (or to one), you require the database server to use the explicit PDQPRIORITY setting as the upper bound for memory that can be allocated to a query. If you set both IMPLICIT_PDQ and BOUND_IMPL_PDQ, then the explicit PDQPRIORITY value determines the upper limit of memory that can be allocated to a query.

If you include an integer value in the SET ENVIRONMENT statement, you must put quote marks around the value. However, do not put quote marks around the ON and OFF keywords.

The following examples are statements with integer values:

```
SET ENVIRONMENT IMPLICIT_PDQ "50";
SET ENVIRONMENT BOUND_IMPL_PDQ "1";
```


User control of PDQ resources

To indicate the PDQ priority of a query, you can set the **PDQPRIORITY** environment variable or execute the SET PDQPRIORITY statement prior to issuing a query. These PDQ priority options allow you to request a certain amount of parallel-processing resources for the query.

The resources that you request and the amount of resources that the database server allocates for the query can differ. This difference occurs when the database server administrator uses the MAX_PDQPRIORITY configuration parameter to put a ceiling on user-requested resources, as the following topic explains.

DBA control of resources for PDQ and DSS queries

To manage the total amount of resources that the database server allocates to parallel database and decision-support (DSS) queries, the database server administrator can set the environment variable and configuration parameters.

- Controlling resources allocated to PDQ
  To control resources allocated to PDQ, you can set the **PDQPRIORITY** environment variable. The queries that do not set the **PDQPRIORITY** environment variable before they issue a query do not use PDQ. In addition, to place a ceiling on user-specified PDQ priority levels, you can set the MAX_PDQPRIORITY configuration parameter.
- DBA control of resources allocated to decision-support queries
  A DBA can control the resources that the database server allocates to decision-support queries by setting the DS_TOTAL_MEMORY, DS_MAX_SCANS, and DS_MAX_QUERIES configuration parameters.

Controlling resources allocated to PDQ

To control resources allocated to PDQ, you can set the **PDQPRIORITY** environment variable. The queries that do not set the **PDQPRIORITY** environment variable before they issue a query do not use PDQ. In addition, to place a ceiling on user-specified PDQ priority levels, you can set the MAX_PDQPRIORITY configuration parameter.

When you set the **PDQPRIORITY** environment variable and MAX_PDQPRIORITY parameter, you exert control over the resources that the database server allocates between OLTP and DSS applications. For example, if OLTP processing is particularly heavy during a certain period of the day, you might want to set MAX_PDQPRIORITY to `0`. This configuration parameter puts a ceiling on the resources requested by users who use the **PDQPRIORITY** environment variable, so PDQ is turned off until you reset MAX_PDQPRIORITY to a nonzero value.

DBA control of resources allocated to decision-support queries

A DBA can control the resources that the database server allocates to decision-support queries by setting the DS_TOTAL_MEMORY, DS_MAX_SCANS, and DS_MAX_QUERIES configuration parameters.

In addition to setting limits for decision-support memory and the number of decision-support queries that can run concurrently, the database server uses these parameters to determine the amount of memory to allocate to individual decision-support queries as users submit them. To do so, the database server first calculates a unit of memory called a *quantum* by dividing DS_TOTAL_MEMORY by DS_MAX_QUERIES. When a user issues a query, the database server allocates a percent of the available quanta equal to the PDQ priority of the query.

You can also limit the number of concurrent decision-support scans that the database server allows by setting the DS_MAX_SCANS configuration parameter.

Previous versions of the database server allowed you to set a PDQ priority configuration parameter in the ONCONFIG file. If your applications depend on a global setting for PDQ priority, you can use one of the following methods:

- *For UNIX:* Define **PDQPRIORITY** as a shared environment variable in the **informix.rc** file. For more information about the **informix.rc** file, see the *IBM® Informix® Guide to SQL: Reference*.
- *For Windows:* Set the **PDQPRIORITY** environment variable for a particular group through a logon profile. For more information about the logon profile, see your operating-system manual.

Monitoring resources used for PDQ and DSS queries

You can monitor the resources (shared memory and threads) that the Memory Grant Manager (MGM) has allocated for PDQ queries and the resources that PDQ and decision-support (DSS) queries currently use.

You monitor PDQ resource use in the following ways:

- Run individual **onstat** utility commands to capture information about specific aspects of a running query.
- Execute a SET EXPLAIN statement before you run a query to write the query plan to an output file.

- Monitoring PDQ resources by using the onstat Utility
  You can use various **onstat** utility commands to determine how many threads are active and the shared-memory resources that those threads use.
- Identifying parallel scans in SET EXPLAIN output
  When PDQ is turned on, the SET EXPLAIN output shows whether the optimizer chose parallel scans. If the optimizer chose parallel scans, the output lists `Parallel`. (If PDQ is turned off, the output lists `Serial`.)

Related concepts:
What PDQ is

Monitoring PDQ resources by using the onstat Utility

You can use various **onstat** utility commands to determine how many threads are active and the shared-memory resources that those threads use.

You can use the **onstat -g mgm** command to monitor how the Memory Grant Manager (MGM) coordinates memory use and to scan threads.

- Monitoring PDQ threads with onstat utility commands
  You can obtain information about all of the threads that are running for a decision-support query by running the **onstat -u** and **onstat -g ath** commands.
- Monitoring resources allocated for a session running a DSS query
  You can monitor the resources allocated for, and used by, a session that is running a decision-support (DSS) query by running the **onstat -g ses** command.

Related information:
onstat -g mgm command: Print MGM resource information

Monitoring PDQ threads with onstat utility commands

You can obtain information about all of the threads that are running for a decision-support query by running the **onstat -u** and **onstat -g ath** commands.

The **onstat -u** option lists all the threads for a session. If a session is running a decision-support query, the output lists the primary thread and any additional threads. For example, session `10` in Figure 1 has a total of five threads running.

Figure 1. onstat -u output

```
Userthreads
address  flags    sessid   user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
80fd48   ---P--- 45       chrisw   ttyp3    0        0    1     573      237
810460   ------- 10       chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42       chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10       chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47       chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46       chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10       chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43       chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10       chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10       chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44       chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

The **onstat -g ath** output also lists these threads and includes a **name** column that indicates the role of the thread. Threads that a primary decision-support thread started have a name that indicates their role in the decision-support query. For example, Figure 2 lists four *scan* threads, started by a primary thread (**sqlexec**).

Figure 2. onstat -g ath Output

```
Threads:
tid      tcb      rstcb    prty     status                    vp-class    name
...
11       994060   0        4        sleeping(Forever)           1cpu     kaio
12       994394   80f2a4   2        sleeping(secs: 51)          1cpu     btclean
26       99b11c   80f630   4        ready                       1cpu     onmode_mon
32       a9a294   812b64   2        ready                       1cpu     sqlexec
113      b72a7c   810b78   2        ready                       1cpu     sqlexec
114      b86c8c   81244c   2        cond wait(netnorm)          1cpu     sqlexec
115      b98a7c   812ef0   2        cond wait(netnorm)          1cpu     sqlexec
116      bb4a24   80fd48   2        cond wait(netnorm)          1cpu     sqlexec
117      bc6a24   81161c   2        cond wait(netnorm)          1cpu     sqlexec
118      bd8a24   811290   2        ready                       1cpu     sqlexec
119      beae88   810f04   2        cond wait(await_MC1)        1cpu     scan_1.0
120      a8ab48   8127d8   2        ready                       1cpu     scan_2.0
121      a96850   810460   2        ready                       1cpu     scan_2.1
122      ab6f30   8119a8   2        running                     1cpu     scan_2.2
```

Related concepts:
Monitoring resources allocated for a session running a DSS query

Monitoring resources allocated for a session running a DSS query

You can monitor the resources allocated for, and used by, a session that is running a decision-support (DSS) query by running the **onstat -g ses** command.

The **onstat -g ses** option displays the following information:

- The shared memory allocated for a session that is running a decision-support query
- The shared memory used by a session that is running a decision-support query
- The number of threads that the database server started for a session

For example, in Figure 1, session number `49` is running five threads for a decision-support query.

Figure 1. onstat -g ses output

```
session                                      #RSAM    total      used
id       user     tty      pid    hostname  threads  memory     memory
57       informix -        0      -         0        8192       5908
56       user_3   ttyp3    2318   host_10   1        65536      62404
55       user_3   ttyp3    2316   host_10   1        65536      62416
54       user_3   ttyp3    2320   host_10   1        65536      62416
53       user_3   ttyp3    2317   host_10   1        65536      62416
52       user_3   ttyp3    2319   host_10   1        65536      62416
51       user_3   ttyp3    2321   host_10   1        65536      62416
49       user_1   ttyp2    2308   host_10   5        188416     178936
2        informix -        0      -         0        8192       6780
1        informix -        0      -         0        8192       4796
```

Related concepts:
Monitoring PDQ threads with onstat utility commands

Identifying parallel scans in SET EXPLAIN output

When PDQ is turned on, the SET EXPLAIN output shows whether the optimizer chose parallel scans. If the optimizer chose parallel scans, the output lists `Parallel`. (If PDQ is turned off, the output lists `Serial`.)

If PDQ is turned on, the optimizer also indicates the maximum number of threads that are required to answer the query. The `# of Secondary Threads` field in the SET EXPLAIN output indicates the number of threads that are required in addition to your user session thread. The total number of threads necessary is the number of secondary threads plus 1.

The following example shows SET EXPLAIN output for a table with fragmentation and PDQ priority set to `LOW`:

```
SELECT * FROM t1 WHERE c1 > 20

Estimated Cost: 2
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN (Parallel, fragments: 2)

 Filters: informix.t1.c1 > 20

# of Secondary Threads = 1
```

The following example of partial SET EXPLAIN output shows a query with a hash join between two fragmented tables and PDQ priority set to `ON`. The query is marked with `DYNAMIC HASH JOIN`, and the table on which the hash is built is marked with `Build Outer`.

```
QUERY:
------
SELECT h1.c1, h2.c1 FROM h1, h2 WHERE h1.c1 = h2.c1

Estimated Cost: 2
Estimated # of Rows Returned: 5

1) informix.h1: SEQUENTIAL SCAN (Parallel, fragments: ALL)

2) informix.h2: SEQUENTIAL SCAN (Parallel, fragments: ALL)


DYNAMIC HASH JOIN (Build Outer)
 Dynamic Hash Filters: informix.h1.c1 = informix.h2.c1

# of Secondary Threads = 6
```

The following example of partial SET EXPLAIN output shows a table with fragmentation, PDQ priority set to `LOW`, and an index that was selected as the access plan:

```
SELECT * FROM t1 WHERE c1 < 13

Estimated Cost: 2
Estimated # of Rows Returned: 1

1) informix.t1: INDEX PATH

 (1) Index Keys: c1 (Parallel, fragments: ALL)
 Upper Index Filter: informix.t1.c1 < 13


# of Secondary Threads = 3
```


Improving individual query performance

You can test, monitor, and improve queries.

- Test queries using a dedicated test system
  You can test a query on a system that does not interfere with production database servers. However, you must be careful, because testing queries on a separate system might distort your tuning decisions.
- Display the query plan
  Before you change a query, display its query plan to determine the kind and amount of resources that the query requires. The query plan shows what parallel scans are used, the maximum number of threads required, and the indexes used.
- Improve filter selectivity
  You can control the amount of information that a query evaluates. The greater the precision with which you specify the desired rows, the greater the likelihood that your queries will complete quickly.
- Automatic statistics updating
  The database server updates statistics automatically according to a predefined schedule and a set of expiration policies. The Auto Update Statistics (AUS) maintenance system identifies tables and indexes that require new optimizer statistics and runs the appropriate UPDATE STATISTICS statements to optimize query performance.

- Update statistics when they are not generated automatically
  The UPDATE STATISTICS statement updates the statistics in the system catalog tables that the optimizer uses to determine the lowest-cost query plan.
- Improve performance by adding or removing indexes
  You can often improve the performance of a query by adding or, in some cases, removing indexes. You can also enable the optimizer to automatically fetch a set of keys from an index buffer.
- Optimizer estimates of distributed queries
  The optimizer assumes that access to a row from a remote database takes longer than access to a row in a local database. The optimizer estimates include the cost of retrieving the row from disk and transmitting it across the network.
- Improve sequential scans
  You can improve the performance of sequential read operations on large tables by eliminating repeated sequential scans.
- Enable view folding to improve query performance
  You can significantly improve the performance of a query that involves a view by enabling view folding.
- Reduce the join and sort operations
  After you understand what the query is doing, you can look for ways to obtain the same output with less effort.
- Optimize user-response time for queries
  You can influence the amount of time that Informix takes to optimize a query and to return rows to a user.
- Optimize queries for user-defined data types
  Queries that access user-defined data types (UDTs) can take advantage of the same performance features that built-in data types use.
- Optimize queries with the SQL statement cache
  Before the database server runs an SQL statement, it must first parse and optimize the statement. Optimizing statements can be time consuming, depending on the size of the SQL statement.
- Monitor sessions and threads
  You can monitor the number of active sessions and threads and the amount of resources that they are using. Monitoring sessions and threads is important for sessions that perform queries as well as sessions that perform inserts, updates, and deletes.
- Monitor transactions
  You can monitor transactions to track open transactions and the locks that those transactions hold. You can use several **onstat** utility options to view transaction, lock, and session statistics.

Related information:
Tune the new version for performance and adjust queries

Test queries using a dedicated test system

You can test a query on a system that does not interfere with production database servers. However, you must be careful, because testing queries on a separate system might distort your tuning decisions.

Even if you use your database server as a data warehouse, you might sometimes test queries on a separate system until you understand the tuning issues that are relevant to the query.

If you are trying to improve performance of a large query, one that might take several minutes or hours to complete, you can prepare a scaled-down database in which your tests can complete more quickly. However, be aware of these potential problems:

- The optimizer can make different choices in a small database than in a large one, even when the relative sizes of tables are the same. Verify that the query plan is the same in the real and the model databases.
- Execution time is rarely a linear function of table size. For example, sorting time increases faster than table size, as does the cost of indexed access when an index goes from two to three levels. What appears to be a big improvement in the scaled-down environment can be insignificant when applied to the full database.

Therefore, any conclusion that you reach as a result of tests in the model database must be tentative until you verify them in the production database.

You can often improve performance by adjusting your query or data model with the following goals in mind:

- If you are using a multiuser system or a network, where system load varies widely from hour to hour, try to perform your experiments at the same time each day to obtain repeatable results. Start tests when the system load is consistently light so that you are truly measuring the impact of your query only.
- If the query is embedded in a complicated program, you can extract the SELECT statement and embed it in a DB-Access script.

Related information:
Tune the new version for performance and adjust queries

Display the query plan

Before you change a query, display its query plan to determine the kind and amount of resources that the query requires. The query plan shows what parallel scans are used, the maximum number of threads required, and the indexes used.

After you study the query plan, examine your data model to see if the changes this chapter suggests will improve the query.

You can display the query plan with one of the following methods:

- Execute one of the following SET EXPLAIN statements just before the query:
  - SET EXPLAIN ON
    This SQL statement displays the chosen query plan. For a description of the SET EXPLAIN ON output, see Report that shows the query plan chosen by the optimizer.
  - SET EXPLAIN ON AVOID_EXECUTE
    This SQL statement displays the chosen query plan and does not execute the query.

- Use one of the following EXPLAIN directives in the query:
  - EXPLAIN
  - EXPLAIN AVOID_EXECUTE

For more information about these EXPLAIN directives, see EXPLAIN directives.

Improve filter selectivity

You can control the amount of information that a query evaluates. The greater the precision with which you specify the desired rows, the greater the likelihood that your queries will complete quickly.

To control the amount of information that the query evaluates, use the WHERE clause of the SELECT statement. The conditional expression in the WHERE clause is commonly called a *filter*.

For information about how filter selectivity affects the query plan that the optimizer chooses, see Filters in the query. The following sections provide some guidelines to improve filter selectivity.

- Filters with user-defined routines
  You can improve the selectivity of query filters that include user-defined routines (UDRs).
- Avoid some filters
  For best performance, avoid filters for certain difficult regular expressions and filters for noninitial strings.
- Use join filters and post-join filters
  The database server provides support for using a subset of the ANSI join syntax.

Filters with user-defined routines

You can improve the selectivity of query filters that include user-defined routines (UDRs).

You can improve the selectivity if the UDRs have the following features:

- Functional indexes
  You can create a *functional index* on the resulting values of a user-defined routine or a built-in function that operates on one or more columns. When you create a functional index, the database server computes the return values of the function and stores them in the index. The database server can locate the return value of the function in an appropriate index without executing the function for each qualifying column.

  For more information about indexing user-defined functions, see Using a functional index.

- User-defined selectivity functions
  You can write a function that calculates the expected fraction of rows that qualify for the function. For a brief description of user-defined selectivity functions, see Selectivity and cost functions. For more information about how to write and register user-defined selectivity functions, see *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

Related concepts:
Avoid some filters
Use join filters and post-join filters

Avoid some filters

For best performance, avoid filters for certain difficult regular expressions and filters for noninitial strings.

- Avoid difficult regular expressions
  The MATCHES and LIKE keywords support *wildcard* matches, which are technically known as *regular expressions*. Some regular expressions are more difficult than others for the database server to process.
- Avoid noninitial substrings
  For best performance, avoid filters for noninitial strings. A filter based on a noninitial substring of a column requires the database server to test every value in the column.

Related concepts:
Filters with user-defined routines
Use join filters and post-join filters

Avoid difficult regular expressions

The MATCHES and LIKE keywords support *wildcard* matches, which are technically known as *regular expressions*. Some regular expressions are more difficult than others for the database server to process.

A wildcard in the initial position, as in the following example (find customers whose first names do not end in *y*), forces the database server to examine every value in the column:

```
SELECT * FROM customer WHERE fname NOT LIKE '%y'
```

You cannot use an index with such a filter, so the table in this example must be accessed sequentially.

If a difficult test for a regular expression is essential, avoid combining it with a join. If necessary, process the single table and apply the test for a regular expression to select the desired rows. Save the result in a temporary table and join that table to the others.

Regular-expression tests with wildcards in the middle or at the end of the operand do not prevent the use of an index when one exists.

Related concepts:
Avoid noninitial substrings

Avoid noninitial substrings

For best performance, avoid filters for noninitial strings. A filter based on a noninitial substring of a column requires the database server to test every value in the column.

For example, in the following code, a noninitial substring requires the database server to test every value in the column:

```
SELECT * FROM customer
   WHERE zipcode[4,5] > '50'
```

The database server cannot use an index to evaluate such a filter.

The optimizer uses an index to process a filter that tests an initial substring of an indexed column. However, the presence of the substring test can interfere with the use of a composite index to test both the substring column and another column.

Related concepts:
Avoid difficult regular expressions

Use join filters and post-join filters

The database server provides support for using a subset of the ANSI join syntax.

This syntax that includes the following keywords:

- ON keyword to specify the join condition and any optional join filters
- LEFT OUTER JOIN keywords to specify which table is the dominant table (also referred to as *outer table*)

For more information about this ANSI join syntax, see the *IBM® Informix® Guide to SQL: Syntax*.

In an ANSI outer join, the database server takes the following actions to process the filters:

- Applies the join condition in the ON clause to determine which rows of the subordinate table (also referred to as *inner table*) to join to the outer table
- Applies optional join filters in the ON clause before and during the join
  If you specify a join filter on a base inner table in the ON clause, the database server can apply it prior to the join, during the scan of the data from the inner table. Filters on a base subordinate table in the ON clause can provide the following additional performance benefits:
  - Fewer rows to scan from the inner table prior to the join
  - Use of index to retrieve rows from the inner table prior to the join
  - Fewer rows to join
  - Fewer rows to evaluate for filters in the WHERE clause

  For information about what occurs when you specify a join filter on an outer table in the ON clause, see the *IBM Informix Guide to SQL: Syntax*.
- Applies filters in the WHERE clause after the join
  Filters in the WHERE clause can reduce the number of rows that the database server needs to scan and reduce the number of rows returned to the user.

  The term *post-join filters* refers to these WHERE clause filters.

When distributed queries that use ANSI-compliant LEFT OUTER syntax for specifying joined tables and nested loop joins are executed, the query is sent to each participating database server for operations on local tables of those servers.

For example, the demonstration database has the **customer** table and the **cust_calls** table, which tracks customer calls to the service department. Suppose a certain call code had many occurrences in the past, and you want to see if calls of this kind have decreased. To see if customers no longer have this call code, use an outer join to list all customers.

Figure 1 shows a sample SQL statement to accomplish this ANSI join query and the SET EXPLAIN ON output for it.
Figure 1. SET EXPLAIN ON output for an ANSI join

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
c.phone, u.call_dtime, u.call_code, u.call_descr
```

```
FROM customer c
LEFT JOIN cust_calls u ON c.customer_num = u.customer_num
ORDER BY u.call_dtime

Estimated Cost: 14
Estimated # of Rows Returned: 29
Temporary Files Required For: Order By

1) virginia.c: SEQUENTIAL SCAN

2) virginia.u: INDEX PATH

   (1) Index Keys: customer_num call_dtime (Serial, fragments: ALL)
     Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

  ON-Filters:virginia.c.customer_num = virginia.u.customer_num
  NESTED LOOP JOIN(LEFT OUTER JOIN)
```

Look at the following lines in the SET EXPLAIN ON output in Figure 1:

- The `ON-Filters:` line lists the join condition that was specified in the ON clause.
- The last line of the SET EXPLAIN ON output shows all three keywords `(LEFT OUTER JOIN)` for the ANSI join even though this query specifies only the LEFT JOIN keywords in the FROM clause. The OUTER keyword is optional.

Figure 2 shows the SET EXPLAIN ON output for an ANSI join with a join filter that checks for calls with the `I` call_code.

Figure 2. SET EXPLAIN ON output for a join filter in an ANSI join

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c LEFT JOIN cust_calls u
ON c.customer_num = u.customer_num
AND u.call_code = 'I'
ORDER BY u.call_dtime

Estimated Cost: 13
Estimated # of Rows Returned: 25
Temporary Files Required For: Order By

  1) virginia.c: SEQUENTIAL SCAN

  2) virginia.u: INDEX PATH

     Filters: virginia.u.call_code = 'I'

      (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
          Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

ON-Filters:(virginia.c.customer_num = virginia.u.customer_num
               AND virginia.u.call_code = 'I' )
NESTED LOOP JOIN(LEFT OUTER JOIN)
```

The main differences between the output in Figure 1 and Figure 2 are as follows:

- The optimizer chooses a different index to scan the inner table.
  This new index exploits more filters and retrieves a smaller number of rows. Consequently, the join operates on fewer rows.
- The ON clause join filter contains an additional filter.

The value in the `Estimated # of Rows Returned` line is only an estimate and does not always reflect the actual number of rows returned. The sample query in Figure 2 returns fewer rows than the query in Figure 1 because of the additional filter.

Figure 3 shows the SET EXPLAIN ON output for an ANSI join query that has a filter in the WHERE clause.

Figure 3. SET EXPLAIN ON output for the WHERE clause filter in an ANSI join

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
   c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c LEFT JOIN cust_calls u
ON c.customer_num = u.customer_num
   AND u.call_code = 'I'
WHERE c.zipcode = "94040"
ORDER BY u.call_dtime

Estimated Cost: 3
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

  1) virginia.c: INDEX PATH

    (1) Index Keys: zipcode   (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.zipcode = '94040'

  2) virginia.u: INDEX PATH

     Filters: virginia.u.call_code = 'I'

    (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

ON-Filters:(virginia.c.customer_num = virginia.u.customer_num
      AND virginia.u.call_code = 'I' )
```

```
NESTED LOOP JOIN(LEFT OUTER JOIN)

PostJoin-Filters:virginia.c.zipcode = '94040'
```

The main differences between the output in Figure 2 and Figure 3 are as follows:

- The index on the **zipcode** column in the post-join filter is chosen for the dominant table.
- The `PostJoin-Filters` line shows the filter in the WHERE clause.

Related concepts:
Filters with user-defined routines
Avoid some filters

Automatic statistics updating

The database server updates statistics automatically according to a predefined schedule and a set of expiration policies. The Auto Update Statistics (AUS) maintenance system identifies tables and indexes that require new optimizer statistics and runs the appropriate UPDATE STATISTICS statements to optimize query performance.

The AUS maintenance system updates the statistics for tables that are in logged databases, regardless of the database locale. By making current table statistics available to the query optimizer, the AUS maintenance system can reduce the risk of performance degradation from inefficient query plans.

Depending on your system, you might need to adjust the AUS expiration policies or schedule. The AUS maintenance system resides in the **sysadmin** database.

- How AUS works
  The Auto Update Statistics (AUS) maintenance system uses a combination of Scheduler sensors, tasks, thresholds, and tables to evaluate and update statistics.
- AUS expiration policies
  The Auto Update Statistics (AUS) maintenance system uses expiration policies as criteria for identifying user tables that have changed to the extent that their statistics need to be recalculated.
- Viewing AUS statements
  You can view the UPDATE STATISTICS statements generated by the AUS maintenance system in the **aus_cmd_list** view before they are run and in the **aus_cmd_comp** view after they are run successfully. Both tables are in the **sysadmin** database.
- Prioritizing databases in AUS
  You can assign a priority to each of your databases in the AUS maintenance system.
- Rescheduling AUS
  You can change when and for how long the Auto Update Statistics Refresh task runs.
- Disabling AUS
  You can prevent statistics from being updated automatically by disabling the AUS maintenance system.

Related concepts:
Update statistics when they are not generated automatically

How AUS works

The Auto Update Statistics (AUS) maintenance system uses a combination of Scheduler sensors, tasks, thresholds, and tables to evaluate and update statistics.

The Scheduler tasks, sensors, thresholds, and tables reside in the **sysadmin** database. By default, only user **informix** is granted access to the **sysadmin** database.

The following sequence of events describes how statistics are automatically updated:

1. The **mon_table_profile** sensor of the Scheduler runs every day to read data from the **systables** table in the **sysmaster** database. The sensor updates the **mon_table_profile** table in the **sysadmin** database with information about how much each table has changed.
2. The Auto Update Statistics Evaluation task gathers information every day from the **mon_table_profile** table and the **systable**, **sysdistrib**, **syscolumns**, and **sysindices** tables in the **sysmaster** database.
3. The Auto Update Statistics Evaluation task determines which tables need updates based on the expiration policies.
4. The Auto Update Statistics Evaluation task generates UPDATE STATISTICS statements and inserts them into the **aus_command** table in the **sysadmin** database.
5. The Auto Update Statistics Refresh task runs the UPDATE STATISTICS statements from the **aus_command** table on Saturday and Sunday mornings between 1:00 AM and 5:00 AM and inserts the results back into the **aus_command** table. Any UPDATE STATISTICS statements that did not complete before 5:00 AM remain in the **aus_command** table.

The following table describes the tasks, sensors, thresholds, tables, and views in the **sysadmin** database that comprise the AUS maintenance system.

Table 1. AUS components

Component	Type	Description
mon_table_profile	sensor	Compiles table profile information, including the total number of updates, inserts, and deletes that occurred on each table. Defined in the **ph_task** table.
mon_table_profile	table	Stores table profile information gathered by its sensor. Many other Scheduler tasks use information from this table.
Auto Update Statistics Evaluation	task	Identifies tables with stale statistics, based on expiration policies, and generates UPDATE STATISTICS statements for those tables. Defined in the **ph_task** table.

Component	Type	Description
aus_command	table	Stores a list of prioritized UPDATE STATISTICS statements that are scheduled to be run, and the results of those statements after they are run. The **aus_cmd_state** column indicates the status of each UPDATE STATISTICS statement: • P = Pending • I = In progress • E = Error • C = Complete without errors If the command status is E, the associated SQL error code is listed in the **aus_cmd_err_sql** column and the associated ISAM error code is listed in the **aus_cmd_err_isam** column. The **aus_cmd_runtime** shows the time that is elapsed for the update statistics command to complete. The **aus_cmd_time** shows the start time for the update statistics command.
Auto Update Statistics Refresh	task	Runs the prepared UPDATE STATISTICS statements on Saturdays and Sundays between 1:00 AM and 5:00 AM. Defined in the **ph_task** table.
expiration policies	thresholds	Define the criteria for when to update statistics. Defined in the **ph_threshold** table.
aus_cmd_comp	view	Shows information from the **aus_command** table about UPDATE STATISTICS statements that were run successfully.
aus_cmd_list	view	Shows information from the **aus_command** table about UPDATE STATISTICS statements that are scheduled to be run.

For information about other features of the Scheduler, see its description in the *IBM® Informix® Administrator's Guide*. For information about the **sysadmin** database, see the *IBM Informix Administrator's Reference*.

AUS expiration policies

The Auto Update Statistics (AUS) maintenance system uses expiration policies as criteria for identifying user tables that have changed to the extent that their statistics need to be recalculated.

Internally, the AUS maintenance system automatically skips any tables or fragments that have current statistics and prioritizes tables or fragments that have more changes. Therefore, all tables are scheduled for updating statistics. For more information, see Automatic management of data distribution statistics.

The **ph_threshold** table of the **sysadmin** database stores the following configurable thresholds for defining AUS expiration policies.

Table 1. AUS expiration policy thresholds

Threshold Name	Default Value	Description
AUS_AGE	30 (days)	A time-based expiration policy. Statistics or distributions are updated for a table after this amount of time regardless of how much data has changed.
AUS_AUTO_RULES	1 (enabled)	If enabled, statistics are updated using the higher of the following default minimum guidelines or user-created distribution options: • All tables are updated in LOW mode. • All the leading index keys are updated in HIGH mode. • All non-leading index keys are updated in MEDIUM mode. • The minimum resolution for MEDIUM mode is 2.0. • The minimum confidence for MEDIUM mode is 0.95. • The minimum resolution for HIGH mode is 0.5. If the UPDATE STATISTICS statement was run manually for a table, the UPDATE STATISTICS statements generated by the AUS maintenance system do not reduce the level, resolution, confidence, or sampling size options. If disabled by being set to 0, the AUS maintenance system checks which columns have existing distributions and generates update statistics statements to refresh them.
AUS_PDQ	10 (priority)	The PDQ priority for UPDATE STATISTICS statements run by the AUS maintenance system. By default, all fragments for each table are analyzed in parallel. For more information about PDQ priority, see Update statistics in parallel on very large databases.
AUS_SMALL_TABLES	100 (rows)	Statistics or distributions are updated every time for a table that has fewer than this number of rows.

- Changing AUS expiration policies
  You can change AUS expiration policies to customize how often statistics are updated based on how old the statistics are, how much data has changed, or how large the table is.

Changing AUS expiration policies

You can change AUS expiration policies to customize how often statistics are updated based on how old the statistics are, how much data has changed, or how large the table is.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To change the value of an expiration policy, update the **value** column in the **ph_threshold** table in the **sysadmin** database.
For example, if you find that queries against small tables with 1000 rows or fewer run faster if their statistics are updated more frequently, you can change the expiration policy to ensure that their statistics are updated every week. The following example changes the value of the AUS_SMALL_TABLES threshold to 1000:

```
UPDATE ph_threshold
SET value = 1000
WHERE name = "AUS_SMALL_TABLES";
```

The new threshold takes effect the next time the Auto Update Statistics Evaluator task runs.

Viewing AUS statements

You can view the UPDATE STATISTICS statements generated by the AUS maintenance system in the **aus_cmd_list** view before they are run and in the **aus_cmd_comp** view after they are run successfully. Both tables are in the **sysadmin** database.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.
To view all scheduled UPDATE STATISTICS statements, run this statement:

```
SELECT * FROM aus_cmd_list;
```

To see all UPDATE STATISTICS statements that were run successfully in the previous 30 days, run this statement:

```
SELECT * FROM aus_cmd_comp;
```

To view all UPDATE STATISTICS statements that failed, run this statement:

```
SELECT aus_cmd_exe, aus_cmd_err_sql, aus_cmd_err_isam
FROM aus_command
WHERE aus_cmd_state = "E";
```


Prioritizing databases in AUS

You can assign a priority to each of your databases in the AUS maintenance system.

By default all databases have a medium priority. You can assign specific databases a high or a low priority to ensure that statistics for your most important databases are updated first. Statistics for low priority databases are updated after high and medium priority databases, if time and resources permit. For example, if you have a system with a production and a test database, you can assign the production database a high priority and the test database a low priority. You can also disable AUS for a database.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To assign a priority to a database in AUS, add a row to the **ph_threshold** table in the **sysadmin** database:

- High priority: Add a row with the **name** column set to AUS_DATABASE_HIGH and the **value** column set to the name of the database.
- Low priority: Add a row with the **name** column set to AUS_DATABASE_LOW and the **value** column set to the name of the database.
- Disable: Add a row with the **name** column set to AUS_DATABASE_DISABLE and the **value** column set to the name of the database.

If you assign more than one priority to a database, the higher priority takes precedence.

Example

The following statement sets the priority for the database that is named **my_database** to high:

```
INSERT INTO ph_threshold(id, name, task_name, value, value_type, description)
     VALUES(0,
            "AUS_DATABASE_HIGH",
            "Auto Update Statistics Evaluation",
            "my_database",
            "STRING",
            "Rank this database as high priority to get its tables done first");
```


Rescheduling AUS

You can change when and for how long the Auto Update Statistics Refresh task runs.

Updating statistics is resource-intensive. Therefore, by default, statistics are automatically updated on Saturdays and Sundays between 1:00 AM and 5:00 AM. If you find that not all pending UPDATE STATISTICS statements can be run in this time period, or you want statistics to be refreshed more often, you can change the start time, the end time, and the days of the week to perform this task.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To change the schedule of the Auto Update Statistics Refresh task, update the **ph_task** table where the value of the **tk_name** column is Auto Update Statistics Refresh. The following example changes the ending time of the task to 6:00 AM:

```
UPDATE ph_task
SET tk_stop_time = "06:00:00"
WHERE tk_name = "Auto Update Statistics Refresh";
```

The following example changes the days that the task is run to every day of the week (Saturday and Sunday are enabled by default):

```
UPDATE ph_task
SET tk_monday = "T",
tk_tuesday = "T",
tk_wednesday = "T",
tk_thursday = "T",
tk_friday = "T"
WHERE tk_name = "Auto Update Statistics Refresh";
```


Disabling AUS

You can prevent statistics from being updated automatically by disabling the AUS maintenance system.

You must be connected to the **sysadmin** database as user **informix** or another authorized user.

To disable AUS, you must disable both the Auto Update Statistics Evaluation task and the Auto Update Statistics Refresh task:

1. Update the value of the **tk_enable** column of the **ph_task** table to `F` where the value of the **tk_name** column is Auto Update Statistics Evaluation.
2. Update the value of the **tk_enable** column of the **ph_task** table to `F` where the value of the **tk_name** column is Auto Update Statistics Refresh.

The following example disables both tasks:

```
UPDATE ph_task
SET tk_enable = "F"
WHERE tk_name = "Auto Update Statistics Evaluation";

UPDATE ph_task
SET tk_enable = "F"
WHERE tk_name = "Auto Update Statistics Refresh";
```


Update statistics when they are not generated automatically

The UPDATE STATISTICS statement updates the statistics in the system catalog tables that the optimizer uses to determine the lowest-cost query plan.

Important: You do not need to run UPDATE STATISTICS operations when the statistics are generated automatically.
The following statistics are generated automatically by the CREATE INDEX statement, with or without the ONLINE keyword:

- Index-level statistics, equivalent to the statistics gathered in the UPDATE STATISTICS operation in LOW mode, for B-tree indexes.
- Column-distribution statistics, equivalent to the distribution generated in the UPDATE STATISTICS operation in HIGH mode, for a non-opaque leading indexed column of an ordinary B-tree index.

To ensure that the optimizer selects a query plan that best reflects the current state of your tables, run UPDATE STATISTICS at regular intervals when the statistics are not generated automatically.

Tip: If you run UPDATE STATISTICS LOW on the **sysutils** database before you use ON-Bar, the time ON-BAR needs for processing is reduced.
The following table summarizes when to run different UPDATE STATISTICS statements if the statistics are not generated automatically. If you need to run UPDATE STATISTICS statements and you have many tables, you can write a script to generate these UPDATE STATISTICS statements.

When to Execute	UPDATE STATISTICS Statement	Reference for Details and Examples
Number of rows has changed significantly	UPDATE STATISTICS LOW DROP DISTRIBUTIONS	Update the statistics for the number of rows or Drop data distributions if necessary when upgrading
For all columns that are not the leading column of any index	UPDATE STATISTICS LOW	Creating data distributions
Queries have non-indexed join columns or filter columns	UPDATE STATISTICS MEDIUM DISTRIBUTIONS ONLY	Creating data distributions
Queries have an indexed join columns or filter columns	UPDATE STATISTICS HIGH table (leading column in index)	Creating data distributions

When to Execute	UPDATE STATISTICS Statement	Reference for Details and Examples
Queries have a multicolumn indexed defined on join columns or filter columns	UPDATE STATISTICS HIGH table (first differing column in multicolumn index)	Creating data distributions
Queries have a multicolumn indexed defined on join columns or filter columns	UPDATE STATISTICS LOW table (all columns in multicolumn index)	Creating data distributions
Queries have many small tables (fit into one extent)	UPDATE STATISTICS HIGH on small tables	Creating data distributions
Queries use SPL routines	UPDATE STATISTICS FOR PROCEDURE	Reoptimizing SPL routines

For information about the specific statistics that the database server keeps in the system catalog tables, see Statistics held for the table and index.

- Update the statistics for the number of rows
  When you run UPDATE STATISTICS LOW, the database server updates the statistics in the table, row, and page counts in the system catalog tables. You should run UPDATE STATISTICS LOW as often as necessary to ensure that the statistic for the number of rows is as current as possible.
- Drop data distributions if necessary when upgrading
  When you upgrade to a new version of the database server, you might need to drop distributions to remove the old distribution structure in the **sysdistrib** system catalog table.
- Creating data distributions
  You can generate statistics for a table and you can build data distributions for each table that your query accesses.
- Updating statistics for join columns
  In some situations, you might want to run the UPDATE STATISTICS statement with the HIGH keyword for specific join columns.
- Updating statistics for columns with user-defined data types
  Programmers can write functions that gather statistics for columns with user-defined data types. You can store the data distributions for user-defined data types in an sbspace.
- Update statistics in parallel on very large databases
  If you have an extremely large database and indexes are fragmented, UPDATE STATISTICS LOW can automatically run statements in parallel.
- Adjust the amount of memory and disk space for UPDATE STATISTICS
  When you execute the UPDATE STATISTICS statement, the database server uses memory and disk to sort and construct data distributions. You can affect the amount of memory and disk space available for UPDATE STATISTICS operations.
- Data sampling during update statistics operations
  If you have a large b-tree index with more than 100 K leaf pages, you can generate index statistics based on sampling when you run UPDATE STATISTICS statements in LOW mode. The gathering of statistics through sampling can increase the speed of the update statistics operation.
- Display data distributions
  You can use the **dbschema** utility to display data distributions.

Related concepts:
Automatic statistics updating
Related information:
UPDATE STATISTICS statement

Update the statistics for the number of rows

When you run UPDATE STATISTICS LOW, the database server updates the statistics in the table, row, and page counts in the system catalog tables. You should run UPDATE STATISTICS LOW as often as necessary to ensure that the statistic for the number of rows is as current as possible.

If the cardinality of a table changes often, run the statement more often for that table.

LOW is the default mode for UPDATE STATISTICS.

The following sample SQL statement updates the statistics in the **systables**, **syscolumns**, and **sysindexes** system catalog tables but does not update the data distributions:

```
UPDATE STATISTICS FOR TABLE tab1;
```


Drop data distributions if necessary when upgrading

When you upgrade to a new version of the database server, you might need to drop distributions to remove the old distribution structure in the **sysdistrib** system catalog table.

To drop the old distribution structure in the **sysdistrib** system catalog table, run this statement:

```
UPDATE STATISTICS DROP DISTRIBUTIONS;
```

- Drop distributions in LOW mode without gathering statistics
  You can remove distribution information from the **sysdistrib** table and update the **systables.version** column in the system catalog for those tables whose distributions were dropped, without gathering any LOW mode table and index statistics.

Drop distributions in LOW mode without gathering statistics

You can remove distribution information from the **sysdistrib** table and update the **systables.version** column in the system catalog for those tables whose distributions were dropped, without gathering any LOW mode table and index statistics.

You do this using the DROP DISTRIBUTIONS ONLY option in the UPDATE STATISTICS statement. Using the DROP DISTRIBUTIONS ONLY option can result in faster performance because the database server does not gather the table and index statistics that the LOW mode option generates when the ONLY keyword does not follow the DROP DISTRIBUTIONS keywords.

For detailed information about how to use the DROP DISTRIBUTIONS ONLY option, see the *IBM® Informix® Guide to SQL: Syntax*.

Creating data distributions

You can generate statistics for a table and you can build data distributions for each table that your query accesses.

(You do not need to run UPDATE STATISTICS operations when the statistics are generated automatically.)
The database server creates data distributions, which provide information to the optimizer, any time the UPDATE STATISTICS MEDIUM or UPDATE STATISTICS HIGH command is executed.

Important:
The database server creates data distributions by sampling a column's data, sorting the data, building distributions bins, and inserting the results into the **sysdistrib** system catalog table.

You can control the sample size for the scan through the keyword HIGH or MEDIUM. The difference between UPDATE STATISTICS HIGH and UPDATE STATISTICS MEDIUM is the number of rows sampled. UPDATE STATISTICS HIGH scans the entire table, while UPDATE STATISTICS MEDIUM samples only a subset of rows, based on the confidence and resolution used by the UPDATE STATISTICS statement.

You can use the LOW keyword with the UPDATE STATISTICS statement only for fully qualified index keys.

If a distribution has been generated for a column, the optimizer uses that information to estimate the number of rows that match a query against a column. Data distributions in **sysdistrib** supersede values in the **colmin** and **colmax** column of the **syscolumns** system catalog table when the optimizer estimates the number of rows returned.

When you use data-distribution statistics for the first time, try to update statistics in MEDIUM mode for all your tables and then update statistics in HIGH mode for all columns that head indexes. This strategy produces statistical query estimates for the columns that you specify. These estimates, on average, have a margin of error less than *percent* of the total number of rows in the table, where *percent* is the value that you specify in the RESOLUTION clause in the MEDIUM mode. The default percent value for MEDIUM mode is 2.5 percent. (For columns with HIGH mode distributions, the default resolution is 0.5 percent.)

With the DISTRIBUTIONS ONLY option, you can execute UPDATE STATISTICS MEDIUM at the table level or for the entire system because the overhead of the extra columns is not large.

The database server uses the storage locations that the DBSPACETEMP environment variable specifies only when you use the HIGH option of UPDATE STATISTICS.

You can prevent UPDATE STATISTICS operations from using indexes when sorting rows by setting the third parameter of the DBUPSPACE environment variable to a value of `1`.

For each table that your query accesses, build data distributions according to the following guidelines. Also see the examples below the guidelines.

To generate statistics on a table:

1. Identify the set of all columns that appear in any single-column or multi-column index on the table.
2. Identify the subset that includes all columns that are not the leading column of any index.
3. Run UPDATE STATISTICS LOW on each column in that subset.

To build data distributions for each table that your query accesses:

1. Run a single UPDATE STATISTICS MEDIUM for all columns in a table that do not head an index.
   Use the default parameters unless the table is very large, in which case you should use a resolution of `1.0` and confidence of `0.99`.
2. Run the following UPDATE STATISTICS statement to create distributions for non-index join columns and non-index filter columns:

   ```
   UPDATE STATISTICS MEDIUM DISTRIBUTIONS ONLY;
   ```

3. Run UPDATE STATISTICS HIGH for all columns that head an index. For the fastest execution time of the UPDATE STATISTICS statement, you must execute one UPDATE STATISTICS HIGH statement for each column, as shown in the example below this procedure.
4. If you have indexes that begin with the same subset of columns, run UPDATE STATISTICS HIGH for the first column in each index that differs, as shown in the second example below this procedure.
5. For each single-column or multi-column index on the table:
   a. Identify the set of all columns that appear in the index.
   b. Identify the subset that includes all columns that are not the leading column of any index.
   c. Run UPDATE STATISTICS LOW on each column in that subset. (LOW is the default.)
6. For the tables on which indexes were created in Step 3, run an UPDATE STATISTICS statement to update the **sysindexes** and **syscolumns** system catalog tables, as shown in the following example:

   ```
   UPDATE STATISTICS FOR TABLE t1(a,b,e,f);
   ```

7. For small tables, run UPDATE STATISTICS HIGH, for example:

   ```
   UPDATE STATISTICS HIGH FOR TABLE t2;
   ```

Because the statement constructs the statistics only once for each index, these steps ensure that UPDATE STATISTICS executes rapidly.

Examples

Example of UPDATE STATISTICS HIGH statements for all columns that head an index

Suppose you have a table **t1** with columns **a**, **b**, **c**, **d**, **e**, and **f** with the following indexes:

```
CREATE INDEX ix_1 ON t1 (a, b, c, d) ...
CREATE INDEX ix_3 ON t1 (f) ...
```

Run the following UPDATE STATISTICS statements for the columns that head an index:

```
UPDATE STATISTICS HIGH FOR TABLE t1(a);
UPDATE STATISTICS HIGH FOR TABLE t1(f);
```

These UPDATE STATISTICS HIGH statements replace the distributions created with the UPDATE STATISTICS MEDIUM statements with high distributions for index columns.

Example of UPDATE STATISTICS HIGH statements for the first column in each index that differs:

For example, suppose you have the following indexes on table **t1:**

```
CREATE INDEX ix_1 ON t1 (a, b, c, d) ...
CREATE INDEX ix_2 ON t1 (a, b, e, f) ...
CREATE INDEX ix_3 ON t1 (f) ...
```

Step 3 executes UPDATE STATISTICS HIGH on column **a** and column **f**. Then run UPDATE STATISTICS HIGH on columns **c** and **e**.

```
UPDATE STATISTICS HIGH FOR TABLE t1(c);
UPDATE STATISTICS HIGH FOR TABLE t1(e);
```

In addition, you can run UPDATE STATISTICS HIGH on column **b**, although this is usually not necessary.

Related concepts:
Virtual portion of shared memory
Related information:
UPDATE STATISTICS statement

Updating statistics for join columns

In some situations, you might want to run the UPDATE STATISTICS statement with the HIGH keyword for specific join columns.

Because of improvements and adjusted cost estimates to establish better query plans, the optimizer depends greatly on an accurate understanding of the underlying data distributions in certain cases. You might still think that a complex query does not execute quickly enough, even though you followed the guidelines in Creating data distributions. If your query involves equality predicates, take one of the following actions:

- Run the UPDATE STATISTICS statement with the HIGH keyword for specific join columns that appear in the WHERE clause of the query. If you followed the guidelines in Creating data distributions, columns that head indexes already have HIGH mode distributions.
- Determine whether HIGH mode distribution information about columns that do not head indexes can provide a better execution path, take the following steps:

To determine if UPDATE STATISTICS HIGH on join columns might make a difference:

1. Issue the SET EXPLAIN ON statement and rerun the query.
2. Note the estimated number of rows in the SET EXPLAIN output and the actual number of rows that the query returns.
3. If these two numbers are significantly different, run UPDATE STATISTICS HIGH on the columns that participate in joins, unless you have already done so.

Important: If your table is very large, UPDATE STATISTICS with the HIGH mode can take a long time to execute.
The following example shows a query that involves join columns:

```
SELECT employee.name, address.city
   FROM employee, address
   WHERE employee.ssn = address.ssn
   AND employee.name = 'James'
```

In this example, the join columns are the **ssn** fields in the **employee** and **address** tables. The data distributions for both of these columns must accurately reflect the actual data so that the optimizer can correctly determine the best join plan and execution order.

You cannot use the UPDATE STATISTICS statement to create data distributions for a table that is external to the current database. For additional information about data distributions and the UPDATE STATISTICS statement, see the *IBM® Informix® Guide to SQL: Syntax*.

Updating statistics for columns with user-defined data types

Programmers can write functions that gather statistics for columns with user-defined data types. You can store the data distributions for user-defined data types in an sbspace.

Because information about the nature and use of a user-defined data type (UDT) is not available to the database server, it cannot collect the **colmin** and **colmax** column of the **syscolumns** system catalog table for user-defined data types. To gather statistics for columns with user-defined data types, programmers must write functions that extend the UPDATE STATISTICS statement. For more information, see the performance chapter in *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

Because the data distributions for user-defined data types can be large, you can optionally store them in an sbspace instead of the **sysdistrib** system catalog table.

To store data distributions for user-defined data types in an sbspace:

1. Use the **onspaces -c -S** command to create an sbspace.
   To ensure recoverability of the data distributions, specify `LOGGING=ON` in the **-Df** option, as the following sample shows:

   ```
   % onspaces -c -S distrib_sbsp -p /dev/raw_dev1 -o 500 -s
         20000
     -m /dev/raw_dev2 500 -Ms 150 -Mo 200 -Df
         "AVG_LO_SIZE=32,LOGGING=ON"
   ```

   For information about sizing an sbspace, see Estimating pages that smart large objects occupy.

   For more information about specifying storage characteristics for sbspaces, see Configuration parameters that affect sbspace I/O.

2. Specify the sbspace that you created in step 1 in the configuration parameter SYSSBSPACENAME.
3. Specify the column with the user-defined data type when you run the UPDATE STATISTICS statement with the MEDIUM or HIGH keywords to generate data distributions.

To print the data distributions for a column with a user-defined data type, use the **dbschema -hd** option.

Update statistics in parallel on very large databases

If you have an extremely large database and indexes are fragmented, UPDATE STATISTICS LOW can automatically run statements in parallel.

To enable statements to automatically run in parallel, you must run UPDATE STATISTICS LOW with the PDQ priority set to a value that is between `1` and `10`. If the PDQ priority is set to `1`, 10 percent of the index fragments are analyzed at one time for the current table. If the PDQ priority is set to `5`, 50 percent of the index fragments are analyzed at one time for the current table. If the PDQ priority is set to `10`, all fragments are analyzed at one time for the current table. (If the PDQ priority is set to a value that is higher than `10`, Informix® operates as if the PDQ priority is set to `10`, analyzing all fragments at one time for the current table.)

If you run UPDATE STATISTICS MEDIUM or HIGH, you can set the PDQ priority to a value that is higher than `10`. Because the UPDATE STATISTICS MEDIUM and HIGH statements perform a large amount of sorting operations, increasing the PDQ priority to a value that is higher than `10` provides additional memory than can improve the speed of the sorting operations.

Adjust the amount of memory and disk space for UPDATE STATISTICS

When you execute the UPDATE STATISTICS statement, the database server uses memory and disk to sort and construct data distributions. You can affect the amount of memory and disk space available for UPDATE STATISTICS operations.

You can affect the amount of memory and disk space available for UPDATE STATISTICS with the following methods:

- PDQ priority
  You can obtain more memory for sorting when you set PDQ priority greater than 0. The default value for PDQ priority is `0`. To set PDQ priority, use either the **PDQPRIORITY** environment variable or the SQL statement SET PDQPRIORITY.

  For more information about PDQ priority, see The allocation of resources for parallel database queries.

- **DBUPSPACE** environment variable
  You can use the **DBUPSPACE** environment variable to specify the amount of system disk space (and the amount of memory for sorting values) that UPDATE STATISTICS MEDIUM or UPDATE STATISTICS HIGH statements can use in each pass to construct column distributions. If you specify too small a value, the database server instead uses enough space to write the largest column to disk.

  For more information about this environment variable, see the *IBM® Informix® Guide to SQL: Reference*.

Data sampling during update statistics operations

If you have a large b-tree index with more than 100 K leaf pages, you can generate index statistics based on sampling when you run UPDATE STATISTICS statements in LOW mode. The gathering of statistics through sampling can increase the speed of the update statistics operation.

By default, when UPDATE STATISTICS statements run, the database server reads all index leaf pages in sequence to gather statistics such as the number of leaf pages, the number of unique lead key values, and cluster information. For a large index this can take a long time. With sampling, the database server reads a fraction of the index leaf pages (the sample) and then deduces index statistics based on statistics gathered from the sample.

A possible trade-off for less time in gathering statistics is the accuracy of the statistics gathered. If there are significant skews in the data distribution for the lead index key, the sampling approach can result in a large error margin for the statistics gathered, which in turn might affect optimizer decisions in query plan generation.

You cannot control how much data is in the sample.

To enable or disable sampling, use the USTLOW_SAMPLE configuration parameter or the USTLOW_SAMPLE environment option of the SET ENVIRONMENT statement.

Related information:
USTLOW_SAMPLE configuration parameter
USTLOW_SAMPLE environment option

Display data distributions

You can use the **dbschema** utility to display data distributions.

Unless column values change considerably, you do not need to regenerate the data distributions. To verify the accuracy of the distribution, compare **dbschema -hd** output with the results of appropriately constructed SELECT statements.

For example, the following **dbschema** command produces a list of distributions for each column of table **customer** in database **vjp_stores** with the number of values in each bin, and the number of distinct values:

```
dbschema -hd customer -d vjp_stores
```

Figure 1 shows the data distributions for the column **zipcode** that this **dbschema -hd** command produces. Because this column heads the **zip_ix** index, UPDATE STATISTICS HIGH was run on it, as the following output line indicates:

```
High Mode, 0.500000 Resolution
```

Figure 1 shows 17 bins with one distinct **zipcode** value in each bin.
Figure 1. Displaying Data Distributions with dbschema -hd

```
dbschema -hd customer -d vjp_stores

...
Distribution for virginia.customer.zipcode

Constructed on 09/18/2000

High Mode, 0.500000 Resolution

--- DISTRIBUTION ---

     (          02135 )
  1: (  1,   1, 02135 )
  2: (  1,   1, 08002 )
  3: (  1,   1, 08540 )
  4: (  1,   1, 19898 )
  5: (  1,   1, 32256 )
  6: (  1,   1, 60406 )
  7: (  1,   1, 74006 )
  8: (  1,   1, 80219 )
  9: (  1,   1, 85008 )
 10: (  1,   1, 85016 )
 11: (  1,   1, 94026 )
 12: (  1,   1, 94040 )
 13: (  1,   1, 94085 )
 14: (  1,   1, 94117 )
 15: (  1,   1, 94303 )
 16: (  1,   1, 94304 )
 17: (  1,   1, 94609 )


--- OVERFLOW ---

  1: (  2,      94022 )
  2: (  2,      94025 )
  3: (  2,      94062 )
  4: (  3,      94063 )
  5: (  2,      94086 )
```

The OVERFLOW portion of the output shows the duplicate values that might skew the distribution data, so **dbschema** moves them from the distribution to a separate list. The number of duplicates in this overflow list must be greater than a critical amount that the following formula determines. Figure 1 shows a resolution value of `.0050`. Therefore, this formula determines that any value that is duplicated more than one time is listed in the overflow section.

```
Overflow = .25 * resolution * number_rows
      = .25 * .0050 * 28
      = .035
```

For more information about the **dbschema** utility, see the *IBM® Informix® Migration Guide*.

Improve performance by adding or removing indexes

You can often improve the performance of a query by adding or, in some cases, removing indexes. You can also enable the optimizer to automatically fetch a set of keys from an index buffer.

To improve the performance of a query, consider using some of the methods that the following topics describe.

In addition:

- Consider using the CREATE INDEX ONLINE and DROP INDEX ONLINE statements to create and drop an index in an online environment, when the database and its associated tables are continuously available. These SQL statements enable you to create and drop indexes without having an access lock placed over the table during the duration of the index builds or drops. For more information, see Creating and dropping an index in an online environment.
- Set the BATCHEDREAD_INDEX configuration parameter to enable the optimizer to automatically fetch a set of keys from an index buffer. This reduces the number of times a buffer is read.

- Replace autoindexes with permanent indexes
  If the query plan includes an *autoindex* path to a large table, you can generally improve performance by adding an index on that column. If you perform a query regularly, you can save time by creating a permanent index.
- Use composite indexes
  The optimizer can use a composite index (one that covers more than one column) in several ways.
- Indexes for data warehouse applications
  Many data warehouse databases use a *star schema*, which consists of a *fact* table and a number of *dimensional* tables. Queries that use tables in a star schema or snowflake schema can benefit from the proper index on the fact table.
- Configure B-tree scanner information to improve transaction processing
  You can improve the performance of transaction processing in logged databases by controlling how the B-tree scanner threads remove deletions from indexes.
- Determine the amount of free space in an index page
  You can use the **oncheck -pT** command to determine the amount of free space in each index page.

Related information:
BATCHEDREAD_INDEX configuration parameter

Replace autoindexes with permanent indexes

If the query plan includes an *autoindex* path to a large table, you can generally improve performance by adding an index on that column. If you perform a query regularly, you can save time by creating a permanent index.

If you perform the query occasionally, you can reasonably let the database server build and discard an index.

Use composite indexes

The optimizer can use a composite index (one that covers more than one column) in several ways.

The database server can use an index on columns **a**, **b**, and **c** (in that order) in the following ways:

- To locate a particular row
  The database server can use a composite index when the first filter is an equality filter and subsequent columns have range (<, <=, >, >=) expressions. The following examples of filters use the columns in a composite index:

  ```
  WHERE a=1
  WHERE a>=12 AND a<15
  WHERE a=1 AND b < 5
  WHERE a=1 AND b = 17 AND c >= 40
  ```

  The following examples of filters cannot use that composite index:

  ```
  WHERE b=10
  WHERE c=221
  WHERE a>=12 AND b=15
  ```

- To replace a table scan when all of the desired columns are contained within the index
  A scan that uses the index but does not reference the table is called a *key-only search*.

- To join column **a**, columns **ab**, or columns **abc** to another table
- To implement ORDER BY or GROUP BY on columns **a**, **ab**, or **abc** but not on **b**, **c**, **ac**, or **bc**

Execution is most efficient when you create a composite index with the columns in order from most to least distinct. In other words, the column that returns the highest count of distinct rows when queried with the DISTINCT keyword in the SELECT statement should come first in the composite index.

If your application performs several long queries, each of which contains ORDER BY or GROUP BY clauses, you can sometimes improve performance by adding indexes that produce these orderings without requiring a sort. For example, the following query sorts each column in the ORDER BY clause in a different direction:

```
SELECT * FROM t1 ORDER BY a, b DESC;
```

To avoid using temporary tables to sort column **a** in ascending order and column **b** in descending order, you must create a composite index on (**a**, **b** DESC) or on (**a** DESC, **b**). You need to create only one of these indexes because of the bidirectional-traverse capability of the database server. For more information about bidirectional traverse, see the *IBM® Informix® Guide to SQL: Syntax*.

On the other hand, it can be less expensive to perform a table scan and sort the results instead of using the composite index when the following criteria are met:

- Your table is well ordered relative to your index.
- The number of rows that the query retrieves represents a large percentage of the available data.

Indexes for data warehouse applications

Many data warehouse databases use a *star schema*, which consists of a *fact* table and a number of *dimensional* tables. Queries that use tables in a star schema or snowflake schema can benefit from the proper index on the fact table.

The fact table is generally large and contains the quantitative or factual information about the subject. A dimensional table describes an attribute in the fact table.

When a dimension needs lower-level information, the dimension is modeled by a hierarchy of tables, called a *snowflake schema*.

Consider the example of a star schema with one fact table named **orders** and four dimensional tables named **customers**, **suppliers**, **products**, and **clerks**. The **orders** table describes the details of each sale order, which includes the customer ID, supplier ID, product ID, and sales clerk ID. Each dimensional table describes an ID in detail. The **orders** table is large, and the four dimensional tables are small.

The following query finds the total direct sales revenue in the Menlo Park region (postal code 94025) for hard drives supplied by the Johnson supplier:

```
SELECT sum(orders.price)
FROM orders, customers, suppliers,product,clerks
WHERE orders.custid = customers.custid
   AND customers.zipcode = 94025
   AND orders.suppid = suppliers.suppid
   AND suppliers.name = 'Johnson'
   AND orders.prodid = product.prodid
   AND product.type = 'hard drive'
   AND orders.clerkid = clerks.clerkid
   AND clerks.dept = 'Direct Sales'
```

This query uses a typical star join, in which the fact table joins with all dimensional tables on a foreign key. Each dimensional table has a selective table filter.

An optimal plan for the star join is to perform a cartesian product on the four dimensional tables and then join the result with the fact table. The following index on the fact table allows the optimizer to choose the optimal query plan:

```
CREATE INDEX ON orders(custid,suppid,prodid,clerkid)
```

Without this index, the optimizer might choose to first join the fact table with a single dimensional table and then join the result with the remaining dimensional tables. The optimal plan provides better performance.

For more information about star schemas and snowflake schemas, see the *IBM® Informix® Database Design and Implementation Guide*.

Configure B-tree scanner information to improve transaction processing

You can improve the performance of transaction processing in logged databases by controlling how the B-tree scanner threads remove deletions from indexes.

The B-tree scanner improves transaction processing for logged databases when rows are deleted from a table with indexes. The B-tree scanner automatically determines which index partitions will be cleaned, based on a priority list. B-tree scanner threads remove deleted index entries and rebalance the index nodes. The B-tree scanner automatically determines which index items are to be deleted.

In a logged database, when a delete or an update operation is performed on a row, any corresponding index entry is not immediately deleted. Instead, the corresponding index entry is flagged as deleted until a B-tree scanner thread scans the index and removes the deleted items. An index containing many deleted items can cause a significant performance problem, because index searches need to scan a larger number of items before finding the first valid item.

The default setting for B-tree scanning provides the following type of scanning, depending on your indexes:

- If the table has more than one attached index, the B-tree scanner uses the leaf scan mode. Leaf scan mode is the only type of scanning possible with multiple attached indexes.
- If the table contains a single attached index or if the indexes are detached, the B-tree scanner uses alice (adaptive linear index cleaning) mode. The initial alice scan mode is optimized for small- to medium-sized systems with few or no indexes above 1 GB. However, if the database server detects that the alice mode is inefficient, the alice scan mode setting is adjusted automatically to accommodate larger indexes.

Depending on your application and the order in which the system adds and deletes keys from the index, the structure of an index can become inefficient.

You use the BTSCANNER configuration parameter to specify the following information, which defines the scan mode:

- The number of B-tree scanner threads to start when the database server starts
  The number of B-tree scanner threads is configurable to any positive number. One B-tree scanner thread will always clean an individual index partition, so if you occasionally or consistently have a higher number of index partitions requiring cleaning, you might want to use more than one B-tree scanner thread. At runtime, you can turn off any B-tree scanner activity by issuing an **onmode -C** command. This command stops all B-tree scanner threads.
- The threshold, which is the minimum number of deleted items an index must encounter before an index is placed on the priority list for eligibility for scanning and cleaning by a B-tree scanner thread
  For example, if you increase the threshold beyond 5000, you might be able to avoid frequent B-tree scanner activity on the indexes that receive the most updates and deletes.
- The range size, in kilobytes, that an index or index fragment must exceed before the index is cleaned with range scanning
- An alice mode value
- The level at which B-tree scanner threads compress indexes by merging two partially used index pages

The server treats a forest of trees index the same way it treats a B-tree index. Therefore, in a logged database, you can control how the B-tree scanner threads remove deletions from both forest of trees and B-tree indexes.

The following table summarizes the differences between the scan modes.

Table 1. Scan modes for B-tree scanner threads

Scan Mode	Description	Performance Advantages or Issues	More Information
Leaf scan mode	In this mode, the leaf level of an attached index is completely scanned for deleted items.	This mode is only possible on attached indexes and is the only mode the server can use if more than one attached index exists in a partition.	Leaf and range scan mode settings
Alice (adaptive linear index cleaning) scan mode	If the BTSCANNER alice option is enabled, every index partition receives a bitmap that tracks where a deleted item was found in the index. The scan that occurs excludes all parts of the index where no delete operations are found. The initial size and granularity of these bitmaps depend on the size of the partitions they represent and the current system-wide alice level. The server periodically checks each bitmap for its efficiency by checking the ratio of pages to be cleaned to pages read, adjusting scanning if necessary to get better information. This mode allocates additional resources (memory) to the index that is consuming excess I/O.	You can greatly improve performance and reduce I/O when using the alice mode. Generally, alice mode is 64 times more efficient than range scanning and can automatically tune itself for unsatisfactory indexes, which range scanning cannot do.	Alice scan mode values
Range scan mode	Range scanning, which is enabled with the `rangesize` option, is performed in the range between the low and high page address. The leaf level of the index partition is only scanned within this range. The server performs light scans, which do not immediately use and strain the buffer pool, even though cleaning occurs through the buffer pool.	Not recommended for Informix® Version 11.10 or higher. Alice scanning is exactly the same as range scanning, but is 64 times more efficient, uses the same resources, and has 128 equal ranges. When you set alice mode scanning, range scanning does not have an effect. If you decide to use range scanning for systems with only a lot of large indexes, set the `rangesize` option to the minimum partition size for range scanning.	Leaf and range scan mode settings

For more information about the BTSCANNER configuration parameter and for more information about how the database server maintains an index tree, see the chapter on configuration parameters and the chapter on disk structure and storage in the *IBM® Informix Administrator's Reference*.

Use the **onstat -C** option to monitor the B-tree scanner activities.

Use the **onmode -C** option to change the configuration of B-tree scanners during runtime.

For more information about **onstat -C** and **onmode -C**, see the *IBM Informix Administrator's Reference*.

- Alice scan mode values
  You enable alice (adaptive linear index cleaning) mode by setting the `alice` option to any value between `1` and `12` (finest initial granularity). For small- to medium-sized systems with few or no indexes above 1 gigabyte, set the `alice` option to `6` or `7`. For systems with large indexes, set alice to a higher mode.
- Leaf and range scan mode settings
  If a table has more than one attached index, the B-tree scanner uses the leaf scan mode. If you want small indexes to be scanned by the leaf scan method, set the `rangesize` option of the BTSCANNER configuration parameter to `100`.
- B-tree scanner index compression levels and transaction processing performance
  B-tree scanner threads compress indexes by merging two partially used index pages if the amount of data on those pages is below the level that is specified by the compression option. You can set the compression level to control the amount of I/O required to find and load data.
- Setting the level for B-tree scanner compression of indexes
  Informix provides several ways to specify the level at which B-tree scanner threads will compress indexes pages. To optimize space and transaction processing, you can lower the compression level if your indexes grow quickly. You can increase the level if your indexes have few delete and insert operations or if batch updates are performed.

Alice scan mode values

You enable alice (adaptive linear index cleaning) mode by setting the `alice` option to any value between `1` and `12` (finest initial granularity). For small- to medium-sized systems with few or no indexes above 1 gigabyte, set the `alice` option to `6` or `7`. For systems with large indexes, set alice to a higher mode.

When you set alice mode, the higher the mode, the more memory is used per index partition. However, the memory used is not a huge amount. The advantage is less I/O, as shown in the following table.

Table 1. Alice mode settings

Alice Mode Setting	Memory or Block I/O
0	Turns off alice scanning.
1	Uses exactly 8 bytes of memory (no adjusting).

Alice Mode Setting	Memory or Block I/O
2	Uses exactly 16 bytes of memory (no adjusting).
3	Each block of pages will need 512 I/O operations for cleaning.
4	Each block of pages will need 256 I/O operations for cleaning.
5	Each block of pages will need 128 I/O operations for cleaning.
6 (default)	Each block of pages will need 64 I/O operations for cleaning.
7	Each block of pages will need 32 I/O operations for cleaning.
8	Each block of pages will need 16 I/O operations for cleaning.
9	Each block of pages will need 8 I/O operations for cleaning.
10	Each block of pages will need 4 I/O operations for cleaning.
11	Each block of pages will need 2 I/O operations for cleaning.
12	Each block of pages will need 1 I/O operations for cleaning.

When you set the alice mode, you need to consider memory usage versus I/O. The lower the alice mode setting, the less memory the index will use. The higher the alice mode setting, the more memory the index will use. `12` is the highest mode value, because it is a direct mapping of a single bit of memory to each instance of I/O.

Suppose you have an online page size of 2 KB and the default B-Tree Scanner I/O size of 256 pages. If you set the alice mode to `6`, each byte of memory can represent 131,072 index pages (256 MB). If you set the mode to `10`, each byte of memory can represent 8,192 index pages (16 MB). Thus, changing the mode setting from `6` to `10` requests 16 times the memory, but requires 16 times less I/O.

If you have an index partition that uses 1 GB, then an alice mode setting of `6` would take 4 bytes of memory, while an alice mode setting of `10` would consume 64 bytes of memory, as shown in this formula:

```
( {mode block size} io per bit  *   8 bits per byte  * 256 page per io )
```

Setting the alice mode to a value between 3 and 12 sets the initial amount of memory that is used for index cleaning. Subsequently, the B-tree scanners automatically adjust the mode based on the efficiency of past cleaning operations.

For example, if after five scans (by default), the I/O efficiency is below 75 percent, the server automatically adjusts to the next alice mode if you set the mode to a value above `2`. For example, if an index is currently operating in alice mode `6`, a B-tree scanner has cleaned the index at least 5 times, and the I/O efficiency is below 75 percent, the server automatically adjusts to mode `7`, the next higher mode. This doubles the memory required, but reduces the I/O by a factor of 2.

The server will re-evaluate the index after five more scans to determine the I/O efficiency again, and will continue to do this until mode `12`. The server stops making adjustments at mode `12`.

The following example sets the alice mode to `6`:

```
BTSCANNER num=2,threshold=10000,alice=6,compression=default
```

Related concepts:
Leaf and range scan mode settings
B-tree scanner index compression levels and transaction processing performance
Related tasks:
Setting the level for B-tree scanner compression of indexes

Leaf and range scan mode settings

If a table has more than one attached index, the B-tree scanner uses the leaf scan mode. If you want small indexes to be scanned by the leaf scan method, set the `rangesize` option of the BTSCANNER configuration parameter to `100`.

If you decide to enable range scan mode when a single index exists in the partition, set `rangesize` option of the BTSCANNER configuration parameter to the minimum size that a partition must have to be scanned using this mode. Specify the size in kilobytes.

The following example specifies that:

- The server will start two B-tree scanner threads.
- The server will consider cleaning indexes in the hot list (a list of indexes that caused the server to do extra work) when 50000 deleted items are found in the index.
- Indexes with a partition size that is equal to or larger than 100 KB will be cleaned using the range scan mode.
- Indexes with a partition size of less than 100 KB will be cleaned using the leaf scan mode.
- Index compression is set at the medium (default) level

```
BTSCANNER num=2,threshold=50000,rangesize=100,compression=default
```

Related concepts:
Alice scan mode values
B-tree scanner index compression levels and transaction processing performance
Related tasks:
Setting the level for B-tree scanner compression of indexes

B-tree scanner index compression levels and transaction processing performance

B-tree scanner threads compress indexes by merging two partially used index pages if the amount of data on those pages is below the level that is specified by the compression option. You can set the compression level to control the amount of I/O required to find and load data.

B-tree scanner threads look for index pages that can be compressed because they are below the specified level. The B-tree scanner can compress index pages with deleted items and pages that do not have deleted items.

By default, a B-tree scanner compresses at the medium level. The following table provides information about the performance benefits and trade-offs if you change the compression level to high or low.

Table 1. B-Tree Scanner Compression Level Benefits and Trade-offs

Compression Level	Performance Benefits and Trade-offs	When to Use
Low	The low compression level is beneficial for an index that is expected to grow quickly, with frequent B-tree node splits. When the compression level is set to low, the B-tree index will not require as many splits as indexes with medium or high compression levels, because more free space remains in the B-tree nodes.	You might want to change the compression level to low if you expect an index to grow quickly with frequent splits.
High	In general, if an index is read-only or 90 percent of it is read-only, the high compression level is beneficial because searching for data will require fewer pages (and less I/O) to traverse. Examples might be indexes that do not have frequent changes or indexes undergoing batch (block) delete operations. Using high level of compression also means a performance trade-off, because it takes more I/O to compress the index more aggressively. Select operations will have less I/O when the compression level is high.	You might want to change the compression level to high under these circumstances: • If an index is read most of the time, and delete and insert operations occur a small percentage of the time. • If tables are loaded and updated in a batch and are kept for a period of time as read-only tables.

If you do not need to change the compression level to high or low, set the compression option of the BTSCANNER configuration parameter to `med` or `default`.

Index Compression and the Index Fill Factor

In addition to the compression option that specifies when to attempt to join two partially used pages, you can use the FILL FACTOR configuration parameter to control when to add new index pages. The index fill factor, which you define with the FILLFACTOR configuration parameter or the FILLFACTOR option of the CREATE INDEX statement, is a percentage of each index page that will be filled during the index build.

Related concepts:
Alice scan mode values
Leaf and range scan mode settings
Related tasks:
Setting the level for B-tree scanner compression of indexes

Setting the level for B-tree scanner compression of indexes

Informix® provides several ways to specify the level at which B-tree scanner threads will compress indexes pages. To optimize space and transaction processing, you can lower the compression level if your indexes grow quickly. You can increase the level if your indexes have few delete and insert operations or if batch updates are performed.

Prerequisites:

- Determine if adjusting the level for index compression will improve performance.
- Get statistics on the number of rows read, deleted, and inserted by running the **onstat -g ppf** command. You can also view information in the **sysptprof** table.
- Analyze the statistics to determine if you want to change the threshold.

For information about compression levels and the circumstances under which you might want to change the level, see B-tree scanner index compression levels and transaction processing performance.
Specify the compression level for an instance with any of the following options:

- Set the `compression` field of the BTSCANNER configuration parameter to `low`, `med` (medium), `high`, or `default`. (The system default value is `med`.)
- Run the **onmode -C compression *value*** command, where *value* is `low`, `med` (medium), `high`, and `default`. The system default value is `med`.
- Run an SQL administration API function with this command:

```
SET INDEX COMPRESSION, partition number, compression level
```

Examples

Set the compression option of the BTSCANNER configuration parameter to `default` as follows:

```
BTSCANNER num=4,threshold=10000,rangesize=-1,alice=6,compression=default
```

Set the compression option of the BTSCANNER configuration parameter to `high` as follows:

```
BTSCANNER num=4,threshold=5000,compression=high
```

Specify the compression level using **onmode -C**, as follows:

```
onmode -C compression high
```

Run either of the following SQL administration API functions to set the compression level for a single fragment of the index that has the partition number `1048960`:

```
EXECUTE FUNCTION TASK("SET INDEX COMPRESSION", 1048960, "DEFAULT");
```

```
EXECUTE FUNCTION ADMIN("SET INDEX COMPRESSION", 1048960, "LOW");
```

Run the following SELECT statement to execute the task function over all index fragments. This command sets the compression level for all fragments of an index named `idx1` in a database named `db1`.

```
SELECT sysadmin:TASK("SET INDEX COMPRESSION", partnum, "MED")
FROM sysmaster:systabnames
WHERE dbsname = 'dbs1' AND tabname = 'idx1';
```

You can also run the following SELECT TASK statement to execute the task function over all index fragments and set the compression level for all fragments.

```
SELECT TASK("SET INDEX COMPRESSION", partn, "MED")
FROM dbs1:systables t, dbs1:sysfragments f
WHERE f.tabid = t.tabid AND f.fragtype = 'I' AND indexname ='idx1';
```

Related concepts:
Alice scan mode values
Leaf and range scan mode settings
B-tree scanner index compression levels and transaction processing performance

Determine the amount of free space in an index page

You can use the **oncheck -pT** command to determine the amount of free space in each index page.

If your table has relatively low update activity and a large amount of free space exists, you might want to drop and re-create the index with a larger value for FILLFACTOR to make the unused disk space available.

Optimizer estimates of distributed queries

The optimizer assumes that access to a row from a remote database takes longer than access to a row in a local database. The optimizer estimates include the cost of retrieving the row from disk and transmitting it across the network.

For an example of this higher estimated cost, see The query plan of a distributed query.

- Buffer data transfers for a distributed query
  Informix® uses several factors to determine the size of the buffer that sends and receives data to and from a remote server.
- The query plan of a distributed query
  You can display the chosen query plan of a distributed query. The information displayed for a distributed join differs from information displayed for a local join.

Buffer data transfers for a distributed query

Informix® uses several factors to determine the size of the buffer that sends and receives data to and from a remote server.

The server uses the following factors to determine the buffer size:

- The row size
  The database server calculates the row size by summing the average move size (if available) or the length (from the **syscolumns** system catalog table) of the columns.
- The setting of the **FET_BUF_SIZE** environment variable on the client
  You might be able to reduce the size and number of data transfers by using the **FET_BUF_SIZE** environment variable to increase the size of the buffer that the database server uses to send and receive rows to and from the remote database server.

  The minimum buffer size is 1024 or 2048 bytes, depending on the row size. If the row size is larger than either 1024 or 2048 bytes, the database server uses the **FET_BUF_SIZE** value.

  For more information about the **FET_BUF_SIZE** environment variable, see the *IBM® Informix Guide to SQL: Reference.*

The query plan of a distributed query

You can display the chosen query plan of a distributed query. The information displayed for a distributed join differs from information displayed for a local join.

The following figure shows the chosen query plan for the distributed query.
Figure 1. Selected Output of SET EXPLAIN ALL for Distributed Query, Part 3

```
QUERY:
------
select l.customer_num, l.lname, l.company,
         l.phone, r.call_dtime, r.call_descr
       from customer l,  vjp_stores@gilroy:cust_calls r
        where l.customer_num = r.customer_num

 Estimated Cost: 9
Estimated # of Rows Returned: 7

  1) informix.r:  REMOTE PATH

  2) informix.l: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.l.customer_num = informix.r.customer_num
NESTED LOOP JOIN
```

The following table shows the main differences between the chosen query plans for the distributed join and the local join.

Output Line in Figure 1 for Distributed Query	Output Line in Figure 1 for Local-Only Query	Description of Difference
vjp_stores@gilroy: virginia.cust_calls	informix.cust_calls	The remote table name is prefaced with the database and server names.
Estimated Cost: 9	Estimated Cost: 7	The optimizer estimates a higher cost for the distributed query.
informix.r: REMOTE PATH	informix.r: SEQUENTIAL SCAN	The optimizer chose to keep the outer, remote **cust_calls** table at the remote site.
select x0.call_dtime,x0.call_descr,x0. customer_num from vjp_stores:"virginia".cust_ calls x0		The SQL statement that the local database server sends to the remote site. The remote site reoptimizes this statement to choose the actual plan.

Improve sequential scans

You can improve the performance of sequential read operations on large tables by eliminating repeated sequential scans.

Sequential access to a table other than the first table in the plan is ominous because it threatens to read every row of the table once for every row selected from the preceding tables.

If the table is small, it is harmless to read it repeatedly because the table resides completely in memory. Sequential search of an in-memory table can be faster than searching the same table through an index, especially if maintaining those index pages in memory pushes other useful pages out of the buffers.

When the table is larger than a few pages, however, repeated sequential access produces poor performance. One way to prevent this problem is to provide an index to the column that is used to join the table.

Any user with the Resource privilege can build additional indexes. Use the CREATE INDEX statement to make an index.

An index consumes disk space proportional to the width of the key values and the number of rows. (See Estimating index pages.) Also, the database server must update the index whenever rows are inserted, deleted, or updated; the index update slows these operations. If necessary, you can use the DROP INDEX statement to release the index after a series of queries, which frees space and makes table updates easier.

Enable view folding to improve query performance

You can significantly improve the performance of a query that involves a view by enabling view folding.

You do this by setting the IFX_FOLDVIEW configuration parameter to `1`.

When view folding is enabled, views are folded into a parent query. Because the views are folded into the parent query, the query results are not placed in a temporary table.

You can use view folding in the following types of queries:

- Views that contain a UNION ALL clause and the parent query incldues a regular join, Informix® join, ANSI join, or an ORDER BY clause

View folding does not occur for the following types of queries that perform a UNION ALL operation involving a view:

- The view has one of the following clauses: AGGREGATE, GROUP BY, ORDER BY, UNION, DISTINCT, or OUTER JOIN (either Informix or ANSI type).
- The parent query has a UNION or UNION ALL clause.

In these situations, a temporary table is created to hold query results.

Reduce the join and sort operations

After you understand what the query is doing, you can look for ways to obtain the same output with less effort.

The following suggestions can help you rewrite your query more efficiently:

- Avoid or simplify sort operations.
- Use parallel sorts.
- Use temporary tables to reduce sorting scope.

- Avoid or simplify sort operations
  In many situations you can determine how to avoid or reduce frequent or complex sort operations.
- Use parallel sorts
  When you cannot avoid sorting, the database server takes advantage of multiple CPU resources to perform the required sort-and-merge operations in parallel. The database server can use parallel sorts for any query, not just PDQ queries. You can control the number of threads that the database server uses to sort rows.
- Use temporary tables to reduce sorting scope
  You can use a temporary, ordered subset of a table to increase the speed of a query. The temporary table can also simplify the work of the query optimizer, cause the optimizer to avoid multiple-sort operations, and simplify the work of the optimizer in other ways.
- Configuring memory for queries with hash joins, aggregates, and other memory-intensive elements
  Certain configuration parameters can be set to provide more memory for queries that require sorting, hash joins, aggregates, and other memory-intensive elements.

Avoid or simplify sort operations

In many situations you can determine how to avoid or reduce frequent or complex sort operations.

The sort algorithm is highly tuned and extremely efficient. It is as fast as any external sort program that you might apply to the same data. You do not need to avoid infrequent sorts or sorts of relatively small numbers of output rows.

However, you should try to avoid or reduce the scope of repeated sorts of large tables. The optimizer avoids a sort step whenever it can use an index to produce the output in its proper order automatically. The following factors prevent the optimizer from using an index:

- One or more of the ordered columns is not included in the index.
- The columns are named in a different sequence in the index and the ORDER BY or GROUP BY clause.
- The ordered columns are taken from different tables.

For another way to avoid sorts, see Use temporary tables to reduce sorting scope.

If a sort is necessary, look for ways to simplify it. As discussed in Sort-time costs, the sort is quicker if you can sort on fewer or narrower columns.

Related concepts:
Ordering with fragmented indexes

Use parallel sorts

When you cannot avoid sorting, the database server takes advantage of multiple CPU resources to perform the required sort-and-merge operations in parallel. The database server can use parallel sorts for any query, not just PDQ queries. You can control the number of threads that the database server uses to sort rows.

To control the number of threads that the database server uses to sort rows, use the **PSORT_NPROCS** environment variable.

When PDQ priority is greater than 0 and **PSORT_NPROCS** is greater than 1, the query benefits both from parallel sorts and from PDQ features such as parallel scans and additional memory. Users can use the **PDQPRIORITY** environment variable to request a specific proportion of PDQ resources for a query. You can use the MAX_PDQPRIORITY configuration parameter to limit the number of such user requests. For more information, see Limiting PDQ resources in queries.

In some cases, the amount of data being sorted can overflow the memory resources allocated to the query, resulting in I/O to a dbspace or sort file. For more information, see Configure dbspaces for temporary tables and sort files.

Use temporary tables to reduce sorting scope

You can use a temporary, ordered subset of a table to increase the speed of a query. The temporary table can also simplify the work of the query optimizer, cause the optimizer to avoid multiple-sort operations, and simplify the work of the optimizer in other ways.

For example, suppose your application produces a series of reports on customers who have outstanding balances, one report for each major postal area, ordered by customer name. In other words, a series of queries occurs, each of the following form (using hypothetical table and column names):

```
SELECT cust.name, rcvbles.balance, ...other columns...
   FROM cust, rcvbles
   WHERE cust.customer_id = rcvbles.customer_id
      AND rcvbls.balance > 0
      AND cust.postcode LIKE '98_ _ _'
   ORDER BY cust.name
```

This query reads the entire **cust** table. For every row with the specified postal code, the database server searches the index on **rcvbles.customer_id** and performs a nonsequential disk access for every match. The rows are written to a temporary file and sorted. For more information about temporary files, see Configure dbspaces for temporary tables and sort files.

This procedure is acceptable if the query is performed only once, but this example includes a series of queries, each incurring the same amount of work.

An alternative is to select all customers with outstanding balances into a temporary table, ordered by customer name, as the following example shows:

```
SELECT cust.name, rcvbles.balance, ...other columns...
   FROM cust, rcvbles
   WHERE cust.customer_id = rcvbles.customer_id
      AND cvbls.balance > 0
   INTO TEMP cust_with_balance
```

You can then execute queries against the temporary table, as the following example shows:

```
SELECT *
   FROM cust_with_balance
   WHERE postcode LIKE '98_ _ _'
   ORDER BY cust.name
```

Each query reads the temporary table sequentially, but the table has fewer rows than the primary table.

Configuring memory for queries with hash joins, aggregates, and other memory-intensive elements

Certain configuration parameters can be set to provide more memory for queries that require sorting, hash joins, aggregates, and other memory-intensive elements.

How you configure the amount of memory that is available for a query depends on whether or not the query is a Parallel Database Query (PDQ).

Configuring memory for non-PDQ queries

If the PDQ priority is set to `0` (zero), you can change the amount of memory that is available for a query that is not a PDQ query by changing the setting of the DS_NONPDQ_QUERY_MEM configuration parameter. You can only use this parameter if the PDQ priority is set to zero. Its setting has no effect if the PDQ priority is greater than zero.

You can also change the value of DS_NONPDQ_QUERY_MEM with an **onmode -wm** or **onmode -wf** command.

For example, if you use the **onmode** utility, specify a value as shown in the following example:

```
onmode -wf DS_NONPDQ_QUERY_MEM=500
```

The minimum value for DS_NONPDQ_QUERY_MEM is 128 kilobytes. The maximum supported value is 25 percent of DS_TOTAL_MEMORY. 128 kilobytes is the default value of DS_NONPDQ_QUERY_MEM. If you specify a value for the DS_NONPDQ_QUERY_MEM parameter, determine and adjust the value based on the number and size of table rows involved in the query.

Informix® might recalculate the value of DS_NONPDQ_QUERY_MEM initialization if the value is more than 25 percent of the DS_TOTAL_MEMORY value.

If Informix changes the value that you set, the server sends a message in this format:

```
DS_NONPDQ_QUERY_MEM recalculated and changed from old_value Kb to new_value Kb.
```

In the message, `old_value` represents the value that you assigned to DS_NONPDQ_QUERY_MEM in the user configuration file, and `new_value` represents the value determined by Informix.

For formulas for estimating the amount of additional space to allocate for hash joins, see Estimating temporary space for dbspaces and hash joins.

Configuring memory for PDQ queries

The Memory Grant Manager (MGM) component of Informix coordinates the use of memory, CPU virtual processors (VPs), disk I/O, and scan threads among decision-support queries. The MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and MAX_PDQPRIORITY configuration parameter settings to determine the quantity of these PDQ resources that can be granted to a decision-support query. The MGM also grants memory to a query for such activities as hash joins. For more information about the MGM, see The Memory Grant Manager.

Optimize user-response time for queries

You can influence the amount of time that Informix® takes to optimize a query and to return rows to a user.

- Optimization level
  You normally obtain optimum overall performance with the default optimization level, HIGH. The time that it takes to optimize the statement is usually unimportant. However, if experimentation with your application reveals that your query is still taking too long, you can set the optimization level to LOW.
- Optimization goals
  Optimizing total query time and optimizing user-response time are two optimization goals for improving query performance.

Optimization level

You normally obtain optimum overall performance with the default optimization level, HIGH. The time that it takes to optimize the statement is usually unimportant. However, if experimentation with your application reveals that your query is still taking too long, you can set the optimization level to LOW.

If you change the optimization level to LOW, check the SET EXPLAIN output to see if the optimizer chose the same query plan as before.

To specify a HIGH or LOW level of database server optimization, use the SET OPTIMIZATION statement.

Related information:
SET OPTIMIZATION statement

Optimization goals

Optimizing total query time and optimizing user-response time are two optimization goals for improving query performance.

Total query time is the time it takes to return all rows to the application. Total query time is most important for batch processing or for queries that require all rows be processed before returning a result to the user, as in the following query:

```
SELECT count(*) FROM orders
WHERE order_amount > 2000;
```

User-response time is the time that it takes for the database server to return a screen full of rows back to an interactive application. In interactive applications, only a screen full of data can be requested at one time. For example, the user application can display only 10 rows at one time for the following query:

```
SELECT * FROM orders
WHERE order_amount > 2000;
```

Which optimization goal is more important can have an effect on the query path that the optimizer chooses. For example, the optimizer might choose a nested-loop join instead of a hash join to execute a query if user-response time is most important, even though a hash join might result in a reduction in total query time.

- Specifying the query performance goal
  You can optimize user response time for your entire database server system, within a session, or for individual queries.
- Preferred query plans for user-response-time optimization
  When the optimizer chooses query plans to optimize user-response time, it computes the cost for retrieving the first row in the query for each plan and chooses the plan with the lowest cost. In some cases, the query plan with the lowest cost for retrieving the first row might not be the optimal path to retrieve all rows in the query.

Specifying the query performance goal

You can optimize user response time for your entire database server system, within a session, or for individual queries.

The default behavior is for the optimizer to choose query plans that optimize the total query time. You can specify optimization of user-response time at several different levels:

- For the database server system
  To optimize user-response time, set the OPT_GOAL configuration parameter to `0`, as in the following example:

  ```
  OPT_GOAL 0
  ```

  Set OPT_GOAL to `-1` to optimize total query time.

- For the user environment
  The **OPT_GOAL** environment variable can be set before the user application starts.

  UNIX Only
  To optimize user-response time, set the **OPT_GOAL** environment variable to `0`, as in the following sample commands:

  ```
  Bourne shell                OPT_GOAL = 0
                  export OPT_GOAL

  C shell                setenv OPT_GOAL 0
  ```

For total-query-time optimization, set the **OPT_GOAL** environment variable to `-1`.

- Within the session
  You can control the optimization goal with the SET OPTIMIZATION statement in SQL. The optimization goal set with this statement stays in effect until the session ends or until another SET OPTIMIZATION statement changes the goal.

  The following statement causes the optimizer to choose query plans that favor total-query-time optimization:

  ```
  SET OPTIMIZATION ALL_ROWS
  ```

  The following statement causes the optimizer to choose query plans that favor user-response-time optimization:

  ```
  SET OPTIMIZATION FIRST_ROWS
  ```

- For individual queries
  You can use FIRST_ROWS and ALL_ROWS optimizer directives to instruct the optimizer which query goal to use. For more information about these directives, see Optimization-goal directives.

The precedence for these levels is as follows:

- Optimizer directives
- SET OPTIMIZATION statement
- **OPT_GOAL** environment variable
- OPT_GOAL configuration parameter

For example, optimizer directives take precedence over the goal that the SET OPTIMIZATION statement specifies.

Preferred query plans for user-response-time optimization

When the optimizer chooses query plans to optimize user-response time, it computes the cost for retrieving the first row in the query for each plan and chooses the plan with the lowest cost. In some cases, the query plan with the lowest cost for retrieving the first row might not be the optimal path to retrieve all rows in the query.

The following sections explain some of the possible differences in query plans.

- Nested-loop joins versus hash joins
  Hash joins generally have a higher cost to retrieve the first row than nested-loop joins do. The database server must build the hash table before it retrieves any rows. However, in some cases, total query time is faster if the database server uses a hash join.
- Table scans versus index scans
  In cases where the database server returns a large number of rows from a table, the lower-cost option for the total-query-time goal might be to scan the table instead of using an index. However, to retrieve the first row, the lower-cost option for the user-response-time goal might be to use the index to access the table.
- Ordering with fragmented indexes
  When an index is not fragmented, the database server can use the index to avoid a sort. However, when an index is fragmented, the ordering can be guaranteed only within the fragment, not between fragments.

Nested-loop joins versus hash joins

Hash joins generally have a higher cost to retrieve the first row than nested-loop joins do. The database server must build the hash table before it retrieves any rows. However, in some cases, total query time is faster if the database server uses a hash join.

In the following example, **tab2** has an index on **col1**, but **tab1** does not have an index on **col1**. When you execute SET OPTIMIZATION ALL_ROWS before you run the query, the database server uses a hash join and ignores the existing index, as the following portion of SET EXPLAIN output shows:

```
QUERY:
------
SELECT * FROM tab1,tab2
WHERE tab1.col1 = tab2.col1
Estimated Cost: 125
Estimated # of Rows Returned: 510
1) lsuto.tab2: SEQUENTIAL SCAN
2) lsuto.tab1: SEQUENTIAL SCAN
DYNAMIC HASH JOIN
    Dynamic Hash Filters: lsuto.tab2.col1 = lsuto.tab1.col1
```

However, when you execute SET OPTIMIZATION FIRST_ROWS before you run the query, the database server uses a nested-loop join. The clause (FIRST_ROWS OPTIMIZATION) in the following partial SET EXPLAIN output shows that the optimizer used user-response-time optimization for the query:

```
QUERY:        (FIRST_ROWS OPTIMIZATION)
------
SELECT * FROM tab1,tab2
WHERE tab1.col1 = tab2.col1
Estimated Cost: 145
Estimated # of Rows Returned: 510
1) lsuto.tab1: SEQUENTIAL SCAN
2) lsuto.tab2: INDEX PATH
    (1) Index Keys: col1
        Lower Index Filter: lsuto.tab2.col1 = lsuto.tab1.col1
NESTED LOOP JOIN
```

Table scans versus index scans

In cases where the database server returns a large number of rows from a table, the lower-cost option for the total-query-time goal might be to scan the table instead of using an index. However, to retrieve the first row, the lower-cost option for the user-response-time goal might be to use the index to access the table.

Ordering with fragmented indexes

When an index is not fragmented, the database server can use the index to avoid a sort. However, when an index is fragmented, the ordering can be guaranteed only within the fragment, not between fragments.

Usually, the least expensive option for the total-query-time goal is to scan the fragments in parallel and then use the parallel sort to produce the proper ordering. However, this option does not favor the user-response-time goal.

Instead, if the user-response time is more important, the database server reads the index fragments in parallel and merges the data from all of the fragments. No additional sort is generally needed.

Related concepts:
Avoid or simplify sort operations

Optimize queries for user-defined data types

Queries that access user-defined data types (UDTs) can take advantage of the same performance features that built-in data types use.

These features are:

- Indexes
  If a query accesses a small number of rows, an index speeds retrieval because the database server does not need to read every page in a table to find the rows. For more information, see Indexes on user-defined data types.
- Parallel database query (PDQ)
  Queries that access user-defined data can take advantage of parallel scans and parallel execution.

  To turn on parallel execution for a query, set the **PDQPRIORITY** environment variable or use the SQL statement SET PDQPRIORITY. For more information about how to set PDQ priority and configuration parameters that affect PDQ, see The allocation of resources for parallel database queries.
- Optimizer directives

In addition, programmers can write the following functions or UDRs to help the optimizer create an efficient query plan for your queries:

- Parallel UDRs that can take advantage of parallel database queries
- User-defined selectivity functions that calculate the expected fraction of rows that qualify for the function
- User-defined cost functions that calculate the expected relative cost to execute a user-defined routine
- User-defined statistical functions that the UPDATE STATISTICS statement can use to generate statistics and data distributions
- User-defined negator functions to allow more choices for the optimizer

The following sections summarize these techniques. For a more complete description of how to write and register user-defined selectivity functions and user-defined cost functions, see *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

- Parallel UDRs
  One way to execute UDRs is in an expression in a query. You can take advantage of parallel execution if the UDR is in an expression in the query.
- Selectivity and cost functions
  You can use the CREATE FUNCTION statement to create a UDR. Then, you can use routine modifiers to change the cost or selectivity that is specified in the statement.
- User-defined statistics for UDTs
  Because information about the nature and use of a user-defined data type (UDT) is not available to the database server, it cannot collect distributions or the **colmin** and **colmax** information (found in the **syscolumns** system catalog table) for a UDT. Instead, you can create a special function that populates these statistics.
- Negator functions
  A *negator function* takes the same arguments as its companion function, in the same order, but returns the Boolean complement. That is, if a function returns `TRUE` for a given set of arguments, its negator function returns `FALSE` when passed the same arguments, in the same order.

Parallel UDRs

One way to execute UDRs is in an expression in a query. You can take advantage of parallel execution if the UDR is in an expression in the query.

For parallel execution, the UDR must be in one of the following parts of a query:

- WHERE clause
- SELECT list
- GROUP by list
- Overloaded comparison operator
- User-defined aggregate
- HAVING clause
- Select list for a parallel insertion statement
- Generic B-tree index scan on multiple index fragments if the compare function used in the B-tree index scan is parallelizable

For example, suppose that you create an opaque data type **circle**, a table **cir_t** that defines a column of type **circle**, a user-defined routine **area**, and then run the following sample query:

```
SELECT circle, area(circle)
   FROM cir_t
   WHERE circle > "(6,2,4)";
```

In this sample query, the following operations can run in parallel:

- The UDR **area(circle)** in the SELECT list
  If the table **cir_t** is fragmented, multiple **area** UDRs can run in parallel, one UDR on each fragment.
- The expression `circle > "(6,2,4)"` in the WHERE clause
  If the table **cir_t** is fragmented, multiple scans of the table can run in parallel, one scan on each fragment. Then multiple "`>`" comparison operators can run in parallel, one operator per fragment.

By default, a UDR does not run in parallel. To enable parallel execution of UDRs, you must take the following actions:

- Specify the PARALLELIZABLE modifier in the CREATE FUNCTION or ALTER FUNCTION statement.
- Ensure that the UDR does not call functions that are not PDQ thread-safe.
- Turn on PDQ priority.
- Use the UDR in a parallel database query.

Selectivity and cost functions

You can use the CREATE FUNCTION statement to create a UDR. Then, you can use routine modifiers to change the cost or selectivity that is specified in the statement.

After you create a UDR, you can place it in an SQL statement.

The following example shows how you can place a UDR in an SQL statement:

```
SELECT * FROM image
WHERE get_x1(image.im2) and get_x2(image.im1)
```

The optimizer cannot accurately evaluate the cost of executing a UDR without additional information. You can provide the cost and selectivity of the function to the optimizer. The database server uses cost and selectivity together to determine the best path. For more information about selectivity, see Filters with user-defined routines.

In the previous example, the optimizer cannot determine which function to execute first, the **get_x1** function or the **get_x2** function. If a function is expensive to execute, the DBA can assign the function a larger cost or selectivity, which can influence the optimizer to change the query plan for better performance. In the previous example, if **get_x1** costs more to execute, the DBA can assign a higher cost to the function, which can cause the optimizer to execute the **get_x2** function first.

You can add the following routine modifiers to the CREATE FUNCTION statement to change the cost or selectivity that the optimizer assigns to the function:

- **selfunc**=*function_name*
- **selconst**=*integer*
- **costfunc**=*function_name*
- **percall_cost**=*integer*

For more information about cost or selectivity modifiers, see the *IBM® Informix® User-Defined Routines and Data Types Developer's Guide*.

User-defined statistics for UDTs

Because information about the nature and use of a user-defined data type (UDT) is not available to the database server, it cannot collect distributions or the **colmin** and **colmax** information (found in the **syscolumns** system catalog table) for a UDT. Instead, you can create a special function that populates these statistics.

The database server runs the statistics collection function when you execute UPDATE STATISTICS.

For more information about the importance of updating statistics, see Statistics held for the table and index. For information about improving performance, see Updating statistics for columns with user-defined data types.

Negator functions

A *negator function* takes the same arguments as its companion function, in the same order, but returns the Boolean complement. That is, if a function returns `TRUE` for a given set of arguments, its negator function returns `FALSE` when passed the same arguments, in the same order.

In certain cases, the database server can process a query more efficiently if the sense of the query is reversed. That is, "Is **x** greater than **y**?" changes to "Is **y** less than or equal to **x**?"

Optimize queries with the SQL statement cache

Before the database server runs an SQL statement, it must first parse and optimize the statement. Optimizing statements can be time consuming, depending on the size of the SQL statement.

The database server can store the optimized SQL statement in the virtual portion of shared memory, in an area that is called the SQL statement cache. The SQL statement cache (SSC) can be accessed by all users, and it allows users to bypass the optimize step before they run the query. This capability can result in the following significant performance improvements:

- Reduced response times when users are running the same SQL statements.
  SQL statements that take longer to optimize (usually because they include many tables and many filters in the WHERE clause) run faster from the SQL statement cache because the database server does not optimize the statement.
- Reduced memory usage because the database server shares query data structures among users.
  Memory reduction with the SQL statement cache is greater when a statement has many column names in the select list.

For more information about the effect of the SQL statement cache on the performance of the overall system, see Monitor and tune the SQL statement cache.

- When to use the SQL statement cache
  Applications might benefit from use of the SQL statement cache if multiple users execute the same SQL statements. The database server considers statements to be the same if all characters match exactly.
- Using the SQL statement cache
  The DBA usually makes the decision to enable the SQL statement cache. If the SQL statement cache is enabled, individual users can decide whether or not to use the SQL statement cache for their specific environment or application.
- Monitoring memory usage for each session
  You can use several **onstat -g** command options to obtain memory information for each session.
- Monitoring usage of the SQL statement cache
  If you notice a sudden increase in response time for a query that had been using the SQL statement cache, the entry might have been dropped or deleted. You can monitor the usage of the SQL statement cache and check for a dropped or deleted entry by displaying **onstat -g ssc** command output.

When to use the SQL statement cache

Applications might benefit from use of the SQL statement cache if multiple users execute the same SQL statements. The database server considers statements to be the same if all characters match exactly.

For example, if 50 sales representatives execute the **add_order** application throughout the day, they all execute the same SQL statements if the application contains SQL statements that use host variables, such as the following example:

```
SELECT * FROM ORDERS WHERE order_num = :hostvar
```

This kind of application benefits from use of the SQL statement cache because users are likely to find the SQL statements in the SQL statement cache.

The database server does not consider the following SQL statements exact matches because they contain different literal values in the WHERE clause:

```
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
   AND order_date > "01/01/07"
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
   AND order_date > "01/01/2007"
```

Performance does not improve with the SQL statement cache in the following situations:

- If a report application is run once nightly, and it executes SQL statements that no other application uses, it does not benefit from use of the statement cache.
- If an application prepares a statement and then executes it many times, performance does not improve with the SQL statement cache because the statement is optimized just once during the PREPARE statement.

When a statement contains host variables, the database server replaces the host variables with placeholders when it stores the statement in the SQL statement cache. Therefore, the statement is optimized without the database server having access to the values of the host variables. In some cases, if the database server had access to the values of the host variables, the statement might be optimized differently, usually because the distributions stored for a column inform the optimizer exactly how many rows pass the filter.

If an SQL statement that contains host variables performs poorly with the SQL statement cache turned on, try flushing the SQL statement cache with the **onmode -e flush** command and running the query with values that are more frequently used across multiple executions of the query. When you flush the cache, the database server reoptimizes the query and generates a query plan that is optimized for these frequently used values.

Important: The database server flushes an entry from the SQL statement cache only if it is not in use. If an application prepares the statement and keeps it, the entry is still in use. In this case, the application needs to close the statement before the flush is beneficial.

Using the SQL statement cache

The DBA usually makes the decision to enable the SQL statement cache. If the SQL statement cache is enabled, individual users can decide whether or not to use the SQL statement cache for their specific environment or application.

The database server incurs some processing overhead in managing the SQL statement cache, so you should use the SQL statement cache only when multiple users want to share the SQL statements.

To enable the SQL statement cache, set the STMT_CACHE configuration parameter to a value that defines either of the following modes:

- Always use the SQL statement cache unless a user explicitly specifies do not use the cache.
- Use the SQL statement cache only when a user explicitly specifies use it.

For more information, see Enabling the SQL statement cache. For more information about the STMT_CACHE configuration parameter, see the *IBM® Informix® Administrator's Reference*.

- Enabling the SQL statement cache
  The database server does not use the SQL statement cache when the STMT_CACHE configuration parameter is `0` (the default value). You can change this value to enable the SQL statement cache in one of two modes.
- Placing statements in the cache
  SELECT, UPDATE, INSERT and DELETE statements can be placed in the SQL statement cache, with some exceptions. When the database server checks if an SQL statement is in the cache, it must find an exact match.

Enabling the SQL statement cache

The database server does not use the SQL statement cache when the STMT_CACHE configuration parameter is `0` (the default value). You can change this value to enable the SQL statement cache in one of two modes.

Use one of the following methods to change this STMT_CACHE default value:

- Update the ONCONFIG file to specify the STMT_CACHE configuration parameter and restart the database server.
  If you set the STMT_CACHE configuration parameter to `1`, the database server uses the SQL statement cache for an individual user when the user sets the **STMT_CACHE** environment variable to `1` or executes the SET STATEMENT CACHE ON statement within an application.

  ```
  STMT_CACHE 1
  ```

  If the STMT_CACHE configuration parameter is `2`, the database server stores SQL statements for all users in the SQL statement cache except when individual users turn off the feature with the **STMT_CACHE** environment variable or the SET STATEMENT CACHE OFF statement.

  ```
  STMT_CACHE 2
  ```

- Use the **onmode -e** command to override the STMT_CACHE configuration parameter dynamically.
  If you use the **enable** keyword, the database server uses the SQL statement cache for an individual user when the user sets the **STMT_CACHE** environment variable to `1` or executes the SET STATEMENT CACHE ON statement within an application.

  ```
  onmode -e enable
  ```

  If you use the **on** keyword, the database server stores SQL statements for all users in the SQL statement cache except when individual users turn off the feature with the **STMT_CACHE** environment variable or the SET STATEMENT CACHE OFF statement.

  ```
  onmode -e on
  ```

Note: statement cache save and statement cache restore are set to save and restore the SQL statement cache.
The following table summarizes the use of the SQL statement cache, which depends on the setting of the STMT_CACHE configuration parameter (or the execution of **onmode -e**) and the use in an application of the **STMT_CACHE** environment variable and the SET STATEMENT CACHE statement.

STMT_ CACHE Configuration Parameter or onmode -e	STMT_CACHE Environment Variable	SET STATEMENT CACHE Statement	Resulting Behavior
0 (default)	Not applicable	Not applicable	Statement cache not used
1	0 (or not set)	OFF	Statement cache not used
1	1	OFF	Statement cache not used
1	0 (or not set)	ON	Statement cache used
1	1	ON	Statement cache used
1	1	Not executed	Statement cache used
1	0	Not executed	Statement cache not used
2	1 (or not set)	ON	Statement cache used
2	1 (or not set)	OFF	Statement cache not used

STMT_ CACHE Configuration Parameter or onmode -e	STMT_CACHE Environment Variable	SET STATEMENT CACHE Statement	Resulting Behavior
2	0	ON	Statement cache used
2	0	OFF	Statement cache not used by user
2	0	Not executed	Statement cache not used by user
2	1 (or not set)	Not executed	Statement cache used by user

Placing statements in the cache

SELECT, UPDATE, INSERT and DELETE statements can be placed in the SQL statement cache, with some exceptions. When the database server checks if an SQL statement is in the cache, it must find an exact match.

For a complete list of the exceptions and a list of requirements for an exact match, see SET STATEMENT CACHE in the *IBM® Informix® Guide to SQL: Syntax*.

Monitoring memory usage for each session

You can use several **onstat -g** command options to obtain memory information for each session.

You obtain memory information by identifying the SQL statements that use a large amount of memory.

To identify SQL statements using large amount of memory:

1. Run the **onstat -g ses** command to display memory of all sessions and see which session has the highest memory usage.
2. Run the **onstat -g ses** *session-id* command to display more details on the session with the highest memory usage.
3. Run the **onstat -g stm** *session-id* command to display the memory used by the SQL statements.

- Display all user threads and session memory usage
  Use the **onstat -g ses** command to display all user sessions and memory usage by session ID.
- Display detailed session information and memory usage
  Use the **onstat -g ses** *session-id* command to display detailed information for a session, including memory usage.
- Display information about session SQL statements
  Use the **onstat -g sql** *session-id* or **onstat -g spf** commands to display information about the SQL statements executed by a session.
- Display information about the memory that SQL statements use in a session
  Use the **onstat -g stm** *session-id* to display information about the memory each SQL statement uses in a session.

Display all user threads and session memory usage

Use the **onstat -g ses** command to display all user sessions and memory usage by session ID.

When the session shares the memory structures in the SSC, the value in the **used memory** column should be lower than when the cache is turned off. For example, Figure 1 shows sample **onstat -g ses** output when the SQL statement cache is not enabled. Figure 2 shows output after the SQL statement cache is enabled and the queries in Session 4 are run again. Figure 1 shows that Session 4 has 45656 bytes of used memory. Figure 2 shows that Session 4 has less used bytes (36920) when the SQL statement cache is enabled.

Figure 1. **onstat -g ses** output when the SQL statement cache is not enabled

```
session                                          #RSAM    total    used     dynamic
id       user     tty      pid      hostname     threads  memory   memory   explain
12       informix -        0        -            0        12288    7632     off
4        informix 11       5158     smoke        1        53248    45656    off
3        informix -        0        -            0        12288    8872     off
2        informix -        0        -            0        12288    7632     off
```

Figure 2. **onstat -g ses** output when the SQL statement cache is enabled

```
session                                          #RSAM    total    used     dynamic
id       user     tty      pid      hostname     threads  memory   memory   explain
17       informix -        0        -            0        12288    7632     off
16       informix 12       5258     smoke        1        40960    38784    off
4        informix 11       5158     smoke        1        53248    36920    off
3        informix -        0        -            0        12288    8872     off
2        informix -        0        -            0        12288    7632     off
```

Figure 2 also shows the memory allocated and used for Session 16, which runs the same SQL statements as Session 4. Session 16 allocates less total memory (40960) and uses less memory (38784) than Session 4 (Figure 1 shows 53248 and 45656) because Session 16 uses the existing memory structures in the SQL statement cache.

Display detailed session information and memory usage

Use the **onstat -g ses** *session-id* command to display detailed information for a session, including memory usage.

The following **onstat -g ses** *session-id* output columns display memory usage:

- The `Memory pools` portion of the output
  - The **totalsize** column shows the number of bytes currently allocated
  - The **freesize** column shows the number of unallocated bytes
- The last line of the output shows the number of bytes allocated from the sscpool.

Figure 1 shows that Session 16 has currently allocated 69632 bytes, of which 11600 bytes are allocated from the sscpool.
Figure 1. **onstat -g ses** session-id output

```
onstat -g ses 14


session                                          #RSAM    total      used
id       user     tty      pid      hostname    threads  memory     memory
14       virginia 7        28734    lyceum      1        69632      67384

tid      name     rstcb    flags    curstk   status
38       sqlexec  a3974d8  Y--P---  1656     cond wait(netnorm)

Memory pools    count 1
name         class addr      totalsize   freesize  #allocfrag #freefrag
14           V     a974020   69632       2248     156         2

...
Sess  SQL             Current            Iso Lock        SQL  ISAM F.E.
Id    Stmt type       Database           Lvl Mode        ERR  ERR  Vers
14    SELECT          vjp_stores         CR  Not Wait    0    0    9.03

Current statement name : slctcur

Current SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

Last parsed SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

 11600 byte(s) of memory is allocated from the sscpool
```


Display information about session SQL statements

Use the **onstat -g sql** *session-id* or **onstat -g spf** commands to display information about the SQL statements executed by a session.

The following figure shows that **onstat -g sql** *session-id* displays the same information as the bottom portion of the **onstat -g ses** *session-id* command in Figure 1, which includes the number of bytes allocated from the sscpool.
Figure 1. **onstat -g sql** session-id output

```
onstat -g sql 14


Sess  SQL             Current            Iso Lock        SQL  ISAM F.E.
Id    Stmt type       Database           Lvl Mode        ERR  ERR  Vers
14    SELECT          vjp_stores         CR  Not Wait    0    0    9.03

Current statement name : slctcur

Current SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

Last parsed SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

 11600 byte(s) of memory is allocated from the sscpool
```


Display information about the memory that SQL statements use in a session

Use the **onstat -g stm** *session-id* to display information about the memory each SQL statement uses in a session.

The following figure displays the output of **onstat -g stm** *session-id* for the same session (14) as in **onstat -g ses** *session-id* in Figure 1 and **onstat -g sql** *session-id* in Figure 1.

When the SQL statement cache (SSC) is on, the database server creates the heaps in the SSC pool. Therefore, the **heapsz** output field in Figure 1 shows that this SQL statement uses 10056 bytes, which is contained within the 11600 bytes in the SSC pool that the **onstat -g sql 14** shows.

Figure 1. **onstat -g stm** session-id output

```
onstat -g stm 14


session   14 ------------------------------------------------------------------
 sdblock   heapsz   statement ('*' = Open cursor)
 aa11018    10056  *SELECT C.customer_num, O.order_num
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
```


Monitoring usage of the SQL statement cache

If you notice a sudden increase in response time for a query that had been using the SQL statement cache, the entry might have been dropped or deleted. You can monitor the usage of the SQL statement cache and check for a dropped or deleted entry by displaying **onstat -g ssc** command output.

The database server drops an entry from the cache when one of the objects that the query depends on is altered so that it invalidates the data dictionary cache entry for the query. The following operations cause a dependency check failure:

- Execution of any data definition language (DDL) statement (such as ALTER TABLE, DROP INDEX, or CREATE INDEX) that might alter the query plan
- Alteration of a table that is linked to another table with a referential constraint (in either direction)
- Execution of UPDATE STATISTICS FOR TABLE for any table or column involved in the query
- Renaming a column, database, or index with the RENAME statement

When an entry is marked as dropped or deleted, the database server must reparse and reoptimize the SQL statement the next time it executes. For example, Figure 1 shows the entries that the **onstat -g ssc** command displays after UPDATE STATISTICS was executed on the **items** and **orders** table between the execution of the first and second SQL statements.

The `Statement Cache Entries` portion of the **onstat -g ssc** output in Figure 1 displays a **flag** field that indicates whether or not an entry has been dropped or deleted from the SQL statement cache.

- The first entry has a **flag** column with the value `DF`, which indicates that the entry is fully cached, but is now dropped because its entry was invalidated.
- The second entry has the same statement text as the third entry, which indicates that it was reparsed and reoptimized when it was executed after the UPDATE STATISTICS statement.

Figure 1. Sample **onstat -g ssc** command output for a dropped entry

```
onstat -g ssc

...
Statement Cache Entries:

lru hash ref_cnt hits  flag  heap_ptr  database     user
---------------------- ---- ----------------------------------------------------
...
  2  232       1    1    DF   aa3d020  vjp_stores   virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num

  3  232       1    0   -F  aa8b020  vjp_stores   virginia
   SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num
...
```

Invalidating a statement

You can selectively invalidate entries of your choice by setting the sysmaster:syssscelem:valid column to 0 as user Informix

For example, Figure 2 shows the entries that the **onstat -g ssc** command displays before and after invalidating a query from the items table

The `Statement Cache Entries` portion of the **onstat -g ssc** output in Figure 2 displays a **flag** field that indicates whether or not an entry has been invalidated in the SQL statement cache.

Figure 2. Sample **onstat -g ssc** command output for an invalidate entry

```
onstat -g ssc snipit

...
Statement Cache Entries:

uniqid lru hash ref_cnt hits  flag  heap_ptr       database           user
------------------------------ ---- ----------------------------------------------------
...
    7   1 2404       0    0      F  463d0438       stores_demo        informix
```

```
select count(*) from items

...

Invalidate it:

update syssscelem set valid = 0 where uniqid = 7;

Confirm it is invalid with onstat -g ssc:

Statement Cache Entries:

uniqid lru hash ref_cnt hits  flag  heap_ptr       database            user
------------------------------ ---- ----------------------------------------------------
...
    7   1 2404       0    0     DF  463d0438       stores_demo         informix

select count(*) from items
```

The user can confirm a flag of 'D' in 'onstat -g ssc' output and can query sysmaster:syssscelem to confirm 'valid' column is 0.
Note: Invalid entry cannot be changed to valid.

Locking a statement

You can lock an entry of your choice in the Statement Cache even when UPDATE STATISTICS is executed on tables in the sql statement.

For example, Figure 3 shows the entries that the **onstat -g ssc** command displays after UPDATE STATISTICS was executed on the **items** table between the execution of the first and second SQL statements.

The `Statement Cache Entries` portion of the **onstat -g ssc** output in Figure 3 displays a **flag** field that indicates whether or not an entry has been locked in the SQL statement cache.

Figure 3. Sample **onstat -g ssc** command output for locking an entry

```
onstat -g ssc snipit

...
Statement Cache Entries:

uniqid lru hash ref_cnt hits  flag  heap_ptr       database            user
------------------------------ ---- ----------------------------------------------------
...
    7   1 2404       0    0      F  463d0438       stores_demo         informix
  select count(*) from items

  3  232       1    0   -F  aa8b020  vjp_stores   virginia

...

Lock it:

update syssscelem set locked = 1 where uniqid = 7;

Confirm it is locked with onstat -g ssc:

Statement Cache Entries:

uniqid lru hash ref_cnt hits  flag  heap_ptr       database            user
------------------------------ ---- ----------------------------------------------------
...
    7   1 2404       0    0     FL  463d0438       stores_demo         informix

select count(*) from items
```

The user can confirm a flag of 'L' in 'onstat -g ssc' output and can query sysmaster:syssscelem to confirm 'locked' column is 1.
Note: Statements can be locked and unlocked as many times as desired.

Monitor sessions and threads

You can monitor the number of active sessions and threads and the amount of resources that they are using. Monitoring sessions and threads is important for sessions that perform queries as well as sessions that perform inserts, updates, and deletes.

Some of the information that you can monitor for sessions and threads allows you to determine if an application is using a disproportionate amount of the resources.

Note: Session threads for a stored procedure with a PDQ priority setting and a GROUP BY clause are not released until a session is completed.

- Monitor sessions and threads with onstat commands
  You can use several **onstat** utility commands to monitor active sessions and threads.
- Monitor sessions and threads with SMI tables
  You can use the **syssessions** and the **syssesprof** system-monitoring interface (SMI) tables to obtain information about sessions and threads.

Monitor sessions and threads with onstat commands

You can use several **onstat** utility commands to monitor active sessions and threads.

Use the following **onstat** utility commands to monitor sessions and threads:

- **onstat -u**
- **onstat -g ath**
- **onstat -a act**
- **onstat -a cpu**
- **onstat -a ses**
- **onstat -g mem**
- **onstat -g stm**

- Monitor blocking threads with the onstat -g bth and onstat -g BTH commands
  Running threads take ownership of various objects and resources; for example, buffers, locks, mutexes, decision support memory, and others. Contention for these resources among hundreds or thousands of threads can result in chains of dependencies. Use the **onstat -g bth** command to display the dependencies between blocking and waiting threads. Use the **onstat -g BTH** command to display session and stack information for the blocking threads.
- Monitor threads with onstat –u output
  Use the **onstat –u** command to display information about active threads that require a database server task-control block.
- Monitor threads with onstat -g ath output
  Use the **onstat -g ath** command to view a list of all threads. Unlike the **onstat –u** command, this list includes internal daemon threads that do not have a database server task-control block.
- Monitor threads with onstat -g act output
  Use the **onstat -g act** command to obtain a list of active threads. The **onstat -g act** output shows a subset of the threads that are also listed in **onstat -g ath** output.
- Monitor threads with onstat -g cpu output
  Use the **onstat –g cpu** command to display the last time the thread ran, how much CPU time the thread used, the number of times the thread ran, and other statistics about all the threads running in the server
- Monitor session resources with onstat -g ses output
  Use the **onstat -g ses** command to monitor the resources allocated for and used by a session, in particular, a session that is running a decision-support query. The **onstat -g ses** command also shows information on recently terminated sessions.
- Monitor session memory with onstat -g mem and onstat -g stm output
  Use the **onstat -g mem** and **onstat -g stm** commands to obtain information about the memory used for each session.

Monitor blocking threads with the onstat -g bth and onstat -g BTH commands

Running threads take ownership of various objects and resources; for example, buffers, locks, mutexes, decision support memory, and others. Contention for these resources among hundreds or thousands of threads can result in chains of dependencies. Use the **onstat -g bth** command to display the dependencies between blocking and waiting threads. Use the **onstat -g BTH** command to display session and stack information for the blocking threads.

For example, a thread that is blocked waiting to enter a critical section might own a row lock for which another thread is waiting. The second thread might be blocking a third thread that is waiting in the MGM query queue. Usually, the duration of such contention is short. However, if a thread is blocked long enough to be noticed, you might need to identify the source of the contention. The **onstat -g bth** command discovers the chains of dependency and displays blocker threads followed by waiting threads, in order. You can use the resulting picture of contentions to diagnose and correct the issues.

The following example of the **onstat -g bth** command output has multiple threads that are waiting on resources.

Figure 1. The ouptut of the onstat -g bth command

```
This command attempts to identify any blocking threads.

Highest level blocker(s)
 tid      name                 session
 48       sqlexec              26

Threads waiting on resources
 tid      name                 blocking resource              blocker
 49       sqlexec              MGM                            48
 13       readahead_0          Condition (ReadAhead)           -
 50       sqlexec              Lock (0x4411e578)              49
 51       sqlexec              Lock (0x4411e578)              49
 52       sqlexec              Lock (0x4411e578)              49
 53       sqlexec              Lock (0x4411e578)              49
 57       bf_priosweep()       Condition (bp_cond)             -
 58       scan_1.0             Condition (await_MC1)           -
 59       scan_1.0             Condition (await_MC1)           -

Run 'onstat -g BTH' for more info on blockers.
```

In this example, four threads are waiting for a lock that is owned by thread 49. Thread 49 is waiting for MGM resources that are owned by thread 48. If you run the **onstat -g BTH** command, the output shows the session and stack information for the blocking thread, which in this case is thread 48.

Related information:
onstat -g bth and -g BTH: Print blocked and waiting threads

Monitor threads with onstat –u output

Use the **onstat –u** command to display information about active threads that require a database server task-control block.

Active threads include threads that belong to user sessions, as well as some that correspond to database server daemons (for example, page cleaners). Figure 1 shows an example of **onstat -u** output.

Also use the **onstat -u** command to determine if a user is waiting for a resource or holding too many locks, or to get an idea of how much I/O the user has performed.

The utility output displays the following information:

- The address of each thread
- Flags that indicate the present state of the thread (for example, waiting for a buffer or waiting for a checkpoint), whether the thread is the primary thread for a session, and what type of thread it is (for example, user thread, daemon thread, and so on)
  For information on these flags, see the *IBM® Informix® Administrator's Reference*.
- The session ID and user login ID for the session to which the thread belongs
  A session ID of `0` indicates a daemon thread.
- Whether the thread is waiting for a specific resource and the address of that resource
- The number of locks that the thread is holding
- The number of read calls and the number of write calls that the thread has executed
- The maximum number of current, active user threads

If you execute **onstat -u** while the database server is performing fast recovery, several database server threads might appear in the display.
Figure 1. **onstat -u** output

```
Userthreads
address  flags   sessid  user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0       informix -        0        0    0     33       19
80ef18   ---P--F 0       informix -        0        0    0     0        0
80f2a4   ---P--B 3       informix -        0        0    0     0        0
80f630   ---P--D 0       informix -        0        0    0     0        0
80fd48   ---P--- 45      chrisw   ttyp3    0        0    1     573      237
810460   ------- 10      chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42      chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10      chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47      chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46      chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10      chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43      chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10      chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10      chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44      chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

Related information:
onstat -u command: Print user activity profile

Monitor threads with onstat -g ath output

Use the **onstat -g ath** command to view a list of all threads. Unlike the **onstat –u** command, this list includes internal daemon threads that do not have a database server task-control block.

The **onstat -g ath** command display does not include the session ID (because not all threads belong to sessions).

The `status` field contains information on the status of thread, such as `running`, `cond wait`, `IO Idle`, `IO Idle`, `sleeping secs: number_of_seconds`, or `sleeping forever`. The following output example identifies many threads as `sleeping forever`. To improve performance, you can remove or reduce the number of threads that are identified as `sleeping forever`.

Figure 1. **onstat -g ath** output

```
Threads:
tid  tcb           rstcb        prty  status               vp-class   name
2    10bbf36a8      0            2    sleeping forever       3lio    lio vp 0
3    10bc12218      0            2    sleeping forever       4pio    pio vp 0
4    10bc31218      0            2    sleeping forever       5aio    aio vp 0
5    10bc50218      0            2    sleeping forever       6msc    msc vp 0
6    10bc7f218      0            2    sleeping forever       7aio    aio vp 1
7    10bc9e540      10b231028    4    sleeping secs: 1       1cpu    main_loop()
8    10bc12548      0            2    running                1cpu    tlitcppoll
9    10bc317f0      0            3    sleeping forever       1cpu    tlitcplst
10   10bc50438      10b231780    2    sleeping forever       1cpu    flush_sub(0)
11   10bc7f740      0            2    sleeping forever       8aio    aio vp 2
12   10bc7fa00      0            2    sleeping forever       9aio    aio vp 3
13   10bd56218      0            2    sleeping forever      10aio    aio vp 4
14   10bd75218      0            2    sleeping forever      11aio    aio vp 5
15   10bd94548      10b231ed8    3    sleeping forever       1cpu    aslogflush
16   10bc7fd00      10b232630    1    sleeping secs: 26      1cpu     btscanner 0
32   10c738ad8      10b233c38    4    sleeping secs: 1       1cpu     onmode_mon
50   10c0db710      10b232d88    2    cond wait  netnorm     1cpu     sqlexec
```

Threads that a primary decision-support thread started have a name that indicates their role in the decision-support query. The following figure shows four scan threads that belong to a decision-support thread.

Figure 2. **onstat -g ath** output showing scan threads belonging to a decision-support thread

```
Threads:
tid  tcb           rstcb      prty  status               vp-class   name
11   994060        0           4    sleeping(Forever)      1cpu    kaio
12   994394        80f2a4      2    sleeping(secs: 51)     1cpu    btclean
26   99b11c        80f630      4    ready                  1cpu    onmode_mon
32   a9a294        812b64      2    ready                  1cpu    sqlexec
113  b72a7c        810b78      2    ready                  1cpu    sqlexec
114  b86c8c        81244c      2    cond wait(netnorm)     1cpu    sqlexec
115  b98a7c        812ef0      2    cond wait(netnorm)     1cpu    sqlexec
116  bb4a24        80fd48      2    cond wait(netnorm)     1cpu    sqlexec
117  bc6a24        81161c      2    cond wait(netnorm)     1cpu    sqlexec
118  bd8a24        811290      2    ready                  1cpu    sqlexec
119  beae88        810f04      2    cond wait(await_MC1)   1cpu    scan_1.0
120  a8ab48        8127d8      2    ready                  1cpu    scan_2.0
121  a96850        810460      2    ready                  1cpu    scan_2.1
122  ab6f30        8119a8      2    running                1cpu    scan_2.2
```

Related concepts:
Improve connection performance and scalability
Related information:
onstat -g ath command: Print information about all threads

Monitor threads with onstat -g act output

Use the **onstat -g act** command to obtain a list of active threads. The **onstat -g act** output shows a subset of the threads that are also listed in **onstat -g ath** output.

For sample output, see the *IBM® Informix® Administrator's Reference*.

Related information:
onstat -g act command: Print active threads

Monitor threads with onstat -g cpu output

Use the **onstat –g cpu** command to display the last time the thread ran, how much CPU time the thread used, the number of times the thread ran, and other statistics about all the threads running in the server

The following output example shows the ID and name of each thread that is running, the ID of the virtual processor in which each thread is running, the day and time when each thread last ran, how much CPU time each thread used, the number of times each thread was scheduled to run, and the status of each thread.

Figure 1. **onstat -g cpu** command output

```
Thread CPU Info:
tid     name              vp       Last Run           CPU Time     #scheds    status
2       lio vp 0          3lio*   07/18 08:35:35        0.0000          1    IO Idle
3       pio vp 0          4pio*   07/18 08:35:36        0.0102          2    IO Idle
4       aio vp 0          5aio*   07/18 08:35:47        0.6876         68    IO Idle
5       msc vp 0          6msc*   07/18 11:47:24        0.0935         14    IO Idle
6       main_loop()       1cpu*   07/18 15:02:43        2.9365      23350    sleeping secs: 1
7       soctcppoll        7soc*   07/18 08:35:40        0.1150          1    running
8       soctcpio          8soc*   07/18 08:35:40        0.0037          1    running
9       soctcplst         1cpu*   07/18 11:47:24        0.1106         10    sleeping forever
10      soctcplst         1cpu*   07/18 08:35:40        0.0103          6    sleeping forever
11      flush_sub(0)      1cpu*   07/18 15:02:43        0.0403      23252    sleeping secs: 1
12      flush_sub(1)      1cpu*   07/18 15:02:43        0.0423      23169    sleeping secs: 1
13      flush_sub(2)      1cpu*   07/18 15:02:43        0.0470      23169    sleeping secs: 1
14      flush_sub(3)      1cpu*   07/18 15:02:43        0.0407      23169    sleeping secs: 1
15      flush_sub(4)      1cpu*   07/18 15:02:43        0.0307      23169    sleeping secs: 1
16      flush_sub(5)      1cpu*   07/18 15:02:43        0.0323      23169    sleeping secs: 1
17      flush_sub(6)      1cpu*   07/18 15:02:43        0.0299      23169    sleeping secs: 1
18      flush_sub(7)      1cpu*   07/18 15:02:43        0.0314      23169    sleeping secs: 1
19      kaio              1cpu*   07/18 14:56:42        1.4560    2375587    IO Idle
20      aslogflush        1cpu*   07/18 15:02:43        0.0657      23166    sleeping secs: 1
21      btscanner_0       1cpu*   07/18 15:00:53        0.0484        784    sleeping secs: 61
37      onmode_mon        1cpu*   07/18 15:02:43        0.3467      23165    sleeping secs: 1
43      dbScheduler       1cpu*   07/18 14:58:14        1.6613        320    sleeping secs: 31
44      dbWorker1         1cpu*   07/18 13:48:10        0.4264        399    sleeping forever
45      dbWorker2         1cpu*   07/18 14:48:11        1.9346       2936    sleeping forever
94      bf_priosweep()    1cpu*   07/18 15:01:42        0.0431         77    cond wait  bp_cond
```

Related information:
onstat -g cpu: Print runtime statistics

Monitor session resources with onstat -g ses output

Use the **onstat -g ses** command to monitor the resources allocated for and used by a session, in particular, a session that is running a decision-support query. The **onstat -g ses** command also shows information on recently terminated sessions.

For example, in Figure 1, session number `49` is running five threads for a decision-support query.

Figure 1. **onstat -g ses** output

```
session                                     #RSAM    total      used
id       user     tty      pid    hostname threads  memory     memory
57       informix -        0      -        0        8192       5908
56       user_3   ttyp3    2318   host_1   1        65536      62404
55       user_3   ttyp3    2316   host_1   1        65536      62416
54       user_3   ttyp3    2320   host_1   1        65536      62416
53       user_3   ttyp3    2317   host_1   1        65536      62416
52       user_3   ttyp3    2319   host_1   1        65536      62416
51       user_3   ttyp3    2321   host_1   1        65536      62416
49       user_1   ttyp2    2308   host_1   5        188416     178936
2        informix -        0      -        0        8192       6780
1        informix -        0      -        0        8192       4796

Last 20 Sessions Terminated

Ses ID  Username  Hostname  PID    Time               Reason
36      user_1    host_1    2122   01/19/2015.15:20   session limit txn time (60s)
40      user_1    host_1    2134   01/19/2015.15:14   session limit memory (5124 KB)
47      user_1    host_1    2140   01/19/2015.15:04   session limit logspace (10242 KB)
50      user_1    host_1    2145   01/19/2015.15:02   session limit txn time (39548 KB)
```

Related information:
onstat -g ses command: Print session-related information

Monitor session memory with onstat -g mem and onstat -g stm output

Use the **onstat -g mem** and **onstat -g stm** commands to obtain information about the memory used for each session.

You can determine which session to focus on by the **used memory** column of the **onstat -g ses** output.

Figure 1 shows sample **onstat -g ses** output and some of the **onstat -g mem** and **onstat -g stm** output for Session 16.

- The output of the **onstat -g mem** command shows the total amount of memory used by each session.
  The **totalsize** column of the **onstat -g mem 16** output shows the total amount of memory allocated to the session.
- The output of the **onstat -g stm** command shows the portion of the total memory allocated to the current prepared SQL statement.
  The **heapsz** column of the **onstat -g stm 16** output in the following figure shows the amount of memory allocated for the current prepared SQL statement.

Figure 1. **onstat -g mem** and **onstat -g stm** to determine session memory

```
onstat -g ses

session                                     #RSAM    total      used
id       user     tty      pid      hostname threads  memory     memory
18       informix -        0        -        0        12288      8928
17       informix 12       28826    lyceum   1        45056      33752
16       virginia 6        28743    lyceum   1        90112      79504
14       virginia 7        28734    lyceum   1        45056      33096
3        informix -        0        -        0        12288      10168
2        informix -        0        -        0        12288      8928


onstat -g mem 16

Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
16           V     a9ea020  90112     10608    159        5
...


onstat -g stm 16

session   16 ------------------------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 aa0d018   10056 *SELECT C.customer_num, O.order_num
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
```

Related information:
onstat -g lap command: Print light appends status information
onstat -g mem command: Print pool memory statistics

Monitor sessions and threads with SMI tables

You can use the **syssessions** and the **syssesprof** system-monitoring interface (SMI) tables to obtain information about sessions and threads.

Query the **syssessions** table to obtain the following information.

Column
Description
sid
Session ID
username
Name (login ID) of the user
uid
User ID
pid
Process ID
connected
Time that the session started
feprogram
Absolute path of the executable program or application

In addition, some columns contain flags that show the following information;

- Whether the *primary* thread of the session is waiting for a latch, lock, log buffer, or transaction
- If the thread is in a critical section.

Important: The information in the **syssessions** table is organized by session, and the information in the **onstat -u** output is organized by thread. Also, unlike the **onstat -u** output, the **syssessions** table does not include information about daemon threads, only user threads.
Query the **syssesprof** table to obtain a profile of the activity of a session. This table contains a row for each session with columns that store statistics on session activity (for example, number of locks held, number of row writes, number of commits, number of deletes, and so on).

For a complete list of the **syssessions** columns and descriptions of **syssesprof** columns, see the chapter on the **sysmaster** database in the *IBM® Informix® Administrator's Reference*.

Monitor transactions

You can monitor transactions to track open transactions and the locks that those transactions hold. You can use several **onstat** utility options to view transaction, lock, and session statistics.

The following **onstat** command-line options display session information.

To monitor	Displays the output of	See
Transaction statistics	**onstat -x**	Display information about transactions
User session statistics	**onstat -u**	Display statistics on user sessions
Lock statistics	**onstat -k**	Display information about transaction locks
Sessions running SQL statements	**onstat -g sql** *session-id*	Display statistics on sessions executing SQL statements

- Display information about transactions
  The output of the **onstat -x** command contains information that you can use to monitor transactions.
- Display information about transaction locks
  The output of the **onstat -k** command contains details on the locks that a transaction holds.
- Display statistics on user sessions
  The output of the **onstat -u** command contains statistics on user sessions.
- Display statistics on sessions executing SQL statements
  The output of the **onstat -g sql** command contains statistics on the SQL statements executed by the session

Display information about transactions

The output of the **onstat -x** command contains information that you can use to monitor transactions.

The **onstat -x** output contains the following information for each open transaction:

- The address of the transaction structure in shared memory
- Flags that indicate the following information:
  - The present state of the transaction (user thread attached, suspended, waiting for a rollback)
  - The mode in which the transaction is running (loosely coupled or tight coupled)
  - The stage that the transaction is in (BEGIN WORK, prepared to commit, committing or committed, rolling back)
  - The nature of the transaction (global transaction, coordinator, subordinate, both coordinator and subordinate)
- The thread that owns the transaction

- The number of locks that the transaction holds
- The logical-log file in which the BEGIN WORK record was logged
- The current logical-log id and position
- The isolation level
- The number of attempts to start a recovery thread
- The coordinator for the transaction (if the subordinate is executing the transaction)
- The maximum number of concurrent transactions since you last started the database server

The **onstat** utility is especially useful for monitoring global transactions. For example, you can determine whether a transaction is executing in loosely coupled or tightly coupled mode. These transaction modes have the following characteristics:

- Loosely coupled mode
  Each branch in a global transaction has a separate transaction ID (XID). This mode is the default.
  - The different database servers coordinate transactions, but do not share resources. No two transaction branches, even if they access the same database, can share locks.
  - The records from all branches of a global transaction display as separate transactions in the logical log.
- Tightly coupled mode
  In a single global transaction, all branches that access the same database share the same transaction ID (XID). This mode only occurs with the Microsoft Transaction Server (MTS) transaction manager.
  - The different database servers coordinate transactions and share resources such as locks and log records. The branches with the same XID share locks and can never wait on another branch with the same XID because only one branch is active at one time.
  - Log records for branches with the same XID appear under the same transaction in the logical log.

Figure 1 shows sample output from **onstat -x**. The last transaction listed is a global transaction, as the `G` value in the fifth position of the **flags** column indicates. The `T` value in the second position of the **flags** column indicates that the transaction is running in tightly coupled mode.
Figure 1. **onstat -x** output

```
Transactions
address flags userthread locks beginlg curlog logposit isol    retrys coord
ca0a018 A---- c9da018    0     0       5      0x18484c COMMIT  0
ca0a1e4 A---- c9da614    0     0       0      0x0      COMMIT  0
ca0a3b0 A---- c9dac10    0     0       0      0x0      COMMIT  0
ca0a57c A---- c9db20c    0     0       0      0x0      COMMIT  0
ca0a748 A---- c9db808    0     0       0      0x0      COMMIT  0
ca0a914 A---- c9dbe04    0     0       0      0x0      COMMIT  0
ca0aae0 A---- c9dcff8    1     0       0      0x0      COMMIT  0
ca0acac A---- c9dc9fc    1     0       0      0x0      COMMIT  0
ca0ae78 A---- c9dc400    1     0       0      0x0      COMMIT  0
ca0b044 AT--G c9dc9fc    0     0       0      0x0      COMMIT  0
10 active, 128 total, 10 maximum concurrent
```

The output in Figure 1 shows that this transaction branch is holding 13 locks. When a transaction runs in tightly coupled mode, the branches of this transaction share locks.

Display information about transaction locks

The output of the **onstat -k** command contains details on the locks that a transaction holds.

To find the relevant locks, match the address in the **userthread** column in **onstat -x** output to the address in the **owner** column of **onstat -k** output.

Figure 1 shows sample output from **onstat -x** and the corresponding **onstat -k** command. The `a335898` value in the **userthread** column in the **onstat -x** output matches the value in the **owner** column of the two lines of **onstat -k** output.
Figure 1. **onstat -k** and **onstat -x** output

```
onstat -x

Transactions
address  flags userthread locks   beginlg curlog  logposit   isol    retrys coord
a366018  A---- a334018    0      0       1       0x22b048  COMMIT  0
a3661f8  A---- a334638    0      0       0       0x0       COMMIT  0
a3663d8  A---- a334c58    0      0       0       0x0       COMMIT  0
a3665b8  A---- a335278    0      0       0       0x0       COMMIT  0
a366798  A----  a335898     2      0       0       0x0       COMMIT  0
a366d38  A---- a336af8    0      0       0       0x0       COMMIT  0
 6 active, 128 total, 9 maximum concurrent

onstat -k

Locks
address  wtlist    owner     lklist   type     tblsnum  rowid    key#/bsiz
a09185c  0         a335898   0        HDR+S    100002   20a        0
a0918b0  0         a335898   a09185c  HDR+S    100002   204        0
 2 active, 2000 total, 2048 hash buckets, 0 lock table overflows
```

In the example in Figure 1, a user is selecting a row from two tables. The user holds the following locks:

- A shared lock on one database
- A shared lock on another database

Display statistics on user sessions

The output of the **onstat -u** command contains statistics on user sessions.

You can find the session-id of the transaction by matching the address in the **userthread** column of the **onstat -x** output with the **address** column in the **onstat -u** output. The **sessid** column of the same line in the **onstat -u** output provides the session id.

For example, Figure 1 shows the address `a335898` in the **userthread** column of the **onstat -x** output. The output line in **onstat -u** with the same address shows the session id `15` in the **sessid** column.

Figure 1. Obtaining session-id for userthread in onstat -x

```
onstat -x

Transactions
address  flags userthread locks   beginlg curlog  logposit   isol    retrys coord
a366018  A---- a334018    0       0       1       0x22b048  COMMIT  0
a3661f8  A---- a334638    0       0       0       0x0       COMMIT  0
a3663d8  A---- a334c58    0       0       0       0x0       COMMIT  0
a3665b8  A---- a335278    0       0       0       0x0       COMMIT  0
a366798  A----  a335898     2       0       0       0x0       COMMIT  0
a366d38  A---- a336af8    0       0       0       0x0       COMMIT  0
 6 active, 128 total, 9 maximum concurrent


onstat -u

address  flags    sessid  user     tty     wait     tout locks nreads   nwrites
a334018  ---P--D 1        informix -        0        0    0     20       6
a334638  ---P--F 0        informix -        0        0    0     0        1
a334c58  ---P--- 5        informix -        0        0    0     0        0
a335278  ---P--B 6        informix -        0        0    0     0        0
 a335898   Y--P---  15        informix 1        a843d70  0    2     64       0
a336af8  ---P--D 11       informix -        0        0    0     0        0
 6 active, 128 total, 17 maximum concurrent
```

For a transaction executing in loosely coupled mode, the first position of the **flags** column in the**onstat -u** output might display a value of `T`. This `T` value indicates that one branch within a global transaction is waiting for another branch to complete. This situation could occur if two different branches in a global transaction, both using the same database, tried to work on the same global transaction simultaneously.

For a transaction executing in tightly coupled mode, this `T` value does not occur because the database server shares one transaction structure for all branches that access the same database in the global transaction. Only one branch is attached and active at one time and does not wait for locks because the transaction owns all the locks held by the different branches.

Display statistics on sessions executing SQL statements

The output of the **onstat -g sql** command contains statistics on the SQL statements executed by the session

To obtain information about the last SQL statement that each session executed, issue the **onstat -g sql** command with the appropriate session ID.

Figure 1 shows sample output for this option using the same session ID obtained from the **onstat -u** sample in Figure 1.

Figure 1. onstat -g sql output

```
onstat -g sql 15

Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
15    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  select l.customer_num, l.lname, l.company,   l.phone, r.call_dtime,
    r.call_descr from customer l, vjp_stores@gilroy:cust_calls r where
    l.customer_num = r.customer_num

Last parsed SQL statement :
  select l.customer_num, l.lname, l.company,   l.phone, r.call_dtime,
    r.call_descr from customer l, vjp_stores@gilroy:cust_calls r where
    l.customer_num = r.customer_num
```


The onperf utility on UNIX

The **onperf** utility is a windowing environment that you can use to monitor the database server performance. The **onperf** utility monitors the database server running on the UNIX operating system.

- Overview of the onperf utility
  The **onperf** utility is a graphical tool that you can use for displaying most of the same database server metrics that you can view on **onstat** utility reports.
- Requirements for running the onperf utility
  The computer that is running the **onperf** utility must support the X terminal and the **mwm** window manager.
- Starting the onperf utility and exiting from it
  Before you start the **onperf** utility, set the **DISPLAY** and **LD_LIBRARY_PATH** environment variables.
- The onperf user interface
  When you invoke the **onperf** utility, it displays an initial graph-tool window. From this graph-tool window, you can display additional graph-tool windows as well as the query-tree, data-collector, and activity tools.
- Why you might want to use onperf
  You can use the **onperf** utility for routine monitoring, diagnosing sudden performance loss, and diagnosing performance degradation.
- onperf utility metrics
  When you use the **onperf** utility, you can view various metric classes.

Related reference:
Database server tools

Overview of the onperf utility

The **onperf** utility is a graphical tool that you can use for displaying most of the same database server metrics that you can view on **onstat** utility reports.

The **onperf** utility provides the following advantages over the **onstat** utility:

- Displays metric values graphically in real time
- Allows you to choose which metrics to monitor
- Allows you to scroll back to previous metric values to analyze a trend
- Allows you to save performance data to a file for review at a later time

You cannot use the **onperf** utility on High-Availability Data Replication (HDR) secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

- Basic onperf utility functions
  The **onperf** utility displays the values of the database server metrics in a tool window and saves the database server metric values to a file. You can review the contents of this file.
- onperf utility tools
  The **onperf** utility provides Motif windows, called *tools*, which display metric values.

Related concepts:
Requirements for running the onperf utility
Starting the onperf utility and exiting from it
The onperf user interface
Why you might want to use onperf
onperf utility metrics

Basic onperf utility functions

The **onperf** utility displays the values of the database server metrics in a tool window and saves the database server metric values to a file. You can review the contents of this file.

- Display metric values
  The **onperf** utility displays database server metrics obtained from shared memory.
- Save metric values to a file
  The **onperf** utility saves collected metrics in a history file.
- Review metric measurements
  You can review the contents of a history file in a tool window. When you request a tool to display a history file, the **onperf** utility starts a playback process that reads the data from disk and sends the data to the tool for display.

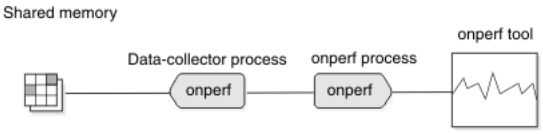
Display metric values

The **onperf** utility displays database server metrics obtained from shared memory.

When **onperf** starts, it activates the following processes:

- **The onperf process.** This process controls the display of **onperf** tools.
- **The data-collector process.** This process attaches to shared memory and passes performance information to the **onperf** process for display in an **onperf** tool.

An **onperf** tool is a Motif window that an **onperf** process manages, as Figure 1 shows.
Figure 1. Data flow from shared memory to an onperf tool window

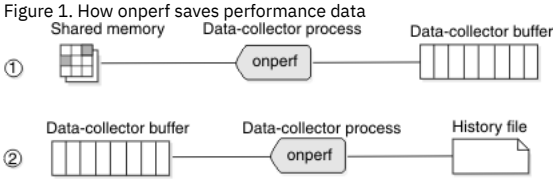


Save metric values to a file

The **onperf** utility saves collected metrics in a history file.

The **onperf** utility allows designated metrics to be continually buffered. The data collector writes these metrics to a circular buffer called the *data-collector buffer*. When the buffer becomes full, the oldest values are overwritten as the data collector continues to add data. The current contents of the data-collector buffer are saved to a history file, as Figure 1 illustrates.

Figure 1. How onperf saves performance data



The **onperf** utility uses either a binary format or an ASCII representation for data in the history file. The binary format is host-dependent and allows data to be written quickly. The ASCII format is portable across platforms.

You have control over the set of metrics stored in the data-collector buffer and the number of samples. You could buffer all metrics; however, this action might consume more memory than is feasible. A single metric measurement requires 8 bytes of memory. For example, if the sampling frequency is one sample per second, then to buffer 200 metrics for 3,600 samples requires approximately 5.5 megabytes of memory. If this process represents too much memory, you must reduce the depth of the data-collector buffer, the sampling frequency, or the number of buffered metrics.

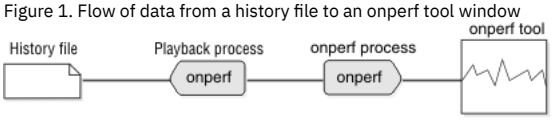
To configure the buffer depth or the sampling frequency, you can use the Configuration dialog box. For more information about the Configuration dialog box, see The graph-tool Configure menu and the Configuration dialog box.

Review metric measurements

You can review the contents of a history file in a tool window. When you request a tool to display a history file, the **onperf** utility starts a playback process that reads the data from disk and sends the data to the tool for display.

The playback process is similar to the data-collector process mentioned under Save metric values to a file. However, instead of reading data from shared memory, the playback process reads measurements from a history file. Figure 1 shows the playback process.

Figure 1. Flow of data from a history file to an onperf tool window



onperf utility tools

The **onperf** utility provides Motif windows, called *tools*, which display metric values.

Table 1. **onperf** utility tools

Tool	Description
Graph tool	This tool allows you to monitor general performance activity. You can use this tool to display any combination of metrics that **onperf** supports and to display the contents of a history file. For more information, see Graph tool.
Query-tree tool	This tool displays the progress of individual queries. For more information, see Query-tree tool.
Status tool	This tool displays status information about the database server and allows you to save the data that is currently held in the data-collector buffer to a file. For more information, see Status tool.
Activity tools	These tools display specific database server activities. Activity tools include disk, session, disk-capacity, physical-processor, and virtual-processor tools. The physical-processor and virtual-processor tools, respectively, display information about all CPUs and VPs. The other activity tools each display the top 10 instances of a resource ranked by a suitable activity measurement. For more information, see Activity tools.

Requirements for running the onperf utility

The computer that is running the **onperf** utility must support the X terminal and the **mwm** window manager.

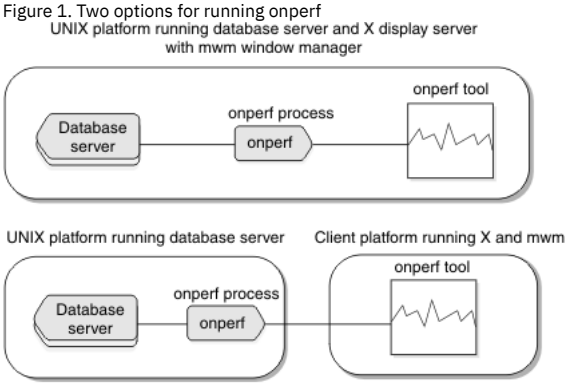
When you install the database server, the following executable files are written to the **$INFORMIXDIR/bin** directory:

- **onperf**
- **onedcu**
- **onedpu**
- **xtree**

In addition, the **onperf.hlp** online help file is placed in the directory **$INFORMIXDIR/hhelp**.

When the database server is installed and running in online mode, you can bring up **onperf** tools either on the computer that is running the database server or on a remote computer or terminal that can communicate with your database server instance. Figure 1 illustrates both possibilities. In either case, the computer that is running the **onperf** tools must support the X terminal and the **mwm** window manager.

Figure 1. Two options for running onperf



Related concepts:
Overview of the onperf utility
Starting the onperf utility and exiting from it
The onperf user interface
Why you might want to use onperf
onperf utility metrics

Starting the onperf utility and exiting from it

Before you start the **onperf** utility, set the **DISPLAY** and **LD_LIBRARY_PATH** environment variables.

Prerequisite: Set the **DISPLAY** environment variable as follows:

```
C shell          setenv DISPLAY displayname0:0 #

Bourne shell     DISPLAY=displayname0:0 #
```

In these commands, *displayname* is the name of the computer or X terminal where the **onperf** window should appear.

Set the **LD_LIBRARY_PATH** environment variable to the appropriate value for the Motif libraries on the computer that is running **onperf**.

With the environment properly set up, you can enter **onperf** to bring up a graph-tool window, as described in The onperf user interface.

You can monitor multiple database server instances from the same Motif client by invoking **onperf** for each database server, as the following example shows:

```
INFORMIXSERVER=instance1 ; export INFORMIXSERVER; onperf
INFORMIXSERVER=instance2 ; export INFORMIXSERVER; onperf
...
```

Exiting from the onperf Utility

To exit from the **onperf** utility, use the **Close** option to close each tool window, use the **Exit** option of a tool, or choose **Window Manager > Close**.

Related concepts:
Overview of the onperf utility
Requirements for running the onperf utility
The onperf user interface
Why you might want to use onperf
onperf utility metrics

The onperf user interface

When you invoke the **onperf** utility, it displays an initial graph-tool window. From this graph-tool window, you can display additional graph-tool windows as well as the query-tree, data-collector, and activity tools.

The graph-tool windows have no hierarchy; you can create and close these windows in any order.

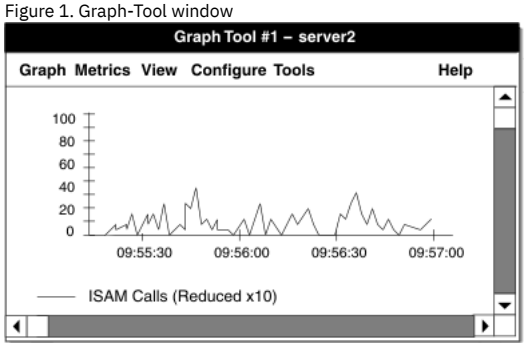
- Graph tool
  The graph tool is the principal **onperf** interface. Use the graph tool to display any set of database server metrics that the **onperf** data collector obtains from shared memory.
- Query-tree tool
  The query-tree tool contains options for monitoring the performance of individual queries.
- Status tool
  The status tool enables you to select metrics to store in the data-collector buffer. In addition, you can use this tool to save the data currently held in the data-collector buffer to a file.
- Activity tools
  Activity tools are specialized forms of the graph tool that display instances of the specific activity, based on a ranking of the activity by some suitable metric.

Related concepts:
Overview of the onperf utility
Requirements for running the onperf utility
Starting the onperf utility and exiting from it
Why you might want to use onperf
onperf utility metrics

Graph tool

The graph tool is the principal **onperf** interface. Use the graph tool to display any set of database server metrics that the **onperf** data collector obtains from shared memory.

The Figure 1 shows a diagram of a graph tool that displays a graph of metrics for ISAM calls.
Figure 1. Graph-Tool window



You cannot bring up a graph-tool window from a query-tree tool, a status tool, or one of the activity tools.

- Graph-tool title bar
  When you invoke **onperf**, the initial graph-tool window displays **serverName**, the database server that the **INFORMIXSERVER** environment variable specifies, in the title bar. The data comes from the shared memory of the indicated database server instance.
- Graph-tool graph menu
  The **Graph** menu contains options for creating, opening, saving the contents of, printing the contents of, annotating, and closing a graph tool.
- Graph-tool metrics menu
  The **Metrics** menu contains options for choosing the class of metrics to display in the graph tool.
- Graph-tool view menu
  The **View** menu contains options for changing how the graph tool appears.
- The graph-tool Configure menu and the Configuration dialog box
  The **Configure** menu contains options of opening, editing, and saving **onperf** configuration information.
- Graph-tool Tools menu
  The **Tools** menu contains options that start additional **onperf** tools.
- Changing the scale of metrics
  When **onperf** displays metrics, it automatically adjusts the scale of the y-axis to accommodate the largest value. You can use the Customize Metric dialog box to establish one for the current display.
- Displaying recent-history values
  When you use the **onperf** utility, you can scroll back over previous metric values that are displayed in a line graph. This is useful for analyzing recent trends.

Graph-tool title bar

When you invoke **onperf**, the initial graph-tool window displays **serverName**, the database server that the **INFORMIXSERVER** environment variable specifies, in the title bar. The data comes from the shared memory of the indicated database server instance.

If the configuration of an initial graph-tool has not yet been saved or loaded from disk, **onperf** does not display the name of a configuration file in the title bar.

If you open a historical data file, for example named **caselog.23April.2PM**, in this graph-tool window, the title bar displays **caselog.23.April.23.April.2PM**.

Graph-tool graph menu

The **Graph** menu contains options for creating, opening, saving the contents of, printing the contents of, annotating, and closing a graph tool.

The **Graph** menu provides the following options.

Option
Use

New
Creates a new graph tool. All graph tools that you create using this option share the same data-collector and **onperf** processes. To create new graph tools, use this option rather than invoke **onperf** multiple times.

Open History File
Loads a previously saved file of historical data into the graph tool for viewing. If the file does not exist, **onperf** prompts you for one. When you select a file, **onperf** starts a playback process to view the file.

Save History File
Saves the contents of the data-collector buffer to either an ASCII or a binary file, as specified in the Configuration dialog box.

Save History File As
Specifies the filename in which to save the contents of the data-collector buffer.

Annotate
Brings up a dialog box in which you can enter a header label and a footer label. Each label is optional. The labels are displayed on the graph. When you save the graph configuration, **onperf** includes these labels in the saved configuration file.

Print
Brings up a dialog box that allows you to select a destination file. You cannot send the contents of the graph tool directly to a printer; you must use this option to specify a file and subsequently send the file to a printer.

Close
Closes the tool. When a tool is the last remaining tool of the **onperf** session, this menu item behaves in the same way as the **Exit** option.

Exit
Exits **onperf**.
Important: To save your current configuration before you load a new configuration from a file, you must choose **Configure > Save Configuration** or **Configure > Save Configuration As**.

Graph-tool metrics menu

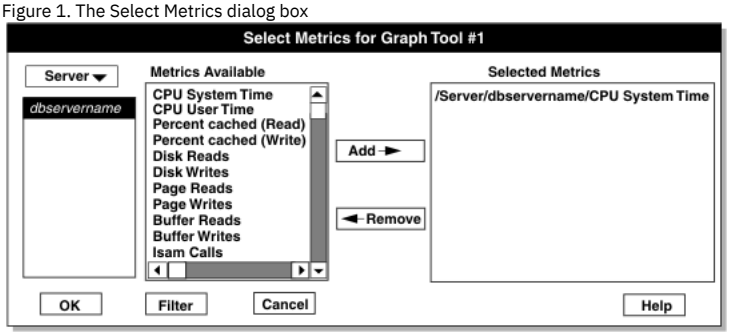
The **Metrics** menu contains options for choosing the class of metrics to display in the graph tool.

Metrics are organized by *class* and *scope*. When you select a metric for the graph tool to display, you must specify the metric class, the metric scope, and the name of the metric.

The *metric class* is the generic database server component or activity that the metric monitors. The *metric scope* depends on the metric class. In some cases, the metric scope indicates a particular component or activity. In other cases, the scope indicates all activities of a given type across an instance of the database server.

The **Metrics** menu has a separate option for each class of metrics. For more information about metrics, see Why you might want to use onperf.

When you choose a class, such as **Server**, you see a dialog box like the one in Figure 1.
Figure 1. The Select Metrics dialog box



The Select Metrics dialog box contains three list boxes. The list box on the left displays the valid scope levels for the selected metrics class. For example, when the scope is set to **Server**, the list box displays the dbservername of the database server instance that is being monitored. When you select a scope from this list, **onperf** displays the individual metrics that are available within that scope in the middle list box. You can select one or more individual metrics from this list and add them to the display by clicking **Add**. To remove them from the display, click **Remove**.
Tip: You can display metrics from more than one class in a single graph-tool window. For example, you might first select **ISAM Calls**, **Opens**, and **Starts** from the **Server** class. When you choose the **Option** menu in the same dialog box, you can select another metric class without exiting the dialog box. For example, you might select the Chunks metric class and add the **Operations**, **Reads**, and **Writes** metrics to the display.

The **Filter** button in the dialog box brings up an additional dialog box in which you can filter long text strings shown in the Metrics dialog box. The Filter dialog box also lets you select tables or fragments for which metrics are not currently displayed.

After you make your selections, you can click OK to proceed, or Cancel if you choose not to proceed.

Graph-tool view menu

The **View** menu contains options for changing how the graph tool appears.

The **View** menu provides the following options.

Line
: Changes the graph tool to the line format. Line format includes horizontal and vertical scroll bars. The vertical scroll bar adjusts the scale of the horizontal time axis. When you raise this bar, **onperf** reduces the scale and vice versa. The horizontal scroll bar allows you to adjust your view along the horizontal time axis.
To change the color and width of the lines in the line format, click the legend in the graph tool. When you do, **onperf** generates a Customize Metric dialog box that provides a choice of line color and width.

Horizontal Bar Graph
: Changes the graph tool to the horizontal bar format.

Vertical Bar Graph
: Changes the graph tool to the vertical bar format.

Pie
: Changes the graph tool to the pie-chart format. To display a pie chart, you must select at least two metrics.

Quick Rescale Axis
: Rescales the axis to the largest point that is currently visible on the graph. This button turns off automatic rescaling.

Configure Axis
: Displays the Axis Configuration dialog box. Use this dialog box to select a fixed value for the y-axis on the graph or select automatic axis scaling.

The graph-tool Configure menu and the Configuration dialog box

The **Configure** menu contains options of opening, editing, and saving **onperf** configuration information.

The **Configure** menu provides the following options.

Edit Configuration
: Brings up the Configuration dialog box, which allows you to change the settings for the current data-collector buffer, graph-tool display options, and data-collector options. The Configuration dialog box appears in Figure 1.

Open Configuration
: Restarts **onperf** with the settings that are stored in the configuration file. Unsaved data in the data-collector buffer is lost. If no configuration file is specified and the default does not exist, the following error message appears:

```
Open file filename failed.
```

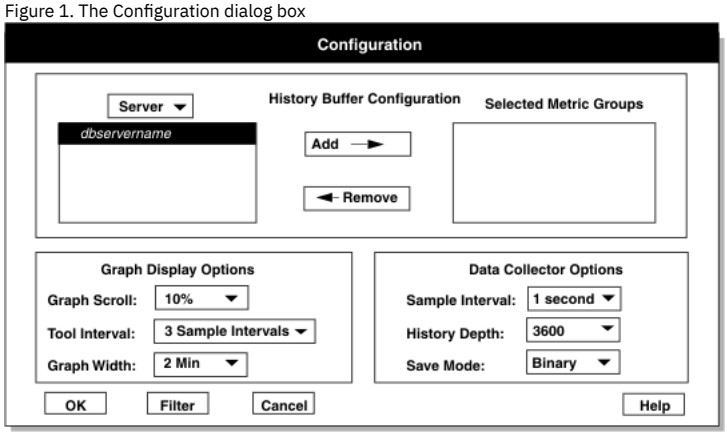
If the specified configuration file does not exist, **onperf** prompts for one.

Save Configuration
: Saves the current configuration to a file. If no configuration file is currently specified, **onperf** prompts for one.

Save Configuration As
: Saves a configuration file. This option always prompts for a filename.

To configure data-collector options, graph-display options, and metrics about which to collect data, choose the **Edit Configuration** option to bring up the Configuration dialog box.

Figure 1. The Configuration dialog box



The Configuration dialog box provides the following options for configuring display.

Option

Use

History Buffer Configuration
: Allows you to select a metric class and metric scope to include in the data-collector buffer. The data collector gathers information about all metrics that belong to the indicated class and scope.

Graph Display Options
: Allows you to adjust the size of the graph portion that scrolls off to the left when the display reaches the right edge, the initial time interval that the graph is to span, and the frequency with which the display is updated.

Data Collector Options
: Controls the collection of data. The sample interval indicates the amount of time to wait between recorded samples. The history depth indicates the number of samples to retain in the data-collector buffer. The save mode indicates the data-collector data should be saved in binary or ASCII format.

Graph-tool Tools menu

The **Tools** menu contains options that start additional **onperf** tools.

This menu provides the following options.

Query Tree
: Starts a query-tree tool. For more information, see Query-tree tool.

Status
: Starts a status tool. For more information, see Status tool.

Disk Activity
: Starts a disk-activity tool. For more information, see Activity tools.

Session Activity
: Starts a session-activity tool. For more information, see Activity tools.

Disk Capacity
: Starts a disk-capacity tool. For more information, see Activity tools.

Physical Processor Activity
: Starts a physical-processor tool. For more information, see Activity tools.

Virtual Processor Activity
: Starts a virtual-processor tool. For more information, see Activity tools.

Changing the scale of metrics

When **onperf** displays metrics, it automatically adjusts the scale of the y-axis to accommodate the largest value. You can use the Customize Metric dialog box to establish one for the current display.

For more information, see Graph-tool view menu.

Displaying recent-history values

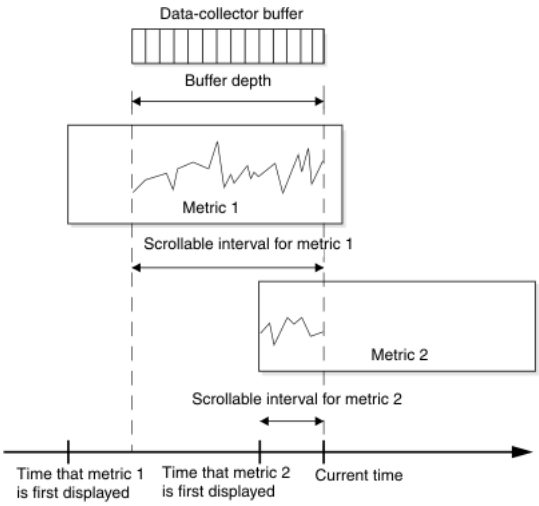
When you use the **onperf** utility, you can scroll back over previous metric values that are displayed in a line graph. This is useful for analyzing recent trends.

The time interval to which you can scroll back is the *lesser* of the following intervals:

- The time interval over which the metric has been displayed
- The history interval that the graph-tool Configuration dialog box specifies
  The length of time you can scroll back through cannot exceed the depth of the data-collector buffer.

  For more information, see The graph-tool Configure menu and the Configuration dialog box.

Figure 1 illustrates the maximum scrollable intervals for metrics that span different time periods.
Figure 1. Maximum scrollable intervals for metrics that span different time periods



Query-tree tool

The query-tree tool contains options for monitoring the performance of individual queries.

The query-tree tool is a separate executable tool that does not use the data-collector process. You cannot save query-tree tool data to a file.

This tool includes a **Select Session** button and a **Quit** button. When you select a session that is running a query, the large detail window displays the SQL operators that constitute the execution plan for the query. The query-tree tool represents each SQL operator with a box. Each box includes a dial that indicates rows per second and a number that indicates input rows. In some cases, not all the SQL operators can be represented in the detail window. The smaller window shows the SQL operators as small icons.

The **Quit** button allows you to exit from the query-tree tool.

Status tool

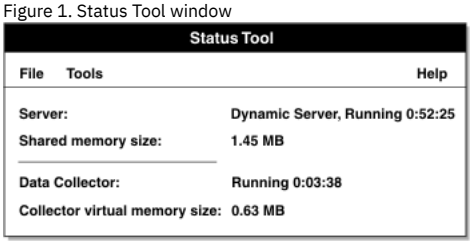
The status tool enables you to select metrics to store in the data-collector buffer. In addition, you can use this tool to save the data currently held in the data-collector buffer to a file.

Figure 1 shows a status tool.

The status tool displays:

- The length of time that the data collector has been running
- The size of the data-collector process area, called the *collector virtual memory size*
  When you select different metrics to store in the data-collector buffer, you see different values for the collector virtual memory size.

Figure 1. Status Tool window



The status tool **File** menu provides the following options.

Option
: **Use**

Close
: This option closes the tool. When it is the last remaining tool of the **onperf** session, Close behaves in the same way as Exit.

Exit
: This option exits **onperf**.

Activity tools

Activity tools are specialized forms of the graph tool that display instances of the specific activity, based on a ranking of the activity by some suitable metric.

You can choose from among the following activity tools:

- The disk-activity tool, which displays the top 10 activities ranked by total operations
- The session-activity tool, which displays the top 10 activities ranked by ISAM calls plus PDQ calls per second
- The disk-capacity tool, which displays the top 10 activities ranked by free space in megabytes
- The physical-processor-activity tool, which displays all processors ranked by nonidle CPU time
- The virtual-processor-activity tool, which displays all VPs ranked by VP user time plus VP system time

The activity tools use the bar-graph format. You cannot change the scale of an activity tool manually; **onperf** always sets this value automatically.

The **Graph** menu provides you with options for closing, printing, and exiting the activity tool.

Why you might want to use onperf

You can use the **onperf** utility for routine monitoring, diagnosing sudden performance loss, and diagnosing performance degradation.

The following sections provide suggestions for different ways to use the **onperf** utility.

- Routine monitoring with onperf
  You can use the **onperf** utility to facilitate routine monitoring. For example, you can display several metrics in a graph-tool window and run this tool throughout the day.
- Diagnosing sudden performance loss
  When you detect a sudden performance dip, you can use the **onperf** utility to examine the recent history of the database server metrics values to identify any trend.
- Diagnosing performance degradation
  You can save the metrics that the **onperf** utility displays, so you can analyze it and compare it to other saved information. This can be useful when analyzing performance problems that gradually develop and might be difficult to diagnose.

Related concepts:
Overview of the onperf utility
Requirements for running the onperf utility
Starting the onperf utility and exiting from it
The onperf user interface
onperf utility metrics

Routine monitoring with onperf

You can use the **onperf** utility to facilitate routine monitoring. For example, you can display several metrics in a graph-tool window and run this tool throughout the day.

Displaying these metrics allows you to monitor database server performance visually at any time.

Diagnosing sudden performance loss

When you detect a sudden performance dip, you can use the **onperf** utility to examine the recent history of the database server metrics values to identify any trend.

The **onperf** utility allows you to scroll back over a time interval, as explained in Displaying recent-history values.

Diagnosing performance degradation

You can save the metrics that the **onperf** utility displays, so you can analyze it and compare it to other saved information. This can be useful when analyzing performance problems that gradually develop and might be difficult to diagnose.

For example, if you detect a degradation in database server response time, it might not be obvious from looking at the current metrics which value is responsible for the slowdown. The performance degradation might also be sufficiently gradual that you cannot detect a change by observing the recent history of metric values. To allow for comparisons over longer intervals, **onperf** allows you to save metric values to a file, as explained in Status tool.

onperf utility metrics

When you use the **onperf** utility, you can view various metric classes.

The following sections describe these metric classes. Each section indicates the scope levels available and describes the metrics within each class.

Database server performance depends on many factors, including the operating-system configuration, the database server configuration, and the workload. It is difficult to describe relationships between **onperf** metrics and specific performance characteristics.

The approach taken here is to describe each metric without speculating on what specific performance problems it might indicate. Through experimentation, you can determine which metrics best monitor performance for a specific database server instance.

- Database server metrics
  The **onperf** utility displays metrics for the named database server, rather than a component of the database server or disk space.
- Disk-chunk metrics
  The **onperf** utility can display metrics for a specific disk chunk.
- Disk-spindle metrics
  The **onperf** utility can display metrics for a disk spindle.
- Physical-processor metrics
  The **onperf** utility can display CPU metrics.
- Virtual-processor metrics
  The **onperf** utility can display metrics for a virtual-processor class.
- Session metrics
  The **onperf** utility can display metrics for an active session.
- Tblspace metrics
  The **onperf** utility can display metrics for a particular tblspace.
- Fragment metrics
  The **onperf** utility can display metrics for an individual table fragment.

Related concepts:
Overview of the onperf utility
Requirements for running the onperf utility
Starting the onperf utility and exiting from it
The onperf user interface
Why you might want to use onperf

Database server metrics

The **onperf** utility displays metrics for the named database server, rather than a component of the database server or disk space.

The **onperf** utility displays the following database server metrics.

Metric Name	Description
CPU System Time	System time, as defined by the platform vendor
CPU User Time	User time, as defined by the platform vendor
Percent Cached (Read)	Percentage of all read operations that are read from the buffer cache without requiring a disk read, calculated as follows: `100 * ((buffer_reads – disk_reads) / (buffer_reads))`
Percent Cached (Write)	Percentage of all write operations that are buffer writes, calculated as follows: `100 * ((buffer_writes – disk_writes) / (buffer_writes))`
Disk Reads	Total number of read operations from disk
Disk Writes	Total number of write operations to disk
Page Reads	Number of pages read from disk
Page Writes	Number of pages transferred to disk
Buffer Reads	Number of reads from the buffer cache
Buffer Writes	Number of writes to the buffer cache
Calls	Number of calls received at the database server
Reads	Number of read calls received at the database server
Writes	Number of write calls received at the database server
Rewrites	Number of rewrite calls received at the database server
Deletes	Number of delete calls received at the database server
Commits	Number of commit calls received at the database server
Rollbacks	Number of rollback calls received at the database server

Metric Name	Description
Table Overflows	Number of times that the tblspace table was unavailable (overflowed)
Lock Overflows	Number of times that the lock table was unavailable (overflowed)
User Overflows	Number of times that the user table was unavailable (overflowed)
Checkpoints	Number of checkpoints written since database server shared memory began
Buffer Waits	Number of times that a thread waited to access a buffer
Lock Waits	Number of times that a thread waited for a lock
Lock Requests	Number of times that a lock was requested
Deadlocks	Number of deadlocks detected
Deadlock Timeouts	Number of deadlock timeouts that occurred (Deadlock timeouts involve distributed transactions.)
Checkpoint Waits	Number of checkpoint waits; in other words, the number of times that threads have waited for a checkpoint to complete
Index to Data Pages Read-aheads	Number of read-ahead operations for index keys
Index Leaves Read-aheads	Number of read-ahead operations for index leaf nodes
Data-path-only Read-aheads	Number of read-ahead operations for data pages
Latch Requests	Number of latch requests
Network Reads	Number of ASF messages read
Network Writes	Number of ASF messages written
Memory Allocated	Amount of database server virtual-address space in kilobytes
Memory Used	Amount of database server shared memory in kilobytes
Temp Space Used	Amount of shared memory allocated for temporary tables in kilobytes
PDQ Calls	The total number of parallel-processing actions that the database server performed
DSS Memory	Amount of memory currently in use for decision-support queries

Disk-chunk metrics

The **onperf** utility can display metrics for a specific disk chunk.

The disk-chunk metrics take the path name of a chunk as the metric scope.

Metric Name	Description
Disk Operations	Total number of I/O operations to or from the indicated chunk
Disk Reads	Total number of reads from the chunk
Disk Writes	Total number of writes to the chunk
Free Space (MB)	The amount of free space available in megabytes

Disk-spindle metrics

The **onperf** utility can display metrics for a disk spindle.

The disk-spindle metrics take the path name of a disk device or operation-system file as the metric scope.

Metric Name	Description
Disk Operations	Total number of I/O operations to or from the indicated disk or buffered operating-system file
Disk Reads	Total number of reads from the disk or operating-system file
Disk Writes	Total number of writes to the disk or operating-system file
Free Space	The amount of free space available in megabytes

Physical-processor metrics

The **onperf** utility can display CPU metrics.

The physical-processor metrics take either a physical-processor identifier (for example, `0` or `1`) or **Total** as the metric scope.

Metric Name	Description
Percent CPU System Time	CPU system time for the physical processors
Percent CPU User Time	CPU user time for the physical processors
Percent CPU Idle Time	CPU idle time for the physical processors
Percent CPU Time	The sum of CPU system time and CPU user time for the physical processors

Virtual-processor metrics

The **onperf** utility can display metrics for a virtual-processor class.

These metrics take a virtual-processor class as a metric scope (cpu, aio, kaio, and so on). Each metric value represents a sum across all instances of this virtual-processor class.

Metric Name	Description
User Time	Accumulated user time for a class
System Time	Accumulated system time for a class
Semaphore Operations	Total count of semaphore operations
Busy Waits	Number of times that virtual processors in class avoided a context switch by spinning in a loop before going to sleep
Spins	Number of times through the loop
I/O Operations	Number of I/O operations per second
I/O Reads	Number of read operations per second
I/O Writes	Number of write operations per second

Session metrics

The **onperf** utility can display metrics for an active session.

These metrics take the active session as the metric scope.

Metric Name	Description
Page Reads	Number of pages read from disk on behalf of a session
Page Writes	Number of pages written to disk on behalf of a session
Number of Threads	Number of threads currently running for the session
Lock Requests	Number of lock requests issued by the session
Lock Waits	Number of lock waits for session threads
Deadlocks	Number of deadlocks involving threads that belong to the session
Deadlock timeouts	Number of deadlock timeouts involving threads that belong to the session
Log Records	Number of log records written by the session
ISAM Calls	Number of ISAM calls by session
ISAM Reads	Number of ISAM read calls by session
ISAM Writes	Number of ISAM write calls by session
ISAM Rewrites	Number of ISAM rewrite calls by session
ISAM Deletes	Number of ISAM delete calls by session
ISAM Commits	Number of ISAM commit calls by session
ISAM Rollbacks	Number of ISAM rollback calls by session
Long Transactions	Number of long transactions owned by session
Buffer Reads	Number of buffer reads performed by session
Buffer Writes	Number of buffer writes performed by session
Log Space Used	Amount of logical-log space used
Maximum Log Space Used	High-watermark of logical-log space used for this session

Metric Name	Description
Sequential Scans	Number of sequential scans initiated by session
PDQ Calls	Number of parallel-processing actions performed for queries initiated by the session
Memory Allocated	Memory allocated for the session in kilobytes
Memory Used	Memory used by the session in kilobytes

Tblspace metrics

The **onperf** utility can display metrics for a particular tblspace.

A tblspace name is composed of the database name, a colon, and the table name (*database:table*).

For fragmented tables, the tblspace represents the sum of all fragments in a table. To obtain measurements for an individual fragment in a fragmented table, use the Fragment Metric class.

Metric Name	Description
Lock Requests	Total requests to lock tblspace
Lock Waits	Number of times that threads waited to obtain a lock for the tblspace
Deadlocks	Number of times that a deadlock involved the tblspace
Deadlock Timeouts	Number of times that a deadlock timeout involved the tblspace
Reads	Number of read calls that involve the tblspace
Writes	Number of write calls that involve the tblspace
Rewrites	Number of rewrite calls that involve the tblspace
Deletes	Number of delete calls that involve the tblspace
Calls	Total calls that involve the tblspace
Buffer Reads	Number of buffer reads that involve tblspace data
Buffer Writes	Number of buffer writes that involve tblspace data
Sequential Scans	Number of sequential scans of the tblspace
Percent Free Space	Percent of the tblspace that is free
Pages Allocated	Number of pages allocated to the tblspace
Pages Used	Number of pages allocated to the tblspace that have been written
Data Pages	Number of pages allocated to the tblspace that are used as data pages

Fragment metrics

The **onperf** utility can display metrics for an individual table fragment.

These metrics take the dbspace of an individual table fragment as the metric scope.

Metric Name	Description
Lock Requests	Total requests to lock fragment
Lock Waits	Number of times that threads have waited to obtain a lock for the fragment
Deadlocks	Number of times that a deadlock involved the fragment
Deadlock Timeouts	Number of times that a deadlock timeout involved the fragment
Reads	Number of read calls that involve the fragment
Writes	Number of write calls that involve the fragment
Rewrites	Number of rewrite calls that involve the fragment
Deletes	Number of delete calls that involve the fragment
Calls	Total calls that involve the fragment
Buffer Reads	Number of buffer reads that involve fragment data
Buffer Writes	Number of buffer writes that involve fragment data
Sequential Scans	Number of sequential scans of the fragment

Metric Name	Description
Percent Free Space	Percent of the fragment that is free
Pages Allocated	Number of pages allocated to the fragment
Pages Used	Number of pages allocated to the fragment that have been written to
Data Pages	Number of pages allocated to the fragment that are used as data pages

Appendix

- Case studies and examples
  This appendix contains a case study with examples of performance-tuning methods that this publication describes.

Case studies and examples

This appendix contains a case study with examples of performance-tuning methods that this publication describes.

- Case study of a situation in which disks are overloaded
  You can identify overloaded disks and the dbspaces that reside on those disks. After you identify the overloaded disks, you can correct the problem.

Case study of a situation in which disks are overloaded

You can identify overloaded disks and the dbspaces that reside on those disks. After you identify the overloaded disks, you can correct the problem.

The following case study illustrates a situation in which the disks are overloaded. This study shows the steps taken to isolate the symptoms and identify the problem based on an initial report from a user, and it describes the needed correction.

A database application that does not have the wanted throughput is being examined to see how performance can be improved. The operating-system monitoring tools reveal that a high proportion of process time was spent idle, waiting for I/O. The database server administrator increases the number of CPU VPs to make more processors available to handle concurrent I/O. However, throughput does not increase, which indicates that one or more disks are overloaded.

To verify the I/O bottleneck, the database server administrator must identify the overloaded disks and the dbspaces that reside on those disks.

To identify overloaded disks and the dbspaces that reside on those disks:

1. To check the asynchronous I/O (AIO) queues, use **onstat -g ioq**. Figure 1 shows the output.
   Figure 1. Output from the onstat -g ioq option

```
AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
opt   0      0      0        0        0        0        0
msc   0      0      0        0        0        0        0
aio   0      0      0        0        0        0        0
pio   0      0      1        1        0        1        0
lio   0      0      1      341        0      341        0
gfd   3      0      1      225        2      223        0
gfd   4      0      1      225        2      223        0
gfd   5      0      1      225        2      223        0
gfd   6      0      1      225        2      223        0
gfd   7      0      0        0        0        0        0
gfd   8      0      0        0        0        0        0
gfd   9      0      0        0        0        0        0
gfd  10      0      0        0        0        0        0
gfd  11      0     28    32693    29603     3090        0
gfd  12      0     18    32557    29373     3184        0
gfd  13      0     22    20446    18496     1950        0
```

   In Figure 1, the **maxlen** and **totalops** columns show significant results:

   - The **maxlen** column shows the largest backlog of I/O requests to accumulate within the queue. The last three queues are much longer than any other queue in this column listing.
   - The **totalops** column shows 100 times more I/O operations completed through the last three queues than for any other queue in the column listing.

   The **maxlen** and **totalops** columns indicate that the I/O load is severely unbalanced.

   Another way to check I/O activity is to use **onstat -g iov**. This option provides a slightly less detailed display for all VPs.

2. To check the AIO activity for each disk device associated with each queue, use **onstat -g iof**, as Figure 2 shows.
   Figure 2. Partial output from the onstat -g iof option

```
gfd pathname         bytes read     page reads  bytes write    page writes io/s
3   /dev/infx5       85456896       41727       207394816      101267      572.9
```

```
        op type     count          avg. time
        seeks       0              N/A
        reads       13975          0.0015
        writes      51815          0.0018
        kaio_reads  0              N/A
        kaio_writes 0              N/A
```

Depending on how your chunks are arranged, several queues can be associated with the same device.

3. To determine the dbspaces that account for the I/O load, use **onstat -d,** as Figure 3 shows.

Figure 3. Output from the onstat -d option

```
Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
c009ad00 1        1        1        1        N        informix rootdbs
c009ad44 2        2001     2        1        N T      informix tmp1dbs
c009ad88 3        1        3        1        N        informix oltpdbs
c009adcc 4        1        4        1        N        informix histdbs
c009ae10 5        2001     5        1        N T      informix tmp2dbs
c009ae54 6        1        6        1        N        informix physdbs
c009ae98 7        1        7        1        N        informix logidbs
c009aedc 8        1        8        1        N        informix runsdbs
c009af20  9         1         9         3         N         informix  acctdbs
 9 active, 32 total

Chunks
address  chk/dbs offset   size     free     bpages   flags pathname
c0099574 1   1   500000   10000    9100               PO-   /dev/infx2
c009960c 2   2   510000   10000    9947               PO-   /dev/infx2
c00996a4 3   3   520000   10000    9472               PO-   /dev/infx2
c009973c 4   4   530000   250000   242492             PO-   /dev/infx2
c00997d4 5   5   500000   10000    9947               PO-   /dev/infx4
c009986c 6   6   510000   10000    2792               PO-   /dev/infx4
c0099904 7   7   520000   25000    11992              PO-   /dev/infx4
c009999c 8   8   545000   10000    9536               PO-   /dev/infx4
c0099a34 9    9    250000  450000    4947               PO-    /dev/infx5
c0099acc 10   9    250000  450000    4997               PO-    /dev/infx6
c0099b64 11   9    250000  450000    169997             PO-    /dev/infx7
 11 active, 32 total
```

In the **Chunks** output, the **pathname** column indicates the disk device. The **chk/dbs** column indicates the numbers of the chunk and dbspace that reside on each disk. In this case, only one chunk is defined on each of the overloaded disks. Each chunk is associated with dbspace number 9.

The **Dbspaces** output shows the name of the dbspace that is associated with each dbspace number. In this case, all three of the overloaded disks are part of the **acctdbs** dbspace.

Although the original disk configuration allocated three entire disks to the **acctdbs** dbspace, the activity within this dbspace suggests that three disks are not enough. Because the load is about equal across the three disks, it does not appear that the tables are necessarily laid out badly or improperly fragmented. However, you might get better performance by adding fragments on other disks to one or more large tables in this dbspace or by moving some tables to other disks with lighter loads.

Related information:
onstat -g iof command: Print asynchronous I/O statistics
onstat -g ioa command: Print combined onstat -g information
onstat -g ioq command: Print I/O queue information
onstat -g iov command: Print AIO VP statistics
onstat -d command: Print chunk information

SNMP Subagent Guide

These topics describe the Simple Network Management Protocol (SNMP) and the software that you need to use SNMP to monitor and manage IBM® Informix® database servers and databases.

These topics are written for the following users:

- Database server administrators
- Backup operators
- Performance engineers

These topics assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience with database server administration, operating-system administration, or network administration

You must install additional software to use the IBM Informix implementation of SNMP. For specific requirements, see Informix implementation of SNMP.

The **onsnmp** utility cannot be run on HDR secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

These topics are taken from *IBM Informix SNMP Subagent Guide*.

- SNMP concepts
  This section provides a brief introduction to Simple Network Management Protocol (SNMP).
- Informix implementation of SNMP
  The IBM Informix implementation of SNMP lets database administrators monitor database servers and databases.

- Management Information Base reference
  An SNMP Network Manager hides most of the structures of the Management Information Base (MIB). However, an understanding of this structure can help you comprehend the information that an SNMP Network Manager displays.

SNMP concepts

This section provides a brief introduction to Simple Network Management Protocol (SNMP).

- What is SNMP?
  The Simple Network Management Protocol (SNMP) is a published, open standard for network management. SNMP lets hardware and software components on networks provide information to network administrators.
- Purpose of the SNMP
  Although the original purpose of the Simple Network Management Protocol (SNMP) was to let network administrators remotely manage an Internet system, the design of SNMP lets network administrators manage applications and systems.
- The SNMP architecture
  The Simple Network Management Protocol (SNMP) architecture includes four layers.

What is SNMP?

The Simple Network Management Protocol (SNMP) is a published, open standard for network management. SNMP lets hardware and software components on networks provide information to network administrators.

Purpose of the SNMP

Although the original purpose of the Simple Network Management Protocol (SNMP) was to let network administrators remotely manage an Internet system, the design of SNMP lets network administrators manage applications and systems.

SNMP provides the following capabilities:

- Hides the underlying system network
- Lets you manage and monitor all network components from one console

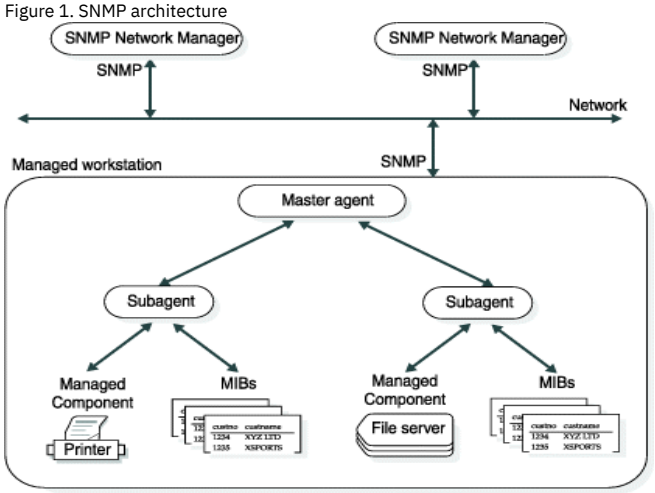
The SNMP architecture

The Simple Network Management Protocol (SNMP) architecture includes four layers.

As the following figure illustrates, the SNMP architecture includes the following layers:

- SNMP Network Managers
- Master agents
- Subagents
- Managed components

Figure 1. SNMP architecture



A network can have multiple SNMP Network Managers. Each workstation can have one master agent. The SNMP Network Managers and master agents use SNMP protocols to communicate with each other. Each managed component has a corresponding subagent and MIBs. SNMP does not specify the protocol for communications between master agents and subagents.

- SNMP network managers
  An SNMP Network Manager is a program that asks for information from master agents and displays that information. You can use most SNMP Network Managers to select the items to monitor and the form in which to display the information.

- Master agents
  A master agent is a software program that provides the interface between an SNMP Network Manager and a subagent.
- Subagents
  A subagent is a software program that provides information to a master agent.
- Managed components
  A managed component is hardware or software that provides a subagent. For example, database servers, operating systems, routers, and printers can be managed components if they provide subagents.
- Management Information Bases
  A Management Information Base (MIB) is a group of tables that specify the information that a subagent provides to a master agent. MIBs follow SNMP protocols.

SNMP network managers

An SNMP Network Manager is a program that asks for information from master agents and displays that information. You can use most SNMP Network Managers to select the items to monitor and the form in which to display the information.

An SNMP Network Manager typically provides the following features:

- Remote monitoring of managed components
- Low-impact sampling of the performance of a managed component
- Correlation of managed component metrics with related system and network metrics
- Graphical presentation of information

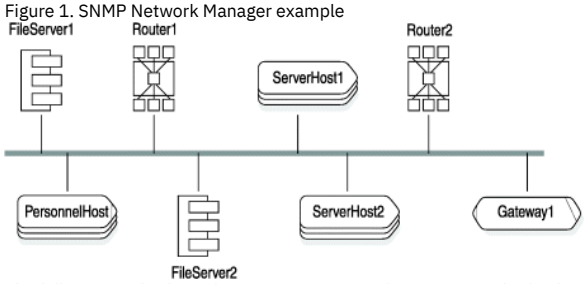
Many hardware and network services have created SNMP Network Managers. For example:

- CA-Unicenter
- Hewlett-Packard Open View
- IBM® NetView®/6000
- Novell Network Management System
- Sun Solstice
- Tivoli® TME 10 NetView

SNMP Network Managers use a connectionless protocol, which means that each exchange between an SNMP Network Manager and a master agent is a separate transaction. A connectionless protocol allows the SNMP Network Manager to perform the following actions:

- Gather information without putting an excessive load on the network
- Function in an environment where heavy traffic can cause network problems

Most SNMP Network Managers provide a graphical user interface (GUI) such as the one that the following figure illustrates. With this SNMP Network Manager, you select a node to monitor and then choose specific information from a menu.
Figure 1. SNMP Network Manager example



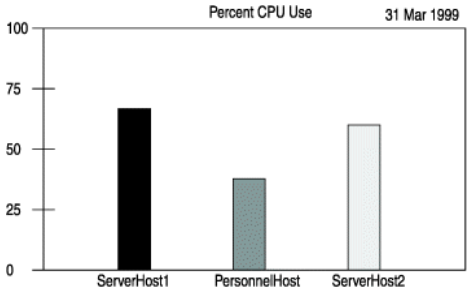
The following code shows how an SNMP Network Manager might display information about the databases on a network. In this example, the network has only one database.

```
Feb 17 1999 [ smoke ] : RDBMS-MIB.rdbmsDbTable
KEY = 72000003
rdbmsDbName = CustomerData
rdbmsDbName.72000003 = AnotherData
rdbmsDbPrivateMibOID = 1.3.6.1.4.1.893
rdbmsDbVendorName = IBM Corporation
rdbmsDbName = CustomerData
rdbmsDbContact = John Doe
```

The following code shows how a different SNMP Network Manager could display the same information.

```
rdbmsDbPrivateMibOID.72000003 = 1.3.6.1.4.1.893
rdbmsDbVendorName.72000003 = IBM Corporation
rdbmsDbName.72000003 = CustomerData
rdbmsDbContact.72000003 = John Doe
```

In addition to text, an SNMP Network Manager might also display graphs or charts, as the following figure illustrates.
Figure 2. Example of monitoring information



Master agents

A master agent is a software program that provides the interface between an SNMP Network Manager and a subagent.

Each workstation that includes a managed component needs to have a master agent. Each managed workstation can have a different master agent. A master agent performs the following tasks:

1. Parses requests from the SNMP Network Manager
2. Routes requests from the SNMP Network Manager to the subagents
3. Collects and formats responses from the subagents
4. Returns the responses to the SNMP Network Manager
5. Notifies the SNMP Network Manager when a request is invalid or information is unavailable

Subagents

A subagent is a software program that provides information to a master agent.

Each managed component has a corresponding subagent. A subagent performs the following tasks:

1. Receives requests from the master agent
2. Collects the requested information
3. Returns the information to the master agent
4. Notifies the master agent when a request is invalid or information is unavailable

Managed components

A managed component is hardware or software that provides a subagent. For example, database servers, operating systems, routers, and printers can be managed components if they provide subagents.

Event notification

When an event occurs that affects the performance or availability of a managed component, the SNMP Network Manager can alert you to that condition.

The following list describes some of the decisions that you can make about event notification:

- Define the conditions that need to be monitored.
- Specify how frequently to poll for each condition.
  When you determine the polling frequency, you must balance the need for prompt notification of an undesirable condition and the burden that polling puts on the network.
- Specify how the SNMP Network Manager notifies you of an event.
  You might choose to have an icon flash or change colors when an event occurs.

Data requests

A data request can be a one-time request or a periodic request. A one-time request is useful for comparing the data for two managed components. Periodic requests are useful for accumulating statistical information about a managed component.

Traps

You can configure the SNMP Network Manager to detect extraordinary events and notify you when they occur.

The following list describes some of the decisions that you can make about traps:

- Define the conditions that need to generate a trap.
- Specify how the SNMP Network Manager notifies you of a trap.
  You might choose to have an icon flash or change colors when a trap occurs.
- Specify how the SNMP Network Manager responds to a trap.
  The SNMP Network Manager can query the managed component to determine the cause and extent of the problem.

Management Information Bases

A Management Information Base (MIB) is a group of tables that specify the information that a subagent provides to a master agent. MIBs follow SNMP protocols.

MIBs use a common interface definition language. The Structure of Management Information (SMI) defines this language and dictates how to use Abstract Syntax Notation One (ASN.1) to describe each table in the MIBs.

MIB table naming conventions

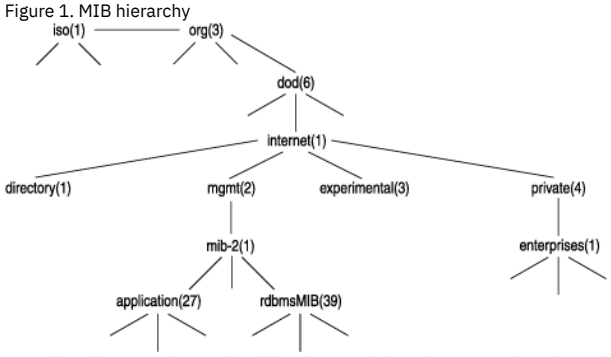
The name of each MIB table starts with the name of the MIB. Thus each table in the RDBMS MIB starts with **rdbms**. For example, the RDBMS MIB includes tables that are named **rdbmsSrvTable** and **rdbmsDbInfoTable**.

The name of each column in an MIB table starts with the name of the table, excluding **Table**. Thus, each column in **rdbmsSrvTable** starts with **rdbmsSrv**. For example, **rdbmsSrvVendorName** and **rdbmsSrvProductName** are columns in **rdbmsSrvTable**.

The MIB hierarchy

All MIBs are part of an information hierarchy that the Internet Assigned Numbers Authority (IANA) defines. The hierarchy defines how to name tables and columns and how to derive the numeric object identifiers (OIDs). The following figure shows the MIB hierarchy.

Figure 1. MIB hierarchy



Even though you rarely see the full path to a table, column, or value, the path is important because the SNMP components use the numeric equivalent of the path to locate data. For example, the following value is the path to the Application MIB:

```
iso.org.dod.internet.mgmt.mib-2.application
```

An OID is the numeric equivalent of a path. It uniquely describes each piece of data that an SNMP Network Manager can obtain and is written as a string of numbers separated by periods (.). For example, the following value is the OID for the Application MIB:

```
1.3.6.1.2.1.27
```

The following value is the OID for a value in the Application MIB:

```
1.3.6.1.2.1.27.1.1.8.2
```

The first part of this OID is the OID for the Application MIB. The final part of the OID assigns values sequentially to each table in the MIB, each column in the table, and each value in a column.

Related concepts:
MIB types and objects

Informix implementation of SNMP

The IBM® Informix® implementation of SNMP lets database administrators monitor database servers and databases.

- Components of the Informix implementation
- Purpose of Informix SNMP
- SNMP standard
  The SNMP standard has two versions: SNMPv1 and SNMPv2.
- SNMP architecture
  The architecture for the IBM Informix implementation of SNMP depends on your operating system.
- Informix implementation of SNMP on UNIX or Linux
- Informix implementation of SNMP on Windows
- GLS and SNMP
  IBM Informix products include a Global Language Support (GLS) feature, which lets you work with languages that use code sets other than the standard English code set. However, the SNMP protocols that OnSNMP supports (SNMPv1 and SNMPv2) do not recognize these different code sets.
- MIB types and objects
  This section describes the types of MIBs and the types of MIB objects that the IBM Informix database server uses.
- Table indexing
- Refresh control value
  As a background task, OnSNMP periodically updates the contents of MIB tables that it derives from catalog information. The refresh control value determines the amount of time that OnSNMP spends refreshing these MIB tables versus the amount of time that it spends responding to queries from the master agent.
- Files installed for SNMP
  This section lists the files that are typically installed for the IBM Informix implementation of SNMP on UNIX and Windows.

Components of the Informix implementation

The IBM® Informix® implementation consists of the following components:

- Master agent
  - On UNIX, a master agent is provided through licensing agreements with vendors.
  - On Windows, install the Microsoft SNMP Extendible master agent.
- Subagent
  The subagent for database servers is OnSNMP.
- Managed components
  In the implementation of SNMP, each database server is a managed component.
- MIBs
  OnSNMP uses several MIBs.

Related tasks:
Windows master agent
Related reference:
UNIX master agents

Purpose of Informix SNMP

- Event notification
  You can configure an SNMP Network Manager to notify you when a specific event occurs.
- Data requests
  You can issue a one-time data request to compare the configuration parameters of two database servers. You can issue periodic data requests to provide statistical information for assessing database performance or resource allocation.
- Traps
- Information that OnSNMP provides
  All the information that OnSNMP provides is available from other sources, such as the system catalog tables, the **sysmaster** and **sysutils** databases, dbaccess calls, and the **onstat** utility.

Event notification

You can configure an SNMP Network Manager to notify you when a specific event occurs.

An event usually has a corresponding object in an MIB table. The following table describes four possible events and the MIB objects that correspond to them.

Table 1. Possible events and the corresponding MIB objects

Event	MIB object
A database server is not available.	**onServerMode**
Database availability changed.	**rdbmsRelState**
A chunk failed.	**onChunkStatus**
A table is running out of space.	**onTablePagesAllocated** **onTablePagesUsed**

For example, you might discover that an application that uses the IBM® Informix® database server stopped responding. You can send email to the help desk to report this problem. The help desk can tell you about the problem, and you can look at **onSessionTable** to determine the cause of the problem.

Data requests

You can issue a one-time data request to compare the configuration parameters of two database servers. You can issue periodic data requests to provide statistical information for assessing database performance or resource allocation.

For example, even if you use a database that is on a local host, you can call a remote technical support representative to report a problem. The problem might be that the data for the transactions running in a particular situation is less than expected. From the remote location, the technical support representative can query an SNMP Network Manager to determine the database server configuration, monitor the database server performance, and identify the bottleneck. OnSNMP provides this information to SNMP Network Managers through the master agent.

Traps

When the status of the database server changes from its current status to any status that is less available, OnSNMP sends a message to the SNMP Network Managers. For example, if a dbspace goes down, the database server status changes from full to limited availability. The message that OnSNMP sends is **rdbmsStateChange**, which is an unsolicited trap. When an SNMP Network Manager notifies you that it received an **rdbmsStateChange** trap, you can query the database server that generated the trap to determine the cause and extent of the problem.

For example, the logical logs for a database server might become full and cause the database server to become unavailable. OnSNMP can notice that the database server is unavailable and send an **rdbmsStateChange** trap to an SNMP Network Manager. The SNMP Network Manager can make an icon flash to notify you of the problem. You can then send data requests to determine the cause of the failure.

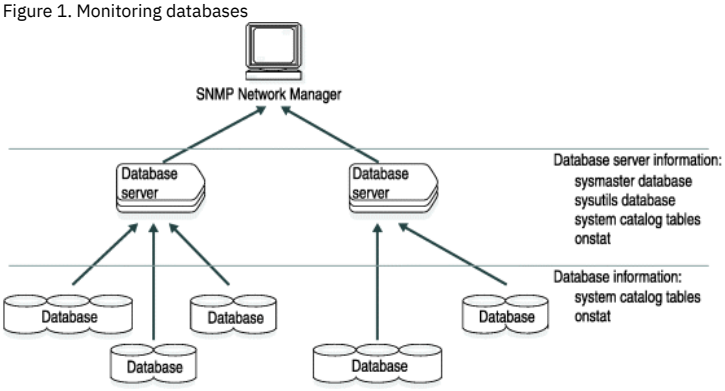
Related tasks:
Installing and configuring a master agent manually

Information that OnSNMP provides

All the information that OnSNMP provides is available from other sources, such as the system catalog tables, the **sysmaster** and **sysutils** databases, dbaccess calls, and the **onstat** utility.

However, the system catalog tables and the **onstat** utility refer only to a single database, and the **sysmaster** and **sysutils** databases refer only to a single database server. OnSNMP provides information that lets an SNMP Network Manager monitor all the IBM® Informix® databases that are on a network. The following figure illustrates this concept.

Figure 1. Monitoring databases



SNMP standard

The SNMP standard has two versions: SNMPv1 and SNMPv2.

The following table lists the versions of the SNMP standard with which OnSNMP complies.

Table 1. Versions of the SNMP standard

Operating system	Version of the SNMP standard
UNIX	SNMPv1 and SNMPv2
Windows	SNMPv1

SNMP architecture

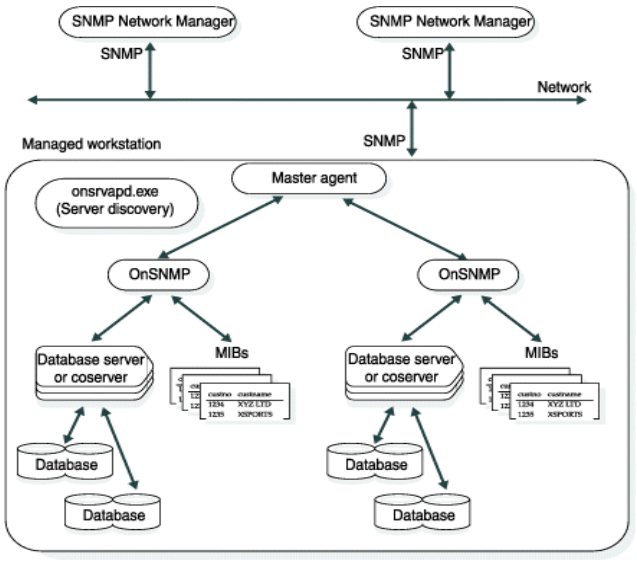
The architecture for the IBM® Informix® implementation of SNMP depends on your operating system.

SNMP is incompatible on High-Availability Data Replication (HDR) secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

SNMP architecture on UNIX

The following figure shows the SNMP architecture for database servers on UNIX. Each managed workstation runs one master agent and one server discovery process. Each database server has one OnSNMP process.

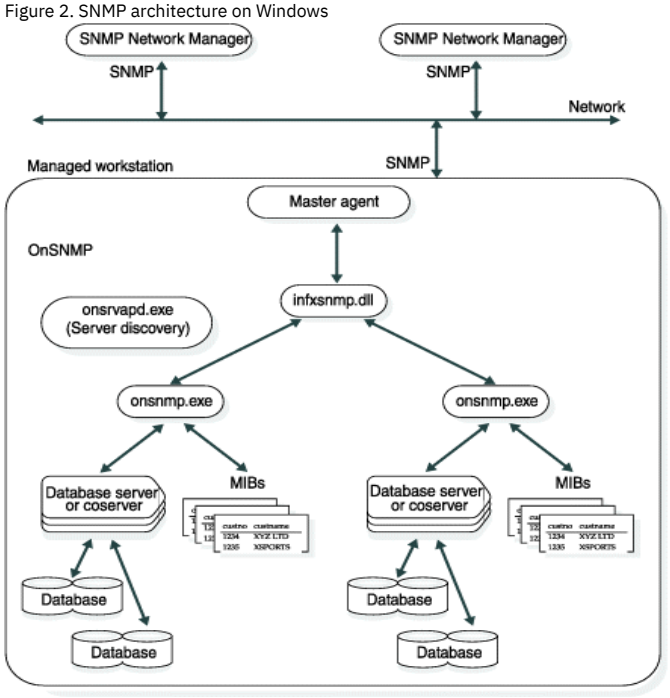
Figure 1. SNMP architecture on UNIX



SNMP architecture on Windows

The following figure shows the SNMP architecture for database servers on Windows. Each managed workstation runs one master agent. The master agent and the SNMP Network Manager use SNMP to communicate with each other. Each managed workstation runs one server discovery process and one infxsnmp.dll. One instance of the **onsnmp** subagent is started for each instance of Informix that runs on the managed workstation. OnSNMP and the master agent do not need to use SNMP to communicate with each other.

Figure 2. SNMP architecture on Windows



Informix implementation of SNMP on UNIX or Linux

To use the IBM® Informix® implementation of SNMP on UNIX or Linux, you must install and start the following software:

- runsnmp.ksh
- An SNMP Network Manager on a network management workstation
- A master agent on each workstation that includes the IBM Informix database server
- the IBM Informix database server
  When you install the database server, the installation procedure installs the OnSNMP subagent and the server discovery process as well as the files needed for SNMP support.

  The discovery process discovers multiple server instances running on the host. These instances might belong to different versions that are installed on different directories. Whenever a server instance is brought online, the discovery process detects it and creates an instance of OnSNMP to monitor the database server.

- The runsnmp.ksh script
  The runsnmp.ksh script on UNIX ensures that both the SNMP master agent and the **onsrvapd** server-discovery daemon are running on a host.

- UNIX master agents
  On UNIX, master agents are provided through licensing agreements.
- UNIX subagent
  When you install the IBM Informix database server on UNIX, the installation procedure installs OnSNMP. OnSNMP consists of the **onsnmp** program.
- UNIX server discovery process
  The runsnmp.ksh script automatically starts the UNIX server discovery process. This section provides procedures for working manually with **onsrvapd**. Some of these procedures include instructions on how to configure OnSNMP.

The runsnmp.ksh script

The runsnmp.ksh script on UNIX ensures that both the SNMP master agent and the **onsrvapd** server-discovery daemon are running on a host.

The runsnmp.ksh file is in the $INFORMIXDIR/snmp directory. You must correctly set the INFORMIXDIR environment variable to the latest installed version of the product and run the script as **root**.

```
>>-runsnmp.ksh--+------------------------+--+------+------------>
                +- -m--master_agent_args-+  '-stop-'
                '- -s--server_disc_args--'

>--+-------+-----------------------------------------------------><
   '-start-'
```

Issue the runsnmp.ksh commands that the following diagram shows.

Table 1. The runsnmp.ksh commands

Option	Description
-m *master_agent_args*	The master-agent arguments can be either **stop** or valid master-agent arguments.
-s *server_disc_args*	The server-discovery arguments can be either **stop** or valid **onsrvapd** arguments.
start	Starts **snmpdm** and **onsrvapd** if they are not running. This option is the default.
stop	Stops **snmpdm** and **onsrvapd** if they are already running and exits.

The *master_agent_args* and the *server_disc_args* are not checked for correctness.

The following examples illustrate how to use runsnmp.ksh:

- Start **snmpdm** and **onsrvapd** if they are not running.

  ```
  runsnmp.ksh
  ```

- Stop **onsrvapd** and **subagents** and then exit.

  ```
  runsnmp.ksh -s stop
  ```

- Stop **onsrvapd** and any **subagents** and then restart **onsrvapd**.

  ```
  runsnmp.ksh -s stop start
  ```

- Stop **snmpdm**, **onsrvapd**, and any **subagents** and then exit.

  ```
  runsnmp.ksh stop
  ```

- Stop **snmpdm** or **snmpdp**, **onsrvapd**, and any **subagents** and then restart **snmpdm** or **snmpdp** and **onsrvapd**.

  ```
  runsnmp.ksh stop start
  ```

- Start **snmpdm** if it is not running, and then start **onsrvapd** with the `none` option, if it is not running.

  ```
  runsnmp.ksh -s "-rnone"
  ```

Related reference:
UNIX subagent

UNIX master agents

On UNIX, master agents are provided through licensing agreements.

The following table lists these master agents.

Master Agent	Company	Website
EMANATE, Version 14.2	SNMP Research	www.snmp.com

For some UNIX platforms, you might be able to use a master agent other than the one provided with the database server. To see whether this applies to your platform, see your release notes.

- Assuring compatibility
- Installing and configuring a master agent manually
- Starting and stopping a master agent
  Start the master agent before you start the IBM® Informix® database server, and stop all IBM Informix database servers on a workstation before you stop the master agent.

Related reference:
Components of the Informix implementation

Assuring compatibility

The following guidelines assure master agent compatibility:

- Only one master agent is provided, usually EMANATE, for each UNIX platform type.
- The subagent that works with the master agent is also provided with the database server.
- In some cases, the platform vendor also supplies a master agent that works with the subagent provided with the database server. This is generally true only if the platform vendor supplies the same type of master agent as that provided with the database server and if the version number of the vendor-supplied master agent is greater than or equal to that of the version provided with the database server.
- Only run one instance of a master agent on a platform. You can run multiple instances of subagents, including multiple instances of **onsnmp**, if multiple database server instances exist.
- IBM® Informix® subagents can coexist with subagents that platform or third-party vendors supply if all the subagents share a common, compatible master agent.

Installing and configuring a master agent manually

The runsnmp.ksh script automatically performs the steps in this section for the master agents provided with the database server. If you bought a master agent from another vendor, follow the installation instructions that the vendor provides.

To configure the EMANATE master agent:

1. Set the following environment variables:
   - Make sure that the PATH environment variable includes $INFORMIXDIR/bin.
   - Set SR_AGT_CONF_DIR to the directory for the EMANATE configuration file.
   - Set SR_LOG_DIR to the directory for the EMANATE log file.

   The EMANATE configuration files are located in the $INFORMIXDIR/snmp/snmpr directory. The log files are located in the /tmp directory. The /tmp directory is the default location if the variable is not set.
2. Make sure that either the Network Information Services or the /etc/services file configures UDP ports 161 and 162 as the SNMP ports.
   a. Use the **grep** command to search /etc/services for `snmp`. The output from **grep** is similar to the following lines:

   ```
   snmp          161/udp
   snmp-trap     162/udp
   ```

   b. Make sure that UDP port 161 is available so that the master agent can be the owner of the port.
3. Add the following line to the snmp configuration file for the **snmpd** daemon to accept messages from **onsnmp**:

   ```
   smuxpeer 0.0
   ```

   If this line does not exist, and the **snmpd** daemon is log enabled, the following message is reported:

   ```
    snmpd log:
   refused smux peer: oid SNMPv2-SMI::zeroDotZero, password , descr rdbms subagent
    onsrvapd log:
   INFO :  onsrvapd pid 9045, poll 5 secs, linger 5 mts, logfile
   /tmp/onsrvapd.42f0d7392355.log.
   MAJOR:  signalCatcher - Caught SIGCHLD.
   MAJOR:  childKilled - Subagent pid 9046 Status 65280.
    onsnmp log:
   MAJOR:  SMUX subagent failed to instantiate managed row
   ```

Related concepts:
Traps

Starting and stopping a master agent

Start the master agent before you start the IBM® Informix® database server, and stop all IBM Informix database servers on a workstation before you stop the master agent.

The best way to start a master agent is to run the runsnmp.ksh script as part of the startup procedure for the system. Similarly, the best way to stop a master agent is to run the runsnmp.ksh script as part of the shutdown procedure. However, you can start or stop a master agent manually if you prefer. Additionally, while a master agent is running, you can make sure that it is running correctly.

The runsnmp.ksh script automatically starts the EMANATE master agent at startup and stops it at shutdown.

If you bought a master agent from another vendor, follow the instructions that the vendor provides.

- Starting a master agent manually
- Stopping a master agent manually
- Making sure that a master agent is running correctly

Starting a master agent manually

To start a master agent manually:

1. Log in as **root**.
   If you do not have **root** user privileges, ask your system administrator to start the master agent.
2. Stop or kill any master agents and daemons that are running on the workstation.
3. Enter the following command for EMANATE: `snmpdm &`

Stopping a master agent manually

To stop a master agent manually:

1. Log in as **root**.
   If you do not have **root** user privileges, ask your system administrator to stop the master agent.
2. Kill the following process:
   For EMANATE, **snmpdm**

The following table describes the command-line options that you can include in the `snmpdm` command for the EMANATE master agent.

Table 1. The snmpdm command-line options

Option	Description
-apall	Turn on all messages.
-aperror	Turn on error messages. Error messages are already turned on by default.
-aptrace	Turn on trace messages.
-apwarn	Turn on warning messages. Warning messages are already turned on by default.
-d	Run the master agent in the foreground.

Making sure that a master agent is running correctly

To make sure that a master agent is running correctly:

1. Check the master agent log file to verify that the master agent has not generated any errors. The log file is located in the **/tmp** directory unless the environment variable mentioned in Installing and configuring a master agent manually is set to a different directory.
2. Verify that the process is running:
   For EMANATE, **snmpdm**

UNIX subagent

When you install the IBM® Informix® database server on UNIX, the installation procedure installs OnSNMP. OnSNMP consists of the **onsnmp** program.

Under normal circumstances, you do not need to start or stop OnSNMP explicitly. If you experience abnormal circumstances and need to start or stop OnSNMP explicitly, contact Technical Support.

The following additional files are provided with the database server for SNMP support.

Table 1. Additional files provided with the database server

Program	Description
onsrvapd daemon	When you start the database server that is on this workstation, **onsrvapd** detects this event and starts OnSNMP for the database server. When the database server halts, **onsrvapd** stops OnSNMP for that database server.
runsnmp.ksh script	This script starts **onsrvapd**. It also starts the master agent that is appropriate for the platform. If you want to run OnSNMP, you need to run runsnmp.ksh each time that you reboot.

Related concepts:
UNIX server discovery process
Related reference:
The runsnmp.ksh script

UNIX server discovery process

The runsnmp.ksh script automatically starts the UNIX server discovery process. This section provides procedures for working manually with **onsrvapd**. Some of these procedures include instructions on how to configure OnSNMP.

The principles for starting and stopping **onsrvapd** manually are the same as the principles for a master agent: start **onsrvapd** before you start the IBM® Informix® database server, and stop all IBM Informix database servers on a workstation before you stop **onsrvapd**.

- Preparing onsrvapd manually
  If you do not use runsnmp.ksh to automatically prepare and start **onsrvapd**, perform the steps in this procedure.
- Issue the onsrvapd command

- Choose an installation directory

Related reference:
UNIX subagent

Preparing onsrvapd manually

If you do not use runsnmp.ksh to automatically prepare and start **onsrvapd**, perform the steps in this procedure.

To prepare **onsrvapd**:

1. Make sure that the owner of **onsrvapd** is **root** and that the group is **informix**.
2. Make sure that the setuid (sticky) bit is set for the **onsrvapd** file.

Issue the onsrvapd command

You can specify the **onsrvapd** command-line options that the following syntax shows. Some of these options affect OnSNMP.

```
>>-onsrvapd--+----------------------+------------------------><
             +- -d------------------+
             +- -g--logginglevel-----+
             +- -k--lingermnts-------+
             +- -l--pathname---------+
             +- -p--pollsecs---------+
             +- -r--server_disc_args-+
             +- -s--level------------+
             '- -V------------------'
```

Table 1. The onsrvapd command-line options

Option	Description
-d	Flag that tells UNIX to run **onsrvapd** once and terminate it instead of starting it as a daemon.
-g *logginglevel*	Logging level to which OnSNMP logs debug information. Valid values are 2, 4, 8, 16, 32, and 64. The default value is 32. The lower the value, the higher the amount of logging. The **onsrvapd** daemon passes this value to OnSNMP.
-k *lingerings*	Number of minutes that **onsrvapd** waits after a database server goes down before **onsrvapd** kills the corresponding OnSNMP. If *lingermnts* is 0, **onsrvapd** waits indefinitely.
-l *pathname*	Directory for the error log files. The file name of the OnSNMP error log is onsnmp.*servername*.log. For example, if your server name is MyServer, the file name of the OnSNMP error log is onsnmp.MyServer.log. The file name of the **onsrvapd** error log is onsrvapd.log.
-p *pollsecs*	Frequency, in seconds, with which OnSNMP polls the database server. The default value is `5` seconds. The **onsrvapd** daemon passes this value to OnSNMP.
-r *level*	Refresh control value. For a description, see Refresh control value.
-V	Prints the OnSNMP version number.

- Starting onsrvapd manually
- Making sure that onsrvapd is running correctly

Related reference:
Refresh control value

Starting onsrvapd manually

To start **onsrvapd** manually:

1. Stop or kill any daemons that are running on the workstation.
2. Enter the command: `onsrvapd`.

To stop **onsrvapd** manually, kill the **onsrvapd** process.

Making sure that onsrvapd is running correctly

To make sure that **onsrvapd** is running correctly:

1. Check the log file to verify that **onsrvapd** has not generated any errors. The log file is located in the /tmp directory.
2. Verify that **onsrvapd** is running.

Choose an installation directory

When you have multiple IBM® Informix® installation directories on a host computer, you must set the latest installation directory as INFORMIXDIR before you run the runsnmp.ksh script to start OnSNMP. If all the directories are for the same type of database server, use the installation directory that has the latest database server version number.

One way to determine the latest directory to use with different types of database server lines is to find the latest version of the SNMP master agent. The EMANATE master agent displays the version when you run it.

Informix implementation of SNMP on Windows

To use the IBM® Informix® implementation of SNMP on Windows, you must install and start the following software:

- Microsoft SNMP service on each workstation that includes the database server
- The database server
  When you install an database server, the installation procedure installs the OnSNMP subagent and the server discovery process as well as the files needed for SNMP support.

- Windows master agent
  The Microsoft TCP/IP custom installation procedure installs the Microsoft SNMP Extendible master agent.
- Windows subagent
- Windows server discovery process
  The IBM Informix Server Discovery Process for SNMP is known as **onsrvapd**. It is installed as a Windows service that runs under the user.

Windows master agent

The Microsoft TCP/IP custom installation procedure installs the Microsoft SNMP Extendible master agent.

For information about this master agent, see the Microsoft TCP/IP Help.

To start the Microsoft TCP/IP Help:

1. Choose Start > Help.
2. Choose the Index tab.
3. Enter the following phrase in the text box: `SNMP`
   In response to this search request, the help system displays a Topics Found dialog box.

4. Choose TCP/IP Procedures Help.

Important: To start or stop the Microsoft SNMP Extendible master agent, you must be a member of the **Administrator Group** on the host workstation.
Related reference:
Components of the Informix implementation

Windows subagent

On Windows, OnSNMP comprises the following files. The table also lists the directories in which the IBM® Informix® installation procedure installs each file.

Table 1. OnSNMP files and associated directories

File	Description	Directory
infxsnmp.dll	Library that provides the interface between onsnmp.exe and the master agent. The IBM Informix installation procedure installs one infxsnmp.dll on each workstation. The initialization process for the master agent loads infxsnmp.dll.	%Windows%\system32
onsnmp.exe	Subagent program. The IBM Informix installation procedure installs an onsnmp.exe file for each database server.	%INFORMIXDIR%\bin
onsrvapd.exe	Server discovery process, which starts onsnmp.exe for each database server that starts. The IBM Informix installation procedure performs the following tasks for onsrvapd.exe: • Installs one onsrvapd.exe on each workstation • Creates the Informix Server Discovery Process for SNMP in the control panel and configures it to start automatically when the system reboots	32-bit platforms: %Windows%\system32 64-bit platforms: Windows\SysWOW64

When you install the database server, the installation procedure automatically installs OnSNMP. When you start the database server that is on a network that uses SNMP, onsrvapd.exe detects this event and starts OnSNMP for the database server. When the database server halts, onsrvapd.exe stops OnSNMP for that database server.

- Start and stop OnSNMP
  Under normal circumstances, you do not need to start or stop OnSNMP explicitly. If you are experiencing abnormal circumstances and need to start or stop OnSNMP explicitly, contact Technical Support.
- Configure OnSNMP
- Windows registry key for the OnSNMP logging level
  On Windows, there is a registry entry to specify the logging level to which OnSNMP logs debugging information.

Start and stop OnSNMP

Under normal circumstances, you do not need to start or stop OnSNMP explicitly. If you are experiencing abnormal circumstances and need to start or stop OnSNMP explicitly, contact Technical Support.

Configure OnSNMP

The IBM® Informix® installation procedure creates a registry key, **OnSnmpSubagent**, under HKEY_LOCAL_MACHINE\SOFTWARE\Informix.

The following table describes the **OnSnmpSubagent** arguments that you can change.

Table 1. OnSnmpSubagent arguments that can be changed

Argument	Value	Description
Environment\LINGER_TIME	*lingermnts*	Number of minutes that the master agent waits after a database server goes down before the master agent kills the corresponding OnSNMP. If *lingermnts* is 0, the master agent waits indefinitely.
Environment\LOGDIR	*pathname*	Complete path of the OnSNMP error-log file, including file name
Environment\REFRESH_TIME	*pollsecs*	Frequency, in seconds, with which OnSNMP polls the database server
Environment\LOGLEVEL	*loglevel*	Logging level to which OnSNMP logs debugging information. The default value is 3. The **onsrvapd** daemon passes this value to OnSNMP.

The following table describes the **OnSnmpSubagent** arguments that you not change.

Table 2. OnSnmpSubagent arguments that do not get changed

Argument	Value	Description
Pathname	*pathname*	Complete path of infxsnmp.dll, including file name
MIBS\APPLMIB	*apploid*	OID for the Application MIB
MIBS\ONMIB	*onoid*	OID for the Online MIB
MIBS\RDBMSMIB	*rdbmsoid*	OID for the RDBMS MIB

The IBM Informix installation procedure also creates an argument, INFXSNMP, under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\SNMP\Parameters\ExtensionAgents. This new argument specifies the location of the **OnSnmpSubagent** registry key, including the name of the key.

To change the OnSNMP configuration, change the values for these arguments.

Windows registry key for the OnSNMP logging level

On Windows, there is a registry entry to specify the logging level to which OnSNMP logs debugging information.

The logging levels that you can specify are:

- 6 (unrecoverable error conditions)
- 5 (major error conditions)
- 4 (warnings in the program)
- 3 (general information)
- 2 (debug information)
- 1 (dump all information)

Windows server discovery process

The IBM® Informix® Server Discovery Process for SNMP is known as **onsrvapd**. It is installed as a Windows service that runs under the user.

The discovery process discovers multiple server instances running on the host. These instances might belong to different versions that are installed on different directories. Whenever a server instance is brought online, the discovery process detects it and creates an instance of OnSNMP to monitor the database server.

- Start and stop onsrvapd
  You can start **onsrvapd** from the services folder in the control panel or from a command prompt.
- Installing the Informix SNMP agent
  If you install the Microsoft SNMP Extendible master agent after you install the IBM Informix database server, the installation procedure cannot create INFXSNMP. To correct this problem, run a program called **inssnmp** to complete the OnSNMP installation.

Start and stop onsrvapd

You can start **onsrvapd** from the services folder in the control panel or from a command prompt.

To start and stop **onsrvapd** from a command prompt, enter the following commands:

- To start **onsrvapd**, enter:

```
net start onsrvapd
```

- To stop **onsrvapd**, enter:

  ```
  net stop onsrvapd
  ```

The OnSNMP Discovery Process (onsrvapd.exe) is installed as an Windows service and starts and stops automatically. You do not need to issue commands at the command line. In the event you want to issue commands from the command line, see the command-line syntax listed in Issue the onsrvapd command.

Ensure that onsrvapd is running correctly, by checking the log file to verify that **onsrvapd** has not generated any errors. For location of the log files, see your release notes. Verify that **onsrvapd** is running.

Installing the Informix SNMP agent

If you install the Microsoft SNMP Extendible master agent after you install the IBM® Informix® database server, the installation procedure cannot create INFXSNMP. To correct this problem, run a program called **inssnmp** to complete the OnSNMP installation.

To run **inssnmp**:

1. Start a Command Prompt session.
2. Go to %INFORMIXDIR%\bin.
3. Enter the following command: `inssnmp`

Tip: If you install a Windows service pack on your computer before you install the Microsoft SNMP Extendible master agent, you might need to reinstall the service pack.

GLS and SNMP

IBM® Informix® products include a Global Language Support (GLS) feature, which lets you work with languages that use code sets other than the standard English code set. However, the SNMP protocols that OnSNMP supports (SNMPv1 and SNMPv2) do not recognize these different code sets.

OnSNMP uses the US English locale when it sends information to the master agent. If OnSNMP cannot convert the code set of the database to the US English locale, it fails and returns error -23101 with the following message:

```
Unable to load locale categories.
```

OnSNMP sends only 7-bit characters. If an eighth bit is present, OnSNMP truncates it. Thus, when an SNMP Network Manager requests character information, OnSNMP returns a value. However, the value might not reflect the name of the database or table.

OnSNMP sends numeric information correctly, regardless of the code set that the database uses.

MIB types and objects

This section describes the types of MIBs and the types of MIB objects that the IBM® Informix® database server uses.

OnSNMP uses the following MIBs:

- Application MIB
- Relational Database Management System (RDBMS) MIB
- IBM Informix Private MIB
- Online MIB in the IBM Informix Private MIB

Application MIB

The Application MIB is a public MIB, which means that the Internet Engineering Task Force (IETF) specifies the structure of the MIB and the MIB tables. A public MIB is the same for all managed components on an SNMP network, not just for IBM Informix products.

OnSNMP uses only **applTable**, which is the portion of the Application MIB that the RDBMS MIB requires.Figure 1 shows the position of the Application MIB in the MIB hierarchy.

The following value is the path to the Application MIB:

```
iso.org.dod.internet.mgmt.mib-2.application
```

The following value is the OID for the Application MIB:

```
1.3.6.1.2.1.27
```

RDBMS MIB

The Relational Database Management System (RDBMS) MIB is a public MIB, which means that the IETF specifies the structure of the MIB and the MIB tables.

A public MIB is the same for all managed database components. However, some of the definitions in the RDBMS MIB are purposely vague to let each vendor tailor the entries to a specific database server. For example, **rdbmsSrvLimitedResourceTable** contains information about the resources that a database server uses. Each database server vendor can decide which resources to include in this table. Figure 1 shows the position of the RDBMS MIB in the MIB hierarchy.

The following value is the path to the RDBMS MIB:

```
iso.org.dod.internet.mgmt.mib-2.rdbmsMIB
```

The following value is the OID for the RDBMS MIB:

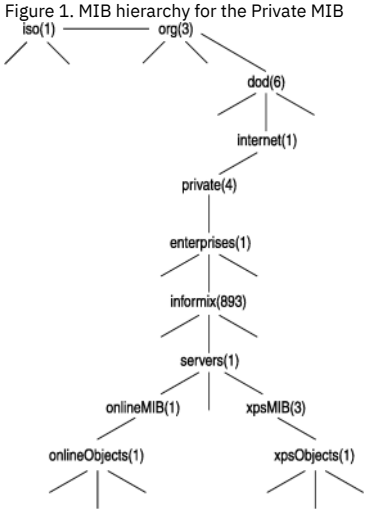
```
1.3.6.1.2.1.39
```

Private MIB

The Private MIB is a private MIB, which means that a private enterprise defines and uses it.

The Internet Assigned Numbers Authority (IANA) assigns a unique enterprise identifier to each company that uses the SNMP protocol. The Private MIB describes information that is relevant to the specific architecture and features of database servers and databases. The following figure shows the MIB hierarchy for the Private MIB.

Figure 1. MIB hierarchy for the Private MIB



The following value is the path to the Private MIB:

```
iso.org.dod.internet.private.enterprises.informix
```

The following value is the OID for the Private MIB:

```
1.3.6.1.4.1.893
```

Online MIB

The Online MIB is in the Private MIB. The Online MIB contains information for all database servers.

In the Online MIB, all tables are after the following node:

```
servers.onlineMIB.onlineObjects
```

The OID for each table in the Online MIB starts with the following value:

```
1.3.6.1.4.1.893.1.1.1
```

MIB objects

An MIB object is similar to a column in a table.

The implementation of SNMP recognizes the following types of MIB objects:

- Traps are defined as MIB objects, but they cannot be retrieved. Instead, when a certain condition is detected, OnSNMP issues an event that includes the object ID that the trap defines.
- Catalog-based MIB objects exist only if the refresh control value (described in Refresh control value) is `once` or `all`.
- Enterprise Replication objects are tables that exist only if a database server is configured to participate in Enterprise Replication.

Related concepts:
Management Information Bases

Table indexing

In the description of the MIBs in Management Information Base reference, the header for each table specifies how each row in the table is indexed. A table can have one or more indexes. For example, the header for **rdbmsSrvTable** is **rdbmsSrvTable[applIndex]**, which means that the table has one index called **applIndex**.

Each index value is concatenated to the column OID with periods between each value. If a MIB table has several indexes, the indexes are concatenated one after the other. Most SNMP Network Managers display only the final portion of the OID that relates to the table being displayed. Some SNMP Network Managers display the OID as part of the information about each individual item; other SNMP Network Managers display the OID as part of a header for a list of values.

- Numeric index values
- Alphabetical index values

Numeric index values

The following line is an example of indexed information:

```
rdbmsRelActiveTime.72000003.893072000 = 11/16/98 12:34:08
```

The following table describes how to interpret the example. For more information about these values, see rdbmsRelTable.

Table 1. Values to interpret the example

Index subvalue	Description
rdbmsRelActiveTime	Name of the column
72000003	**rdbmsDbIndex**
893072000	**applIndex**

Alphabetical index values

When an index is an alphabetic string, such as the name of a configuration parameter, the OID for that index consists of the following elements, all separated by periods:

- Number of letters in the name
- ASCII value for each letter

The following line is an example of alphabetical indexed information:

```
rdbmsSrvParamCurrValue.893072000.4.76.82.85.83.1 = 8
```

The following table describes how to interpret this example. For more information about these values, see rdbmsSrvParamTable.

Table 1. Values to interpret the example

Index subvalue	Description
rdbmsSrvParamCurrValue	Name of the column
893072000	**applIndex**
4.76.82.85.83	**rdbmsSrvParamName**: • 4 = Number of letters • 76 = L • 82 = R • 85 = U • 83 = S
1	**rdbmsSrvParamSubIndex**

Refresh control value

As a background task, OnSNMP periodically updates the contents of MIB tables that it derives from catalog information. The refresh control value determines the amount of time that OnSNMP spends refreshing these MIB tables versus the amount of time that it spends responding to queries from the master agent.

Specify the refresh control value with the runsnmp.ksh -s -r command-line option or the **onsrvapd** -r command-line option. The following table lists the MIB tables that this value affects.

Table 1. MIB tables affected by options

Database-related MIB tables	Table-related MIB tables
rdbmsDbInfoTable **rdbmsDbTable** **rdbmsRelTable** **onBarTable** **onDatabaseTable**	**onActiveTableTable** **onFragmentTable** **onTableTable**

The following table describes the possible values for the refresh control value.

Table 2. Possible values for refresh control value

Value	Description
a or all	Refresh the database-related and table-related tables periodically.
n or none	Do not fill or refresh any of the catalog-based tables. Instead, leave the catalog-based tables empty.
o or once	Fill the database-related and table-related tables once at startup.

The following table lists the default refresh control value for each operating system.

Table 3. Default refresh control values

Operating system	Default refresh control value
UNIX	once

Operating system	Default refresh control value
Windows	all

The best value to use depends on the environment and how you use OnSNMP. If the list of tables and databases changes frequently, it is probably best to use a value of `all` to make sure that the MIB tables are accurate. If the environment includes many tables and databases, it is probably best to use a value of `once` to let OnSNMP respond to queries.

Related concepts:
Issue the onsrvapd command

Files installed for SNMP

This section lists the files that are typically installed for the IBM® Informix® implementation of SNMP on UNIX and Windows.

- Files installed on UNIX or Linux
- Files installed on Windows

Files installed on UNIX or Linux

The runsnmp.ksh file exists for all UNIX versions of SNMP support.

The following files are installed in $INFORMIXDIR/bin.

Table 1. Files installed in $INFORMIXDIR/bin

File name	Description
onsnmp	OnSNMP executable file
onsrvapd	Server discovery process
snmpdm	EMANATE executable or a dummy file for UNIX platforms that EMANATE does not support

The following files are installed in $INFORMIXDIR/snmp.

Table 2. Files installed in $INFORMIXDIR/snmp

File name	Description
./snmpr/snmpd.cnf	EMANATE configuration file or a dummy file for UNIX platforms that EMANATE does not support
.runsnmp.ksh	Script that starts the master agent and **onsrvapd**

OnSNMP uses the following log files by default.

Table 3. Default log files

File name	Description
snmp.log	Log file for EMANATE; not installed on UNIX platforms that EMANATE does not support
onsrvapd.log	Log file for **onsrvapd**.
onsnmp.*.log For IBM® Informix®, the path is **onsnmp.servername.log**	Log file for **onsnmp**.

Files installed on Windows

The following files are created in %Windows%\system32.

Table 1. Files created in %Windows%\system32

File name	Description
infxsnmp.dll	DLL for OnSNMP
onsrvapd.exe	Server discovery process

The following file is created in %INFORMIXDIR%\bin.

Table 2. Files created in %INFORMIXDIR%\bin

File name	Description
onsnmp.exe	OnSNMP executable

In addition, log files are created in the directories that are specified in the registry.

Management Information Base reference

An SNMP Network Manager hides most of the structures of the Management Information Base (MIB). However, an understanding of this structure can help you comprehend the information that an SNMP Network Manager displays.

The descriptions in this section are brief. For detailed descriptions, see the online MIB files. The following table lists the directories for the MIB files.

Table 1. Directories for MIB files

Operating system	MIB directory
UNIX	$INFORMIXDIR/snmp
Windows	%INFORMIXDIR%\etc

Many MIB values are for database servers, depending on the types of database servers that you are using.

This section presents the MIB tables in alphabetical order. For the logical order, see the MIB files. The following table summarizes the MIB tables that OnSNMP uses and indicates the topics that contains more information.

Table 2. MIB tables that OnSNMP uses

MIB	Table	Description
Application	**applTable**	Attributes for each database server
RDBMS	**rdbmsDbInfoTable**	Information about databases
	rdbmsDbTable	Information about databases
	rdbmsRelTable	Information about the relationship between a database and the database server with which it is associated
	rdbmsSrvInfoTable	Information about the database server since it was started
	rdbmsSrvLimited-ResourceTable	Information about the limited resources for each database server
	rdbmsSrvParamTable	Information about the configuration parameters for each database server
	rdbmsSrvTable	Information about a database server
	rdbmsTraps	Information about the traps that OnSNMP can send to the SNMP Network Manager
Online	**onActiveBarTable**	Information about the current ON-Bar activity
	onActiveTableTable	Information about the open and active database tables
	onBarTable	Information about the backup and restore history
	onChunkTable	Information about the chunks that the database servers use
	onDatabaseTable	Information about active databases
	onDbspaceTable	Information about dbspaces
	onErQueueTable	Information about the Enterprise Replication queue
	onErSiteTable	Information about the Enterprise Replication site
	onFragmentTable	Information about the fragments that are in fragmented database tables
	onLockTable	Information about the active locks that database servers are using
	onLogicalLogTable	Information about logical logs
	onPhysicalLogTable	Information about physical logs
	onServerTable	Status and profile information about each active database server
	onSessionTable	Information about each session
	onSqlHostTable	Copy of the connection information
	onTableTable	Information about a database table

- Application MIB
  IBM Informix uses one table from the application MIB. This table provides general-purpose attributes for each database server.
- RDBMS MIB
  The Relational Database Management System (RDBMS) MIB defines several tables that provide information about managed database servers and their databases.
- Online MIB in the Informix Private MIB
  The Online MIB defines several tables that provide information that is relevant for IBM Informix database servers and their databases.

Application MIB

IBM® Informix® uses one table from the application MIB. This table provides general-purpose attributes for each database server.

- applTable

applTable

The following list summarizes this table:

Contents:
Attributes for each database server

Index:
applIndex
Scope of a row:
One database server

The table has the following MIB objects.

Table 1. MIB objects for applTable

MIB object	Description
applIndex	Unique integer index that identifies each database server. This value is the sum of the following values: • IBM® Informix® Enterprise ID * 1,000,000 The IBM Informix Enterprise ID is 893. Therefore, Enterprise ID * 1,000,000 is 893,000,000. • SERVERNUM * 1000
applName	Name of the database server.
applDirectoryName	No OnSNMP support for this MIB object.
applVersion	Version of the database server.
applUptime	Time when the database server was last initialized. This time is the system time according to the master agent. If the database server was last initialized before OnSNMP was last initialized, this value is 0.
applOperStatus	Operating status of the database server: • up (1) • down (2) • halted (3) • (4): OnSNMP does not use this value. • restarting (5)
applLastChange	Time when the database server entered its current state. This time is the system time according to the master agent. If the database server was last initialized before OnSNMP was last initialized, this value is 0.
applInboundAssociations	Number of current SQLCONNECT actions.
applOutboundAssociations	OnSNMP does not support this MIB object.
applAccumulatedInboundAssociations	Number of SQLCONNECT actions that have occurred so far.
applAccumulatedOutboundAssociations	OnSNMP does not support this MIB object.
applLastInboundActivity	Time for the most recent attempt to start or stop a session with a database server. This time is the system time according to the master agent.
applLastOutboundActivity	OnSNMP does not support this MIB object.
applRejectedInboundAssociations	Number of times that the database server rejected an input connection due to administrative reasons or resource limitations.
applFailedOutboundAssociations	OnSNMP does not support this MIB object.

RDBMS MIB

The Relational Database Management System (RDBMS) MIB defines several tables that provide information about managed database servers and their databases. OnSNMP does not support the tables **rdbmsDbLimitedResourceTable** and **rdbmsDbParamTable**.

- rdbmsDbInfoTable
- rdbmsDbTable
- rdbmsRelTable
- rdbmsSrvInfoTable
- rdbmsSrvLimitedResourceTable
- rdbmsSrvParamTable
- rdbmsSrvTable
- rdbmsTraps

rdbmsDbInfoTable

The following list summarizes this table:

Contents:
Information about databases
Index:
rdbmsDbIndex
Scope of a row:
One database that does not have an access state of **unavailable**

The **rdbmsRelState** value indicates the access state for the database.

The table has the following MIB objects.

Table 1. MIB objects for rdbmsDbInfoTable

MIB object	Description
rdbmsDbIndex	See rdbmsDbTable.
rdbmsDbInfoProductName	Name of the database product. For example, this value might be IBM® Informix®.
rdbmsDbInfoVersion	Version number of the database server that created or last restructured this database
rdbmsDbInfoSizeUnits	Units for **rdbmsDbInfoSizeAllocated** and **rdbmsDbInfoSizeUsed**: • Bytes (1) • Kilobytes (2) • Megabytes (3) • Gigabytes (4) • Terabytes (5)
rdbmsDbInfoSizeAllocated	Estimated size allocated for this database in the units that **rdbmsDbInfoSizeUnits** specifies
rdbmsDbInfoSizeUsed	Estimated size in use for this database in the units that **rdbmsDbInfoSizeUnits** specifies
rdbmsDbInfoLastBackup	Date and time when the latest backup of the database was performed. If the database has never been backed up, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

rdbmsDbTable

The following list summarizes this table:

Contents:
Information about databases
Index:
rdbmsDbIndex
Scope of a row:
One database

The table has the following MIB objects.

Table 1. MIB objects for rdbmsDbTable

MIB object	Description
rdbmsDbIndex	Unique integer index that identifies a database. This value is the sum of the following values: • SERVERNUM * 1,000,000 If SERVERNUM is 0, OnSNMP uses 256 instead of 0. • Database number
rdbmsDbPrivateMibOID	OID for the IBM® Informix® Private MIB: 1.3.6.1.4.1.893
rdbmsDbVendorName	Name of the database vendor: IBM Corporation
rdbmsDbName	Name of the database
rdbmsDbContact	Login name of the person who created the database

rdbmsRelTable

The following list summarizes this table:

Contents:
Information about the relationship between a database and the database server with which it is associated

The table has the following MIB objects.

Table 1. MIB objects for rdbmsRelTable

MIB object	Description
rdbmsDbIndex	See rdbmsDbTable.
applIndex	See applTable.

MIB object	Description
rdbmsRelState	Access state between the database server and the database: • Other (1): The database server is online, but one of the dbspaces of the database is down. • Active (2): The database server is actively using the database. The database server is online, and a user opened the database. • Available (3): The database server could use the database if asked to do so. The database server is online, but the database is not open. • Restricted (4): The database is not available. The database server is online, and a user opened the database in exclusive mode. • Unavailable (5)
rdbmsRelActiveTime	Date and time that the database server made the database active. If **rdbmsRelState** is not active, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

rdbmsSrvInfoTable

The following list summarizes this table:

Contents:
Information about the database server since it was started
Index:
applIndex
Scope of a row:
One database server

The table has the following MIB objects.

Table 1. MIB objects for rdbmsSrvInfoTable

MIB Object	Description
applIndex	See applTable.
rdbmsSrvInfoStartupTime	Date and time when the database server was last started
rdbmsSrvInfoFinishedTransactions	Number of transactions completed, either with a commit or with an abort
rdbmsSrvInfoDiskReads	Number of reads from the physical disk
rdbmsSrvInfoLogicalReads	Number of logical reads
rdbmsSrvInfoDiskWrites	Number of writes to the physical disk
rdbmsSrvInfoLogicalWrites	Number of logical writes
rdbmsSrvInfoPageReads	Number of page reads
rdbmsSrvInfoPageWrites	Number of page writes
rdbmsSrvInfoDiskOutOfSpaces	Number of times that the database server has been unable to obtain the desired disk space
rdbmsSrvInfoHandledRequests	Number of requests made to the database server on inbound associations
rdbmsSrvInfoRequestRecvs	Number of receive operations that the database server made while it was processing requests on inbound associations
rdbmsSrvInfoRequestSends	Number of send operations that the database server made while it was processing requests on inbound associations
rdbmsSrvInfoHighwaterInbound-Associations	Greatest number of inbound associations that have been open at the same time
rdbmsSrvInfoMaxInbound-Associations	Greatest number of inbound associations that can be open at the same time

rdbmsSrvLimitedResourceTable

The following list summarizes this table:

Contents:
Information about the limited resources for each database server
Index:
applIndex, **rdbmsSrvLimitedResourceName**
Scope of a row:
One limited resource

The table has the following MIB objects.

Table 1. MIB objects for rdbmsSrvLimitedResourceTable

MIB Object	Description
applIndex	See applTable.

MIB Object	Description
rdbmsSrvLimitedResourceName	Name of the limited resource: • BUFFERS • DS_MAX_QUERIES • DS_MAX_SCANS • DS_TOTAL_MEMORY • LOCKS • LTXEHWM • LTXHWM • STACKSIZE • LOGFILES • DBSPACES • CHUNKS
rdbmsSrvLimitedResourceID	OID or vendor name for the IBM® Informix® Private MIB: 1.3.6.1.4.1.893 or informix
rdbmsSrvLimitedResourceLimit	Maximum value that this limited resource can attain
rdbmsSrvLimitedResourceCurrent	The current value for this limited resource
rdbmsSrvLimitedResourceHighwater	Maximum value that this limited resource has attained since **applUptime** was reset. This value is 0 for DBSPACES and CHUNKS.
rdbmsSrvLimitedResourceFailures	Number of times that the database server tried to exceed the maximum value for this limited resource since **applUptime** was reset. This value is 0 for DBSPACES and CHUNKS.
rdbmsSrvLimitedResourceDescription	Description of the limited resource. This description includes the units for the value for the limited resource.

rdbmsSrvParamTable

The following list summarizes this table:

Contents:
Information about the configuration parameters for each database server
Index:
applIndex, **rdbmsSrvParamName**, **rdbmsSrvParamSubIndex**
Scope of a row:
One configuration parameter that is listed in the configuration file for the database server

The ONCONFIG environment variable specifies the file name of the configuration file. The following table lists the location of the configuration file for each operating system. For more information about the configuration file, see your *IBM Informix Administrator's Guide* and the *IBM Informix Administrator's Reference*. For more information about the ONCONFIG environment variable, see the *IBM Informix Guide to SQL: Reference*.

Table 1. Location of the configure files

Operating system	Location of configuration file
UNIX	$INFORMIXDIR/etc/$ONCONFIG
Windows	%INFORMIXDIR%\etc\%ONCONFIG%

The table has the following MIB objects.

Table 2. MIB objects for rdbmsSrvParamTable

MIB object	Description
applIndex	See applTable.
rdbmsSrvParamName	Name of a configuration parameter
rdbmsSrvParamSubindex	Subindex for the configuration parameter. This value is 1 for every configuration parameter except DATASKIP, DBSPACETEMP, DBSERVERALIASES, and NETTYPE.
rdbmsSrvParamID	OID or vendor name for the IBM® Informix® Private MIB: 1.3.6.1.4.1.893 or informix
rdbmsSrvParamCurrValue	Value of the configuration parameter. OnSNMP obtains this value from the configuration file. Therefore, it does not reflect dynamic changes that you might make to the configuration parameter.
rdbmsSrvParamComment	Purpose of the configuration parameter

rdbmsSrvTable

The following list summarizes this table:

Contents:
Information about a database server
Index:
applIndex

Scope of a row:
One database server

The table has the following MIB objects.

Table 1. MIB objects for rdbmsSrvTable

MIB object	Description
applIndex	See applTable.
rdbmsSrvPrivateMibOID	OID for the IBM® Informix® Private MIB: 1.3.6.1.4.1.893
rdbmsSrvVendorName	Name of the database server vendor: IBM Corporation
rdbmsSrvProductName	Name of the database server product. For example, this value might be IBM Informix.
rdbmsSrvContact	Name of the database server contact: informix

rdbmsTraps

This MIB object contains information about traps that an SNMP subsystem that supports the RDBMS MIB can generate. In this case, the SNMP subsystem is OnSNMP.

- frdbmsStateChange trap

frdbmsStateChange trap

When a database server changes from its status to any less-available status, OnSNMP sends a **rdbmsStateChange** trap message to configured network hosts through the master agent.

The following list summarizes this trap:

Contents:
The **rdbmsRelState** MIB object
Index:
rdbmsDbIndex, **applIndex**
Scope of a row:
If the status of the IBM® Informix® database server becomes unavailable, it generates one trap for each database.

Online MIB in the Informix Private MIB

The Online MIB defines several tables that provide information that is relevant for IBM® Informix® database servers and their databases.

- onActiveBarTable
- onActiveTableTable
- onBarTable
- onChunkTable
- onDatabaseTable
- onDbspaceTable
- onErQueueTable
- onErSiteTable
- onFragmentTable
- onLockTable
- onLogicalLogTable
- onPhysicalLogTable
- onServerTable
- onSessionTable
- onSqlHostTable
- onTableTable

onActiveBarTable

The following list summarizes this table:

Contents:
Information about the current ON-Bar activity
Index:
applIndex, **onActiveBarIndex**
Scope of a row:
One ON-Bar activity

The table has the following MIB objects.

Table 1. MIB objects for onActiveBarTable

MIB Object	Description
applIndex	See applTable.
onActiveBarIndex	A number that OnSNMP assigns
onActiveBarActivityType	Type of activity: • dbspaceBackup (1) • dbspaceRestore (2) • logBackup (3) • logRestore (4) • systemBackup (5) • systemRestore (6)
onActiveBarActivityLevel	Level of activity: • completeBackup (1) • incrementalLevelOne (2) • incrementalLevelTwo (3)
onActiveBarElapsedTime	Length of time since the activity started, in hundredths of seconds
onActiveBarActivitySize	Total number of used pages to scan OnSNMP updates this value as the activity progresses.
onActiveBarActivityScanned	Number of used pages that the activity has scanned so far
onActiveBarActivityCompleted	Number of scanned pages that the activity has transferred for archiving so far
onActiveBarActivityStatus	Status of the activity

onActiveTableTable

The following list summarizes this table:

Contents:
Information about the open and active database tables
Index:
applIndex, **rdbmsDbIndex**, **onTableIndex**
Scope of a row:
One open and active database table

For a fragmented database table, the values in this table are summaries of the values from all the fragments of the database table. The table has the following MIB objects.

Table 1. MIB objects for onActiveTableTable

MIB Object	Description
applIndex	See applTable.
rdbmsDbIndex	See rdbmsDbTable.
onTableIndex	See onDbspaceTable.
onActiveTableStatus	Status of the table: • not Busy (1): The table is not in use. • busy (2): The table is in use. • dirty (3): The table has been modified.
onActiveTableIsBeingAltered	State of the table: • Yes (1): The table is being altered. (An index is being added or dropped, an ALTER TABLE statement is being executed, the alter page count is being updated, or pages are being altered to conform to the latest schema.) • No (2): The table is not being altered.
onActiveTableUsers	Number of users accessing the table
onActiveTableLockRequests	Number of lock requests
onActiveTableLockWaits	Number of lock waits
onActiveTableLockTimeouts	Number of lock timeouts
onActiveTableIsamReads	Number of reads from the database table
onActiveTableIsamWrites	Number of writes to the database table
onActiveTableBufferReads	Number of buffer reads
onActiveTableBufferWrites	Number of buffer writes

onBarTable

The following list summarizes this table:

Contents:
Information about the backup and restore history
Index:
applIndex, **onBarActivityIndex**, **onBarObjectIndex**
Scope of a row:
One object that participated in a backup or restore activity

For information about backup and restore, see the *IBM Informix Backup and Restore Guide*.

The table has the following MIB objects.

Table 1. MIB objects for onBarTable

MIB object	Description
applIndex	See applTable.
onBarActivityIndex	Index to the history
onBarObjectIndex	Index to the object
onBarName	Name of the object
onBarType	Type of object: • blobspace (1) (Only IBM® Informix® provides blobspaces.) • rootDbspace (2) • criticalDbspace (3) • noncriticalDbspace (4) • logicalLog (5)
onBarLevel	Level of the backup action: • completeBackup (1) • incrementalLevelOne(2) • incrementalLevelTwo (3)
onBarStatus	Status of the action on the object: • 0 = successful • Nonzero = error number
onBarTimeStamp	Ending time stamp for the action

onChunkTable

The following list summarizes this table:

Contents:
Information about the chunks that the database servers use
Index:
applIndex, **onDbspaceIndex**, **onChunkIndex**
Scope of a row:
One chunk

The table has the following MIB objects.

Table 1. MIB objects for onChunkTable

MIB object	Description
applIndex	See applTable.
onDbspaceIndex	See rdbmsDbInfoTable.
onChunkIndex	Unique integer index for this chunk The database server generates this value.
onChunkFileName	Path name for the chunk
onChunkFileOffset	Offset into the device, in pages
onChunkPagesAllocated	Chunk size, in pages
onChunkPagesUsed	Number of pages used

MIB object	Description
onChunkType	Type of chunk: • regularChunk (1) • blobChunk (2) • stageBlob (3)
onChunkStatus	Status of the chunk: • offline (1) • online (2) • recovering (3) • inconsistent (4) • dropped (5)
onChunkMirroring	Mirroring status of the chunk: • notMirrored (1) • mirrored (2) • newlyMirrored (3)
onChunkReads	Number of physical-read operations
onChunkPageReads	Number of page reads
onChunkWrites	Number of physical-write operations
onChunkPageWrites	Number of page writes
onChunkMirrorFileName	Path name of the mirror chunk If the chunk is not mirrored, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onChunkMirrorFileOffset	Offset of the mirror, in pages If the chunk is not mirrored, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onChunkMirrorStatus	Mirroring status: • offline (1) • online (2) • recovering (3) • inconsistent (4) • dropped (5)

If the chunk is not mirrored, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

onDatabaseTable

The following list summarizes this table:

Contents:
Information about active databases
Index:
applIndex, **rdbmsDbIndex**
Scope of a row:
One active database
This table does not provide information about an active database if one of the dbspaces for the database is down. (The **rdbmsRelState** MIB object for each database in **rdbmsRelTable** indicates whether a database is active and whether one of its dbspaces is down.)

The table has the following MIB objects.

Table 1. MIB objects for onDatabaseTable

MIB object	Description
applIndex	See applTable.
rdbmsDbIndex	See rdbmsDbTable.
onDatabaseDbspace	Default dbspace
onDatabaseCreated	Creation date and time
onDatabaseLogging	Logging status: • none (1) • buffered (2) • unbuffered (3) • ansi (4)

MIB object	Description
onDatabaseOpenStatus	Database status: • notOpen (1) • open (2) • openExclusive (3)
onDatabaseUsers	Number of users

onDbspaceTable

The following list summarizes this table:

Contents:
Information about dbspaces
Index:
applIndex, **onDbspaceIndex**
Scope of a row:
One dbspace

The table has the following MIB objects.

Table 1. MIB objects for onDbspaceTable

MIB object	Description
applIndex	See applTable.
onDbspaceIndex	Unique integer index for this dbspace. The database server generates this value.
onDbspaceName	Name of the dbspace
onDbspaceOwner	Login name of the owner
onDbspaceCreated	Creation date
onDbspaceChunks	Number of chunks in the dbspace
onDbspaceType	Type of dbspace: • regularDbspace (1) • temporaryDbspace (2) • blobDbspace (3)
onDbspaceMirrorStatus	Mirroring status: • notMirrored (1) • mirrored (2) • mirrorDisabled (3) • newlyMirrored (4)
onDbspaceRecoveryStatus	Recovery status: • noRecoveryNeeded (1) • doneRecovery (2) • physicallyRecovered (3) • logicallyRecovering (4)
onDbspaceBackupStatus	Backup status: • yes (1): The dbspace is backed up. • no (2): The dbspace is not backed up.
onDbspaceMiscStatus	Miscellaneous status: • none (1): no more information • aTableDropped (2)
onDbspacePagesAllocated	Size of all the primary chunks in the dbspace
onDbspacePagesUsed	Number of pages used in all the primary chunks in the dbspace
onDbspaceBackupDate	Date when the latest backup was performed. If the dbspace has never been backed up, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onDbspaceLastBackupLevel	Level of the last backup. If the dbspace has never been backed up, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onDbspaceLastFullBackupDate	Date and time of the last full backup (level 0). If the dbspace has never had a full backup, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

onErQueueTable

The following list summarizes this table:

Contents:
Information about the replication queues for all database servers that participate in Enterprise Replication
Index:
applIndex, **onErQueueReplIndex**
Scope of a row:
One replication queue

The table has the following MIB objects.

Table 1. MIB objects for onErQueueTable

MIB object	Description
applIndex	See applTable.
onErQueueReplIndex	Unique integer index that identifies a replicant
onErQueueSiteIndex	Unique integer that identifies a database server
onErQueueReplName	Display string that describes the replicant or collection of replicants
onErQueueSiteName	Name of the Enterprise Replication database server
onErQueueSize	Current® number of bytes in the send queue
onErQueueLastCommit	Date and time when last transaction was committed
onErQueueLastAck	Date and time when last data was acknowledged

onErSiteTable

The following list summarizes this table:

Contents:
Information about all the remote database servers that participate in Enterprise Replication
Index:
applIndex, **onErSiteIndex**
Scope of a row:
A single replication queue

The table has the following MIB objects.

Table 1. MIB objects for onErSiteTable

MIB object	Description
applIndex	See applTable.
onErSiteIndex	Integer that uniquely identifies a database server as defined in the group entry in **sqlhosts**
onErSiteName	Name of the replication site
onErSiteState	State of the replication activity for this site: • inactive (1) • active (2) • suspend (3) • quiescent (4) • hold (5) • delete (6) • failed (7) • unknown (8)
onErSiteConnectionState	State of the connection to this site: • idle (1) • connected (2) • disconnected (3) • timeout (4) • shutdown (5) • error (6) • unknown (7)
onErSiteConnectionChange	Date and time when the connection state last changed
onErSiteIdleTimeout	Time limit for Enterprise Replication to wait for new data to send or receive. Value is set when database server is defined. Connection is closed if time limit is exceeded.
onErSiteOutMsgs	Total number of messages transmitted from the current database server to this site

MIB object	Description
onErSiteOutBytes	Total number of bytes transmitted from the current database server to this site
onErSiteInMsgs	Total number of messages received by the current database server from this site
onErSiteInBytes	Total number of bytes received by the current database server from this site
onErSiteTransactions	Total number of transactions received from this site
onErSiteCommits	Total number of transactions received and committed from this site
onErSiteAborts	Total number of transactions aborted from this site
onErSiteLastReceived	Date and time when the last transaction was processed from this site
onErSiteRowCommits	Total number of rows committed from this site
onErSiteRowAborts	Total number of rows aborted from this site
onErSiteRcvLatency	Average latency between the source commit time and target receive time; performance measure of network queueing delay
onErSiteCommitLatency	Average latency between source and target commit time; performance measure of network and database server delay
onErSiteClockErrors	Number of transactions received from this site with a time that is ahead of our current time; indicates system clock synchronization problems

onFragmentTable

The following list summarizes this table:

Contents:
Information about the fragments that are in fragmented database tables
Index:
applIndex, **rdbmsDbIndex**, **onTableIndex**, **onFragmentIndex**
Scope of a row:
One fragment of a fragmented database table

The table has the following MIB objects.

Table 1. MIB objects for onFragmentTable

MIB object	Description
applIndex	See applTable.
rdbmsDbIndex	See rdbmsDbTable.
onTableIndex	See onDbspaceTable.
onFragmentIndex	Unique integer index for the fragment
onFragmentType	Type of database table: • fragmentedIndex (1) • fragmentedTable (2)
onFragmentDbspace	Dbspace name for the fragment
onFragmentExpression	Expression text used for fragmentation of the table or index This value is blank if the fragmentation scheme is round-robin.
onFragmentIndexName	Index identifier
onFragmentExtents	Number of extents used
onFragmentPagesAllocated	Total (extent) size allocated to the fragment, in pages
onFragmentPagesUsed	Number of pages used
onFragmentIsamReads	Number of reads from the fragment If the fragment is not active, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onFragmentIsamWrites	Number of writes to the fragment If the fragment is not active, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onFragmentUsers	Number of user threads that access the fragment.
onFragmentLockRequests	Number of locks of any type requested for this fragment.
onFragmentLockWaits	Number of times an initial lock request failed because the lock could not be granted initially for the fragment.
onFragmentLockTimeouts	Number of deadlock timeouts for the fragment.

onLockTable

The following list summarizes this table:

Contents:

Information about the active locks that database servers are using

Index:

applIndex, **onSessionIndex**, **onLockIndex**

Scope of a row:

One lock

A row exists for each lock that the session is using and for each lock on which the session is waiting.

The table has the following MIB objects.

Table 1. MIB objects for onLockTable

MIB object	Description
applIndex	See applTable.
onSessionIndex	See onServerTable.
onLockIndex	Index to this row
onLockDatabaseName	Name of the database that is using or waiting for this lock
onLockTableName	Name of the table that is using or waiting for this lock
onLockType	Type of the lock: • byte (1) • intentShared (2) • shared (3) • sharedByRepeatableRead (4) • update (5) • intentExclusive (6) • sharedIntentExclusive (7) • exclusive (8) • exclusiveByRepeatableRead (9) • waiting (10)
onLockGranularity	Granularity of the lock: • table (1) • page (2) • row (3) • index (4)
onLockRowId	rowid of the locked row
onLockWaiters	Number of sessions that are waiting for the lock
onLockGrantTime	Time when the lock was granted if the session is using the lock If no transaction exists, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

onLogicalLogTable

The following list summarizes this table:

Contents:

Information about logical logs

Index:

applIndex, **onLogicalLogIndex**

Scope of a row:

One logical log

The table has the following MIB objects.

Table 1. MIB objects for onLogicalLogTable

MIB Object	Description
applIndex	See applTable.
onLogicalLogIndex	Index for the logical-log file
onLogicalLogID	Unique integer identification number for the logical-log file
onLogicalLogDbspace	Dbspace name where the log file was created
onLogicalLogStatus	Status of the logical-log file: • newlyAdded (1) • free (2) • current (3) • used (4) • backedUpButNeeded (5)

MIB Object	Description
onLogicalLogContainsLastCheckpoint	Checkpoint status: • yes (1): The logical-log file contains the last checkpoint. • no (2): The logical-log file does not contain the last checkpoint.
onLogicalLogIsTemporary	Temporary status: • yes (1): The logical-log file is temporary. • no (2): The logical-log file is not temporary.
onLogicalLogPagesAllocated	Size of the logical-log file, in pages
onLogicalLogPagesUsed	Number of pages used in the logical-log file
onLogicalLogFillTime	Date and time when the logical-log file last filled up If the log file has never been full, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onLogicalLogTimeUniqueIDChanged	Time stamp when a new unique ID was assigned to this logical-log entry
onLogicalLogTimeLastBackupDate	Date and time of the last backup for this logical-log entry

onPhysicalLogTable

The following list summarizes this table:

Contents:
Information about physical logs
Index:
applIndex
Scope of a row:
One physical log

The table has the following MIB objects.

Table 1. MIB objects for onPhysicalLogTable

MIB object	Description
applIndex	See applTable.
onPhysicalLogDbspace	Dbspace name where the physical log was created
onPhysicalLogBufferSize	Size of the physical-log buffer, in pages
onPhysicalLogBufferUsed	Number of pages of the physical-log buffer that are used
onPhysicalLogPageWrites	Number of pages written to the physical log
onPhysicalLogWrites	Number of (disk) writes to the physical log
onPhysicalLogPagesAllocated	Size of the physical log, in pages
onPhysicalLogPagesUsed	Number of pages used

onServerTable

The following list summarizes this table:

Contents:
Status and profile information about each active database server
Index:
applIndex
Scope of a row:
One database server

The table has the following MIB objects.

Table 1. MIB objects for onServerTable

MIB Object	Description
applIndex	See applTable

MIB Object	Description
onServerMode	Mode of the database server: • initializing (1) • quiescent (2) • fastRecovery (3) • backingUp (4) • shuttingDown (5) • online (6) • aborting (7) • onlineReadOnly (8)
onServerCheckpointInProgress	Checkpoint status: • yes (1): A checkpoint is in progress. • no (2): A checkpoint is not in progress.
onServerPageSize	Size of a page, in bytes
onServerThreads	Number of active threads
onServerVPs	Number of virtual processors
onServerVirtualMemory	Total virtual memory used, in kilobytes
onServerResidentMemory	Total resident memory used, in kilobytes
onServerMessageMemory	Total message memory used, in kilobytes
onServerIsamCalls	Sum of all reads, writes, rewrites, deletes, commits, and rollbacks to and from the database table
onServerLatchWaits	Number of latch waits
onServerLockRequests	Number of lock requests
onServerLockWaits	Number of lock waits
onServerBufferWaits	Number of buffer waits
onServerCheckpointWaits	Number of checkpoint waits
onServerDeadLocks	Number of deadlocks
onServerLockTimeouts	Number of deadlock time outs
onServerLogicalLogRecords	Number of logical-log records
onServerLogicalLogPageWrites	Number of logical-log page writes
onServerLogicalLogWrites	Number of logical-log writes
onServerBufferFlushes	Number of buffer flushes
onServerForegroundWrites	Number of foreground writes
onServerLRUWrites	Number of LRU writes
onServerChunkWrites	Number of chunk writes
onServerReadAheadPages	Number of read-ahead pages This value includes data and index read-ahead pages.
onServerReadAheadPagesUsed	Number of read-ahead pages used
onServerSequentialScans	Number of sequential scans
onServerMemorySorts	Number of memory sorts
onServerDiskSorts	Number of disk sorts
onServerMaxSortSpace	Maximum disk space that a sort uses, in pages
onServerNetworkReads	Number of network reads
onServerNetworkWrites	Number of network writes
onServerPDQCalls	Number of parallel-processing actions performed
onServerTransactionCommits	Number of committed transactions
onServerTransactionRollbacks	Number of rolled-back transactions
onServerTimeSinceLastCheckpoint	Length of time since the last checkpoint, in hundredths of second
onServerCPUSystemTime	Amount of CPU time that the database server has used in System Mode, in hundredths of second
onServerCPUUserTime	Amount of CPU time that the database server has used in User Mode, in hundredths of second

onSessionTable

The following list summarizes this table:

Contents:

Information about each session

Index:

applIndex, **onSessionIndex**

Scope of a row:

One session

The table has the following MIB objects.

Table 1. MIB objects for onSessionTable

MIB Object	Description
applIndex	See applTable.
onSessionIndex	Unique integer index for the session
onSessionUserName	Name of the user, in the form name@host(tty)
onSessionUserProgramVersion	Version of the database server
onSessionUserProcessId	Process ID for the session
onSessionUserTime	Length of time that the user has been connected to the database server, in hundredths of seconds
onSessionState	State of the session: • idle (1) • active (2) • waitingOnMutex (3) • waitingOnCondition (4) • waitingOnLock (5) • waitingOnBuffer (6) • waitingOnCheckPointing (7) • waitingOnLogicalLogWrite (8) • waitingOnTransaction (9)
onSessionDatabase	Connected database
onSessionCurrentMemory	Memory usage, in bytes
onSessionThreads	Number of active threads
onSessionCurrentStack	Average size of the stack for all threads
onSessionHighwaterStack	Maximum amount of memory that any thread has used so far
onSessionLockRequests	Number of lock requests
onSessionLocksHeld	Number of locks held
onSessionLockWaits	Number of lock waits
onSessionLockTimeouts	Number of timeouts for locks
onSessionLogRecords	Number of log records
onSessionIsamReads	Number of reads from database tables
onSessionIsamWrites	Number of writes to database tables
onSessionPageReads	Number of page reads
onSessionPageWrites	Number of page writes
onSessionLongTxs	Number of long transactions
onSessionLogSpace	Logical-log space used, in bytes
onSessionHighwaterLogSpace	Maximum logical-log space that this session has ever used
onSessionSqlStatement	Latest SQL statement, truncated to 250 characters if necessary
onSessionSqlIsolation	SQL isolation level: • noTransactions (1) • dirtyReads (2) • readCommitted (3) • cursorRecordLocked (4) • repeatableRead (5)
onSessionSqlLockWaitMode	Action to take if the isolation level requires a wait: • `-1` = Wait forever. • `0` = Do not wait. • `>0` = Wait for specified number of seconds.
onSessionSqlEstimatedCost	Estimated cost of the SQL statement according to SQLEXPLAIN
onSessionSqlEstimatedRows	Estimated number of rows that the SQL statement selects according to SET EXPLAIN
onSessionSqlError	Error number for the last SQL statement
onSessionSqlIsamError	ISAM error number for the last SQL statement

MIB Object	Description
onSessionTransactionStatus	Status of the transaction: • none (1) • committing (2) • rollingBack (3) • rollingHeuristically (4) • waiting (5)
onSessionTransactionBeginLog	Unique ID of the logical-log file in which the BEGIN WORK record was logged If no transaction exists, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onSessionTransactionLastLog	Unique ID of the logical-log file in which the last record was logged If no transaction exists, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).
onSessionOriginatingSessionId	Local session ID of the global session on the server for which this local session runs

onSqlHostTable

The following list summarizes this table:

Contents:
Copy of the connection information
Index:
applIndex, **onSqlHostIndex**
Scope of a row:
One connectivity value

As the following table shows, the location of the connection information depends on the operating system.

Table 1. Location of connection information

Operating system	Location of connectivity information
UNIX	The INFORMIXSQLHOSTS environment variable specifies the full path name and file name of the connection information. The default location is $INFORMIXDIR/etc/sqlhosts. For information about INFORMIXSQLHOSTS, see the *IBM Informix Guide to SQL: Reference*.
Windows	The connectivity information is in a key in the Windows registry called HKEY_LOCAL_MACHINE\SOFTWARE\Informix\SQLHOSTS.

For details about the connection information, see your *IBM Informix Administrator's Guide*.

The table has the following MIB objects.

Table 2. MIB objects for onSqlHostTable

MIB object	Description
applIndex	See applTable.
onSqlHostIndex	Index to the entry in the connectivity information
onSqlHostName	Host name of the database server
onSqlHostNetType	Connection type
onSqlHostServerName	Name of the database server or its alias
onSqlHostServiceName	Service name
onSqlHostOptions	List server options in the form of key=value pairs

onTableTable

The following list summarizes this table:

Contents:
Information about a database table
Index:
applIndex, **rdbmsDbIndex**, **onTableIndex**
Scope of a row:
One database table

For a fragmented database table, the values in this table are summaries of the values from all the database table fragments. The table has the following MIB objects.

Table 1. MIB objects for onTableTable

MIB object	Description
applIndex	See applTable.
rdbmsDbIndex	See rdbmsDbTable.
onTableIndex	Table number This value is the same as **tabid** in the system catalog table **systables**

MIB object	Description
onTableName	Table name
onTableOwner	Table owner
onTableType	Type of table: • table (1) • view (2) • privateSynonyn (3) • synonym (4)
onTableLockLevel	Locking level of the table: • page (1) • row (2)
onTableCreated	Creation date, in string format
onTableFirstDbspace	Name of the first (or only) dbspace for the table
onTableRowSize	Length of a row
onTableRows	Number of rows
onTableColumns	Number of columns
onTableIndices	Number of indexes
onTableExtents	Number of extents in use
onTablePagesAllocated	Total (extent) size allocated to the table, in pages
onTablePagesUsed	Number of pages in use
onTableFragments	Number of fragments
onTableFragmentStrategy	Fragmentation strategy: • roundRobin (1) • byExpression (2) • tableBased (3)

If the table is not fragmented, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

onTableActiveFragments
Number of active fragments
If the table is not fragmented, this value is noSuchInstance (SNMPv2) or noSuchName (SNMPv1).

InformixHQ Guide

InformixHQ is a modern web console for visualizing, monitoring, and managing your Informix server instances. It is purpose built for ease-of-use, scaling out, and optimizing DevOps needs. It provides critical performance management capabilities, monitoring how key performance metrics are changing over time and tracking how efficiently Informix is running your workload even when you've stepped away from your screen. Its monitoring system feeds directly into a customizable alerting system so you can be immediately alerted via email, Twilio, or PagerDuty whenever an issue occurs on one of your Informix database server instances. InformixHQ is designed to be scalable to efficiently manage and monitor as many Informix database server instances as you need. Moreover, it's a tool that can be shared by the DBAs, the app developers, the ops engineers, and management and accessed from any desktop, laptop, or mobile device. InformixHQ is the centralized hub for graphical monitoring, alerting, and administration of your Informix database servers.

- What's new in InformixHQ
  This topic includes information about new features in InformixHQ.
- Architecture
  InformixHQ consists of three distinct pieces that come together to give you a comprehensive monitoring and administering experience for Informix.
- System Compatibility
  Before you install InformixHQ, make sure that your computer meets the system requirements.
- Getting Started
  This topic provides a brief tutorial to help you get started with InformixHQ.
- InformixHQ Concepts
  This topic covers some of the conceptual aspects of InformixHQ.
- InformixHQ Server
- InformixHQ Agent
- Frequently asked questions (FAQs) about InformixHQ
  These topics provide short answers to some frequently asked questions about InformixHQ.

What's new in InformixHQ

This topic includes information about new features in InformixHQ.

What's new in InformixHQ 14.10.xC6

What's new in InformixHQ 14.10.xC5

What's new in InformixHQ 14.10.xC4

What's new in InformixHQ 14.10.xC2

What's new in InformixHQ 14.10.xC6

Administration

- The **Schema Manager** page has been enhanced to:
  - Create and Drop Table with advanced options

  For more information, see *Schema Manager*.

- The InformixHQ server and agent now use the log4j2 library for logging. By default, the InformixHQ server and agent will log messages at INFO level to an informixhq-server.log file and an informixhq-agent.log file respectively. You can customize the logging behavior by providing a server.log4j.xml when starting the InformixHQ server or an agent.log4j.xml file when starting the InformixHQ agent in the current directory or classpath. Use these log4j2 configuration files to change the logging level (ERROR, WARN, INFO, or DEBUG), change the log file location, or enable rolling window logging.

  For more information, see Logging in InformixHQ .

Ease of use

- Provision is made for the user to specify connection properties for InformixHQ agent separately.
  For more information, see *InformixHQ Agent Setup*.

What's new in InformixHQ 14.10.xC5

Administration

- Enhanced to support stronger hash algorithm for passwords by adding a new config property **user.password.algorithm**.
  For more information, see *InformixHQ Server Configuration*.

Customization

- Customized to save dashboard preferences in Custom Dashboard page
  For more information, see *Custom Dashboards*.

Ease of use

- Script to start or stop the server has been enhanced to support AIX platform.
  For more information, see *Getting Started*.

What's new in InformixHQ 14.10.xC4

Administration

- The **Schema Manager** page has been enhanced to:
  - Browse and view detailed information about database information like Stored Procedures, Sequences, User Defined Types, Data Blades
  - Browse and view detailed information about table information like Indexes, References, Constraints, Triggers
  - Create Database and Drop Database
  - Create Demo Database and Drop Demo Database
  - Create, Enable, Disable and Drop Index with advanced options

  For more information, see *Schema Manager*.

- The **Connection Manager** page allows you to visualize and manage CM unit, SLA and FOC for any CM.
  For more information, see *Connection Manager*.

Ease of use

- New script has been added to start or stop the server which is available in ${INFORMIXDIR}/HQ directory.
  For more information, see *Getting Started*.

What's new in InformixHQ 14.10.xC2

Administration

- The **Schema Manager** page allows you to browse and view detailed information about the various tables and indexes in each of your databases.
- The **Storage > Tables and Indexes** page allows you to analyze the storage characteristics of the tables and indexes in each of your databases, perform storage optimization actions such as compress, shrink, repack, defragment, and manage your automatic storage optimization policies.
- The **High Availability** page makes it easy to visualize and monitor the functioning of your high availability cluster.
- The **Enterprise Replication** page visualizes and provides details drill-down statistics about each Informix node participating in replication.
- The **Auto Update Statistics** pages allows you to manage your automatic update statistics policies, ensuring your queries continue to run efficiently as your data changes over time.
- The **Privileges** page allows you to manage privileges on your database server, including database level privileges, table level privileges, SQL Admin API privileges, and internal users.
- The **System Reports** page provides a full set of detailed reports on various aspects of your database server's performance.
- The **Task Scheduler** pages allows you to manage and customize tasks for your database server.
- The **Memory Manager** page allow you to visualize and monitor your database server's memory usage as well as configure its Low Memory Manager configuration.
- The **Backups** page has been enhanced to display a history timeline of your most recent database server backups.

Ease of use

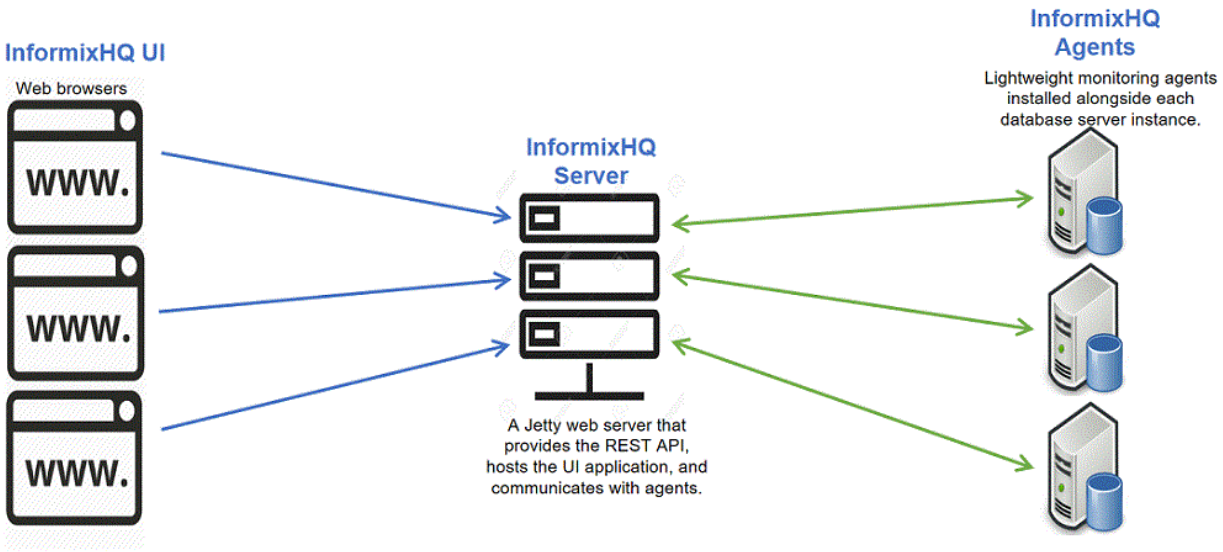
- **Centralized user permission management**, on the redesigned **System Settings > User Management** page, allows you to view and manage user's permissions within InformixHQ in a single place.
- The **group incidents** page allows you to view all the alerting incidents that occurs on an entire group of Informix servers from one centralized page.
- The **new logging framework** allows not only for a better out of the box logging experience, but also provides enhanced options for logging customization.
  For more information, see *Logging in InformixHQ*.

Customization

- Build **custom dashboards** for a single database server or for multiple database servers, configuring what monitoring data is displayed on the dashboard and how that data is arranged and visualized. Custom dashboards allow you to define UI pages that show you the Informix monitoring data that is most important to you.
  For more information, see *Custom Dashboards*.
- Create your own **custom SQL sensors** for monitoring your Informix instances. Any sysmaster SQL query can now be turned into a sensor that will integrate directly into InformixHQ's monitoring and alerting systems.
  For more information, see *Custom SQL Sensors*.
- A new **extensible alerting** option allows you to define a custom script to be executed whenever an alerting incident occurs.
  For more information, see *Alerting incident*.

Architecture

InformixHQ consists of three distinct pieces that come together to give you a comprehensive monitoring and administering experience for Informix.



InformixHQ Server

- Java 8 based Jetty web server
- Monitors and administers many Informix database servers
- Connects directly to Informix databases servers to
  - Gather live data from the system
  - Perform administration
- Connects to the InformixHQ agents regarding
  - Monitored data
  - Alerts/events

InformixHQ Agent

- Lightweight Java 8 based monitoring agent
- Installed alongside each of your Informix database instances
- Only needs read access to database server
- Can perform native command execution to gather OS statistics as well as database statistics

InformixHQ User Interface (UI)

- Modernized web UI for monitoring, managing, visualizing, and assessing your Informix database servers

System Compatibility

Before you install InformixHQ, make sure that your computer meets the system requirements.

InformixHQ Prerequisites

The following table lists the software prerequisites for InformixHQ.

Software	Required Version
Informix Database Server	12.10 or higher
Java	1.8

Note:
In Debian environment, InformixHQ requires "haveged" service to be up and running. To start the service, follow these steps:

- sudo apt-get install haveged
- update-rc.d haveged defaults
- service haveged start

Supported Web Browsers

InformixHQ supports all the latest browsers. Following table lists the web browser that InformixHQ has been tested with.

Web Browser	Version
Google Chrome	81
Mozilla Firefox	74
Microsoft Edge	80

Getting Started

This topic provides a brief tutorial to help you get started with InformixHQ.

Prerequisites

The following table lists the software prerequisites for InformixHQ.

Software	Required Version
Informix Database Server	12.10 or higher
Java	1.8

Starting InformixHQ

InformixHQ Server or Agent can be started using any of following methods:

- Java Command
- Script

Using Java Command

For more information about using Java Command, see Starting the InformixHQ Server and Starting the InformixHQ Agent.

Using the Script:

The script to start the server or agent is in the form of *InformixHQ.bat* (Windows), *InformixHQ.sh* (Bash Shell for Linux) and *InformixHQ.ksh* (Korn shell for AIX) file on various operating system as per the requirement of the user.
Note: This script is indicative and users are free to modify/tune the scripts as per their requirements/environments.
Script Prerequisites

- Server and Agent jar file names should end with .jar and also must contain keyword "*informixhq*" in its name
- If using default filename option with scripts, jar and properties files with default names should be present in same folder as that of script
- User running the script must have read and write access to informixhq-server.log and informixhq-agent.log files
- If log files don't exist in the same folder from where HQ jar is running, HQ will create these files, so user running the script should have permissions to be able to create these files.
- Environment variable JAVA_HOME should be set correctly to JAVA 1.8 and should be included in PATH variable.

This script support BASH shell on linux (.sh), KORN shell on AIX (.ksh) and command prompt on windows (.bat). This command starts/stops InformixHQ Server and Agent.

Syntax:

```
InformixHQ [startserver|startagent] [encoding=<value>] [jvmmemx=<value>] [jarfile=<value>] [propfile=<value>]
    InformixHQ [stop <processid>]
    InformixHQ [list]

    startserver           : Starts InformixHQ Server service
    startagent            : Starts InformixHQ Agent service
    stop                  : Stops InformixHQ Server/Agent with processid
    list                  : Lists InformixHQ running processes
    encoding (Optional)   : Default value is utf-8
    jvmmemx (Optional)    : JVM's default value will be used If not specified
    jarfile (Optional)    : Default is informixhq-server.jar for startServer option
                            and informixhq-agent.jar for startAgent option
                            For user provided filename, it must contain keyword
                           'informixhq' and it should end with .jar"
    propfile (Optional)   : Default is informixhq-server.properties for
                            startServer option and informixhq-agent.properties for startAgent option
                            User can provide any custom name to properties file
                            JVMMemx value in m represents MB & g represents GB.
```

For more information about using the script, see Starting the InformixHQ Server and the Agent on Windows and Starting the InformixHQ Server and the Agent on Linux/AIX.

- Starting the InformixHQ Server
  This topic provides a brief tutorial to help you get started with InformixHQ Server.
- Starting the InformixHQ Agent
  This topic covers the two ways of starting the InformixHQ Agent.
- Starting the InformixHQ Server and the Agent on Windows
  This topic provides a brief tutorial about the script used to get started with InformixHQ Server and Agent on Windows.

- Starting the InformixHQ Server and the Agent on Linux/AIX
  This topic provides a brief tutorial about the script used to get started with InformixHQ Server and Agent on Linux/AIX.
- Logging in InformixHQ
  This topic provides a brief tutorial on logging in InformixHQ.

Starting the InformixHQ Server

This topic provides a brief tutorial to help you get started with InformixHQ Server.

1. Locate the *informixhq-server.jar* and the *server.log4j.xml* file in the $INFORMIXDIR/hq directory of your Informix database server installation.
2. Create an InformixHQ server configuration file. You can refer to the $INFORMIXDIR/hq/informixhq-server-example.properties file as an example.
   The InformixHQ server configuration file must contain an initialAdminPassword . All other configuration properties are optional.

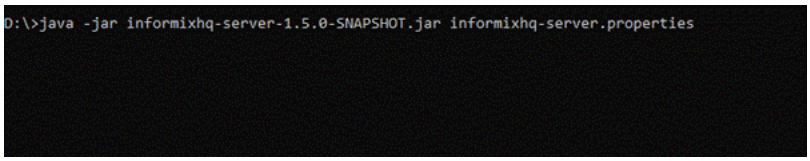
   Sample configuration file

   ```
   # required initial password for the admin user
   initialAdminPassword=myPassword123
   # optionally, uncomment these properties to have InformixHQ encrypt its internal H2
   database
   #h2.encrypt.enable=true
   #h2.encrypt.password=password
   ```

   Note: The **initialAdminPassword** property is only required the very first time you start the InformixHQ server, when it performs its start-up initialization and creates the first user named **admin**. Afterwards, you can remove the **initialAdminPassword** property from the **informixhq-server.properties** file.
   Note: Password must be at least 8 characters and must contain at least one lowercase character, one uppercase character, and one number.
3. Optionally, edit the *server.log4j.xml* file to configure logging in the InformixHQ.
   Note: By default InformixHQ runs on 8080 port. For more information, see InformixHQ Server Configuration.
4. Start the InformixHQ server using the following command:

   ```
   java -jar informixhq-server.jar informixhq-server.properties
   ```

   where **informixhq-server.properties** is the name of the InformixHQ server configuration file.
   Note: Staring from 14.10.xC4 onwards, you can start the server using an alternate method. For more information on this, see Starting the InformixHQ Server and the Agent on Windows and Starting the InformixHQ Server and the Agent on Linux.
   Sample output after starting the InformixHQ server with server.log4j.xml:

   Sample output after starting the InformixHQ server without server.log4j.xml:

   ```
   D:\>java -jar informixhq-server-1.5.0-SNAPSHOT.jar informixhq-server.properties
   2021-05-12 17:58:28,305 main WARN Posix file attribute permissions defined but it is not supported by this files syste
   m.
   2021-05-12 17:58:28,305 main INFO Log4j appears to be running in a Servlet environment, but there's no log4j-web modul
   e available. If you want better web container support, please add the log4j-web JAR to your web archive or server lib
   directory.
   2021-05-12 17:58:28,320 main INFO Log4j appears to be running in a Servlet environment, but there's no log4j-web modul
   e available. If you want better web container support, please add the log4j-web JAR to your web archive or server lib
   directory.
   ```

   When the InformixHQ server is started for the first time, a user with system administrator privileges for InformixHQ is created with the username **admin** and the password as specified in the **initialAdminPassword** property from your configuration file.
5. Using a web browser, go to the InformixHQ UI at http://localhost:8080/ and login with the user **admin** and the password specified in your configuration file.
6. Once logged in, click **Add Server** to add an Informix server that you want to monitor.

Starting the InformixHQ Agent

This topic covers the two ways of starting the InformixHQ Agent.

To get the most out of InformixHQ, you should have an InformixHQ agent running for each Informix database server that you will be monitoring through the tool. While the InformixHQ agent is not required to view information about your database server in the InformixHQ UI, the agent is required if you want to gather monitoring data and configure alerts for that server.

There are two options for starting the InformixHQ agent. You can use the InformixHQ UI to automatically deploy and start the InformixHQ agent or you can manually start the InformixHQ agent on the command line.

Deploying and starting the InformixHQ agent automatically from the UI:

Before you deploy and start the InformixHQ agent automatically from the UI, you must meet the following prerequisites:

Prerequisites

1. SSH must be installed on the database server's host machine.
2. The *informixhq-agent.jar* file and the *informixhq-server.jar* file must be located in the same directory. For customized logging, the *agent.log4j.xml* file must also be included in the same directory.

To deploy and start the InformixHQ agent automatically from the UI:

1. The Informix database server that the agent will be monitoring must first be defined in InformixHQ. If the database server has not been defined yet, in the UI, navigate to an InformixHQ group, click **Add Server** and define the server's connection properties.
2. Navigate to the Informix database server's **Setup** > **Agent** page and define a repository server and database. The repository database is where the monitored data will be stored.
3. From the server's **Setup > Agent** page, you can also try to deploy the agent with default configuration by clicking the **Deploy Agent** button. If the automatic deployment is successful, you are done. Otherwise, proceed to the next step.
4. Configure agent deployment:
   - Username: remote user which will own and run the agent
   - Password or Identity File/Passphrase: remote user's passphrase or remote user's identity file (e.g. private key) and its optional passphrase.
   - Remote directory: Directory to deploy the agent files to. This directory will be created if it does not exist.
5. If SSL is enabled, keystore configuration may be required as well. If keystore configuration is not provided, InformixHQ will try to generate and deploy a keystore for you.
   - Keystore file: Location of an existing keystore on the remote machine (can be relative to the remote directory)
   - Keystore password: Password used to access the existing keystore
   - Keystore type: Existing keystore's type. Default is JKS
6. Click **Deploy Agent**
7. Once the agent is ready, use the InformixHQ UI in your web browser to configure the monitoring profile and alerts for this server.

Starting an InformixHQ agent manually on the command line:

1. The Informix database server that the agent will be monitoring must first be defined in InformixHQ. If the database server has not been defined yet, in the UI, navigate to an InformixHQ group, click **Add Server** and define the server's connection properties.
2. Navigate to the Informix database server's **Setup > Agent** page and define a repository server and database. The repository database is where the monitored data will be stored
3. Locate the *informixhq-agent.jar* and the *agent.log4j.xml* file in the $INFORMIXDIR/hq directory of the InformixHQ database server installation.
   Note: Staring from 14.10.xC4 onwards, you can start the agent using an alternate method. For more information on this, see and Starting the InformixHQ Server and the Agent on Windows and Starting the InformixHQ Server and the Agent on Linux.
4. Copy the agent jar file and the log4j2 configuration file to the Informix database server host machine.
5. Create an agent configuration file.
   Sample agent configuration file:

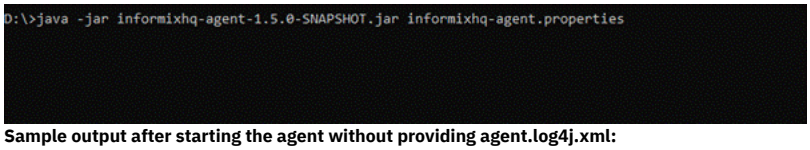
```
# host and port of the InformixHQ server
server.host=localhost
server.port=8080

# The id of the Informix database server as defined in InformixHQ
informixServer.id=1
```

   Note: You can find the id of your Informix database server by navigating to the server's **Setup** page in the UI.
6. Optionally, edit the *agent.log4j.xml* file to configure logging in the InformixHQ. If agent.log4j.xml is not provided, default logging configuration is set and it gives warning as shown below.
7. Start the InformixHQ agent using the following command

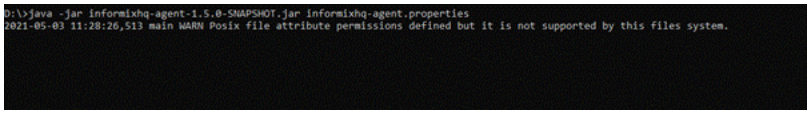
```
java -jar informixhq-agent.jar agent.properties
```

   where **agent.properties** is the name of your InformixHQ agent configuration file.

   Sample output after starting the agent with agent.log4j.xml:



   Sample output after starting the agent without providing agent.log4j.xml:



8. At this point the agent is ready and running. Use the InformixHQ UI in your web browser to configure the monitoring profile and alerts for this server.

Logging in InformixHQ

This topic provides a brief tutorial on logging in InformixHQ.

The InformixHQ server and agent use the log4j2 library for logging. By default, the InformixHQ server and agent will log messages at INFO level to an *informixhq-server.log* file and an *informixhq-agent.log* file respectively.

You can customize the logging behavior by providing a *server.log4j.xml* file in the current directory or classpath when starting the InformixHQ server or a *agent.log4j.xml* file in the current directory or classpath when starting the InformixHQ agent. Use these log4j2 configuration files to change the logging level (ERROR, WARN, INFO, or DEBUG), change the log file location, or enable rolling window logging. For more information, see the log4j2 documentation.

Sample log4j2 configuration files are provided in $INFOMRIXDIR/hq/example-server.log4j.xml and $INFORMIXDIR/hq/example-agent.log4j.xml.

ConsoleAppender

The ConsoleAppender writes its output to either System.out or System.err with System.out being the default target. A Layout must be provided to format the LogEvent.

RollingFile Appender

The RollingFileAppender is an OutputStreamAppender that writes to the file named in the fileName parameter and rolls the file over according to the TriggeringPolicy and the RolloverPolicy.

The CompositeTriggeringPolicy takes multiple triggering policies and returns true if any of the configured policies return true. The CompositeTriggeringPolicy is configured simply by combining other policies in a Policies element.

SizeBased Triggering Policy

Once the file reaches the specified size, the SizeBasedTriggeringPolicy causes a rollover. The size can be specified in bytes, with the suffix KB, MB or GB, for example 20MB. When combined with a time based triggering policy, the file pattern must contain a %i otherwise the target file will be overwritten on every rollover as the SizeBased Triggering Policy will not cause the timestamp value in the file name to change. When used without a time based triggering policy, the SizeBased Triggering Policy will cause the timestamp value to change.

TimeBased Triggering Policy

The TimeBasedTriggeringPolicy causes a rollover once the date/time pattern no longer applies to the active file. This policy accepts an interval attribute which indicates how frequently the rollover should occur based on the time pattern and a modulate boolean attribute.

Default Rollover Policy

The default rollover takes both date/time pattern and an integer specified in filePattern Attribute in RollingFileAppender. If the pattern contains an integer, it will be incremented on every rollover. If the date/time pattern is present, it will be replaced with current date and time values. If the file pattern ends with ".gz", ".zip", ".bz2", ".deflate", ".pack200", or ".xz", the resulting archive will be compressed using the compression scheme that matches the suffix.

This example shows a rollover strategy that will keep up to 20 files before removing them.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
   <Appenders>
      <RollingFile name="RollingFile" fileName="logs/app.log" filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz">
         <PatternLayout>
            <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
         </PatternLayout>
         <Policies>
            <TimeBasedTriggeringPolicy />
            <SizeBasedTriggeringPolicy size="250 MB" />
         </Policies>
         <DefaultRolloverStrategy max="20" />
      </RollingFile>
   </Appenders>
   <Loggers>
      <Root level="error">
         <AppenderRef ref="RollingFile" />
      </Root>
   </Loggers>
</Configuration>
```


InformixHQ Concepts

This topic covers some of the conceptual aspects of InformixHQ.

Group

InformixHQ provides the ability to create groups of servers to make them easier to manage and monitor. InformixHQ's groups are based on a hierarchy. The base “root” group is the top level group for all InformixHQ groups and servers. From this root group, you can add as many servers and sub-groups as you desire, nesting them to whatever level makes sense for your organization.

You can define monitoring and alerting profiles for groups, simplifying the task of managing monitoring for all of your database servers.

Agent

The InformixHQ agent is a lightweight Java based program that is designed to run alongside each Informix database server, gathering data about the performance of the system. The data gathered by the agent is fully configurable and is defined by the list of **sensors** in the server's **monitoring profile**. The data gathered by the agent is stored in a **repository database**.

Repository Database

A repository database holds information collected by the InformixHQ agent about the Informix database server. The repository database can either be local to the database server that is being monitored or it can be on a remote Informix instance. You can define a common repository database shared by multiple Informix database server instances, or you can define a separate repository database for each monitored instance.

The repository database must be an Informix database and must exist before the agent is started. You must define the database server to be used as a repository in the InformixHQ UI, using the **Add Server** action on any group dashboard, before it can be defined as a repository.

To define a repository database for a particular Informix database server, go to the server's **Setup** page in the UI and click on the **Agent** tab.

Sensor

A sensor defines a metric or set of metrics for the agent to gather. An example is the "DBSpace Usage" sensor that gathers metrics on used and free space for all database server spaces.

Monitoring Profile

A monitoring profile defines the list of sensors that the agent runs to gather data about and Informix database server instance or about its host operating system.

For each sensor in a monitoring profile, you can configure the frequency at which that sensor will run and how long that sensor's data will be kept in the repository database.

Monitoring profiles can be configured for groups as well as servers. InformixHQ uses the concept of inheritance for determining a particular server's or group's monitoring profile. All servers and groups inherit monitoring profile information from its parent group in the hierarchy. Servers or groups can also disable or override the configuration of any sensors inherited from a parent group.

Alert

An alert defines a condition that should trigger an alerting incident in InformixHQ. An example would be an alert defined for when the Informix database server goes offline.

Alerting profile

An alerting profile defines the set of alerts configured for a particular server or group. Alerting profiles follow an inheritance model similar to monitoring profiles.

Alerting incident

While the monitoring data is collected by the InformixHQ agent, it is the InformixHQ server that is tasked with evaluating alerting conditions. As data is collected by the agent, the InformixHQ server evaluates each new incoming data point against the alert definitions in the server's alerting profile. Any data point that meets an alerting condition triggers an alerting incident.

Alerting incidents are shown in the InformixHQ UI for that server. Alerting incident counts are also aggregated and highlighted on the group dashboards. Alerting incidents can be acknowledged as read or even deleted from the Incidents page in the UI. Users of InformixHQ can configure their own alerting preferences to automatically receive alerting incidents directly via email, Twilio, or PagerDuty.

InformixHQ Server

The InformixHQ server is a Java 8 based Jetty web server. The server is the heart of InformixHQ. It manages the monitoring profiles for all instances, communicates with agents, handles all alerting activities, hosts the web UI, and provides the REST services that the web UI depends on.

- InformixHQ Server Configuration
- InformixHQ UI

InformixHQ Server Configuration

A properties file is required to run the InformixHQ server. When starting the InformixHQ server, you can pass the properties file name as part of the start command. Otherwise, InformixHQ will look for a properties file named informixhq-server.properties in the classpath.

An example configuration file documenting all supported InformixHQ server configuration properties can be found in $INFORMIXDIR/hq/informixhq-server-example.properties.

- Required configuration properties on initial startup
  - initialAdminPassword
- Optional configuration properties
  - alert.numberConditionCheckThreads
  - alert.startNumberAlertSendThreads
  - dataSource.IFX_ISOLATION_LEVEL
  - hostname
  - httpPort
  - httpsPort
  - h2.encrypt.algorithm
  - h2.encrypt.enable
  - h2.encrypt.password
  - pool.connectionTimeout
  - pool.maximumPoolSize
  - pool.minimumIdle
  - pool.idleTimeout
  - redirectHTTPtoHTTPS
  - rest.session.timeout
  - ssl.keystore.file
  - ssl.keystore.password
  - ssl.key.password
  - user.password.algorithm
  - user.password.maxAge
  - user.password.minLength
  - user.password.requireLowerCase
  - user.password.requireNumber
  - user.password.requireSpecialCharacterFromSet
  - user.password.requireUpperCase

initialAdminPassword

When starting the InformixHQ server for the first time, an **admin** user will be created with the password specified in the **initialAdminPassword** property. This user will have system administrative privileges on the InformixHQ server which includes the ability to create other users, grant privileges, and make configuration changes to the server.

This property is only required the very first time you start the InformixHQ server. For security reasons, it is recommended that you remove this property from your InformixHQ server configuration file after the InformixHQ server has been initialized for the first time.

alert.startNumberAlertSendThreads

Configures the number of threads in the thread pool that processes and dispatches alert notifications (by email, Twilio, Pager Duty, etc.) when an alerting incident occurs. The default number of threads is 4.

alert.numberConditionCheckThreads

Configures the number of threads in the thread pool that checks whether alerting conditions have been violated whenever new monitoring data comes in. The default number of threads is 4.

dataSource.IFX_ISOLATION_LEVEL

Specifies the isolation level to set on JDBC connections to the various Informix database servers. The default value is 1.

hostname

The host name of the InformixHQ server. The host name determines the network adapter or interface that the InformixHQ server binds the server socket to.

The default value is an empty string. When set to an empty string, the InformixHQ server will bind to all available network interfaces on the host machine.

httpPort

The HTTP port to run the InformixHQ server on. This port will serve both the InformixHQ web UI and the InformixHQ REST API. Set this value to -1 to disable the HTTP protocol for InformixHQ. If httpPort is set to -1, make sure that httpsPort is set to something other than -1.The default value is 8080.

httpsPort

The HTTPS port to run the InformixHQ server on. This port will serve both the InformixHQ web UI and the InformixHQ REST API.

Set this value to -1 to disable the HTTPS protocol for InformixHQ. If httpsPort is set to -1, make sure that httpPort is set to something other than -1.

If httpsPort is something other than -1, you must set the **ssl.keystore.file** and **ssl.keystore.password** properties, and potentially also the **ssl.key.password** property if your key password is different from the keystore password.

The default value is -1 indicating that HTTPS is disabled by default.

h2.encrypt.algorithm

Sets the algorithm for H2 database file encryption. The encryption algorithms supported by H2 are AES, XTEA, and FOG. The default value is AES.

h2.encrypt.enable

Controls whether the H2 database file which holds InformixHQ server's internal metdata is encrypted. If you set this property to true, you must also set the h2.encrypt.password property. The default value is false.

h2.encrypt.password

Sets the password to use for H2 database file encryption. If h2.encrpyt.enable is set to true, you must set the password for encryption.

pool.connectionTimeout

Specifies the number of milliseconds to wait for a JDBC connection to an Informix database server to be established before it times out. The default value is 5000 (5 seconds).

pool.idleTimeout

Specifies the number of milliseconds that a JDBC connection can be idle in the connection pool before it is closed. The default value is 60000 (1 minute).

pool.maximumPoolSize

The maximum number of JDBC connections in each connection pool. The InformixHQ server will maintain a connection pool for each Informix database that it needs to connect to. The **pool.maximumPoolSize** puts a cap on the total number of open JDBC connections that can be established to each database.

The default value is 5.

pool.minimumIdle

The minimum number of idle JDBC connections in each connection pool. The InformixHQ server will maintain a connection pool for each Informix database server that it needs to connect to. Setting **pool.minimumIdle** to zero indicates that all JDBC connections in the connection pool should be closed when they have been sitting idle for longer than the **pool.idleTimeout** threshold. Setting **pool.minimumIdle** to a positive integer indicates the number of connections that should be kept open in the connection pool even when they exceed the **pool.idleTimeout**. The default and recommended value is 0.

redirectHTTPtoHTTPS

If set to true, HTTP traffic to InformixHQ will automatically be redirected to HTTPS. This will include web socket communication between the InformixHQ server and agent. If this value is set to true, you will be required to configure SSL in your agent configuration properties.

The default value is false.

rest.session.timeout

Specifies the number of milliseconds that a REST session can be idle before it is closed. The default value is 3600000 (60 minutes).

ssl.keystore.file

The path to the keystore file that contains the certificate to use for network encryption. This property must be set if **httpsPort** is set to something other than -1.

ssl.keystore.password

The password to unlock the keystore file for network encryption. If this property is not set and the HTTPS is configured, you will be prompted on the command line to enter the keystore password when starting the InformixHQ server.

ssl.key.password

The password to unlock the entry into the keystore. The default value is no password, which means to use the keystore password. If the entry into the keystore requires a password that is different from the keystore password, set this property to the entry password.

user.password.algorithm

Sets the algorithm for InformixHQ login password. The encryption algorithms supported by InformixHQ are SHA-1, SHA-256, SHA-384, SHA-512. The default value is SHA-256.

user.password.maxAge

Controls the maximum age (in days) of a user password. User passwords that are older than the max age will be considered as expired. Setting this property to zero, which is the default value, specifies that user passwords never expire. Setting this property to a value greater than zero specifies the maximum age (in days) of a user password before it expires. A user will start receiving notifications in the InformixHQ UI when the difference between the current date and the password expiration date is less than or equal to 15 days.

user.password.minLength

Controls the minimum length for a user password. The default value is 8.

user.password.requireLowerCase

Controls whether user passwords are required to include at least one lowercase character. The default value is true.

user.password.requireNumber

Controls whether user passwords are required to include at least one number. The default value is true.

user.password.requireSpecialCharacterFromSet

Controls whether user passwords are required to include at least one special character. An empty string indicates that no special characters are required. Setting this value to "!@#$%^&*()" would require user passwords to include at least one of those characters. The default value is an empty string.

user.password.requireUpperCase

Controls whether user passwords are required to include at least one uppercase character. The default value is true.

InformixHQ UI

This topic highlights important aspects of the InformixHQ user interface. The InformixHQ UI is run by the Jetty web server that is embedded in the InformixHQ server. The default URL for the InformixHQ server is http://localhost:8080/ but the InformixHQ server configuration file can be used to change the port and configure whether https is enabled.

- Adding Servers and Groups
- Exploring Groups
- Exploring Informix Database Servers
- Configuring Monitoring
- Configuring Alerting
- Custom Dashboards
- Schema Manager
- Connection Manager
- InformixHQ Server Settings
- User Settings
- Users and Permissions

Adding Servers and Groups

After logging in to InformixHQ for the very first time, you will be taken to a group dashboard for the root group. This dashboard will initially contain zero servers and zero groups.

Adding Servers

To add servers, click the **Add Server** button to add connection information for your Informix database server instances.

When adding a server to InformixHQ, you can provide two sets of credentials:

- Monitoring credentials
- Admin credentials

The **monitoring credentials** are used by the InformixHQ server whenever a user navigates to any part of the UI that issues a REST query which needs to gather data from the live Informix database server instance. The **monitoring credentials** are also used by the agent whenever it gathers data from your database server. The **admin credentials** are used whenever a user in the UI requests that an administration action be performed on the database server, for example create a dbspace, edit an onconfig parameter, or deploy an agent.

Required privileges for the monitoring user	• CONNECT access to the sysmaster database • CONNECT access to the sysadmin database
Optional privileges for the monitoring user	• select privileges on the sysconblock, syssqltrace, syssqltrace_info, syssqltrace_hvar, and syssqltrace_iter tables in the sysmaster database. • select privileges on the ph_task, ph_run, ph_alert, ph_threshold, ph_bg_jobs, and ph_bg_jobs_results tables in the sysadmin database.

	Note: Providing these optional privileges to the monitoring user enables the UI to show information about the server's the storage pool, sql tracing, auto update statistics, and scheduler tasks.
Required privileges for the admin user	• Part of the DBSA group. • CONNECT access to the sysadmin database. • Privilege to run SQL Admin API commands. • select/insert/update/delete privileges on ph_task, ph_run, ph_alert, ph_threshold, ph_bg_jobs, and ph_bg_jobs_results tables in the sysadmin database. • Execute privilege on the following functions: ◦ exectask(lvarchar) ◦ exectask(lvarchar,lvarchar) ◦ exectask(integer) ◦ exectask(integer,lvarchar) If this server is used as a repository server storing monitoring data for one or more instances, the admin user must also have: • RESOURCE access to the repository database.

It is required that you provide monitoring credentials when adding your Informix database server information to InformixHQ. Admin credentials need only be provided if you want to use InformixHQ to run administrative actions on your database server or if you want the InformixHQ server to automatically deploy your agents.

SSL Connections

If your database supports or requires SSL connections you can setup SSL using the connection properties on the **Add Server** page. Specify the following connection parameters:

SSLCONNECTION	`true`
SSL_TRUSTSTORE	Absolute or relative path to the truststore/keystore file from the perspective of the directory from which the agent and the InformixHQ server start. The truststore/keystore file must be present where both InformixHQ server and InformixHQ agent are running
SSL_TRUSTORE_PASSWORD	Password to unlock the truststore/keystore file for both InformixHQ server and InformixHQ agent

Advanced Connection Properties

You can specify any of the advanced JDBC connection parameters when you setup your connection to alter the behavior of the underlying connections InformixHQ makes to the Informix database servers. For more information, see Informix environment variables with the Informix JDBC Driver

Adding Groups

From the dashboard, you can also click the **Add Group** button to create groups of servers to make it easier to monitor and manage multiple servers.

Exploring Groups

On a group dashboard, you will see all of the servers and sub-groups that belong to the group. For each sub-group, the UI will show a card indicating the number of servers in the group (the first number is the number of servers directly in the group; the number in parenthesis is the total number of servers under it in the hierarchy). It will also show if and how many unread incidents exist for servers within the group.

There will also be a card for each server in the current group indicating the server and agent status. Since the agent monitors server status information, the server status will be unknown if the agent is offline.

From the group dashboard, you can use the **Add Server** and **Add Group** buttons to add servers or groups to the hierarchy. Each server or group also has additional actions to rename, edit connection information (for servers only), move, or delete that object. To drill down on any server or group, click on the server or group card.

From the group dashboard, you can also access the monitoring and alerting profiles for the group by selecting those menu items from the left-hand sidebar. If you are a System Administrator, you will also see a link for granting/revoking permissions on the group.

Exploring Informix Database Servers

Clicking on any server card from a group dashboard will take you to the server's dashboard. The dashboard includes sections on server status and any alerting incidents that have occurred, as well as information on high availability, threads, storage performance metrics, host memory and CPU usage, etc. If the agent is not running or

sensors are not enabled, you will see the latest measurement for these metrics queried directly from the live database server. If the corresponding sensors are running, the server dashboard will show the recent history of each of these metrics graphically to give you a visual indication of how the database server is performing.

Note: If you see a triangular exclamation icon in the top right of any dashboard, that is an indication that the dashboard uses sensors that do not exist in the server's monitoring profile. Clicking on that icon will open a pop-up detailing the sensors used by the dashboard and providing a one-click button to enable all of those sensors. For a particular database server, use the left-hand sidebar menu to continue to explore the server.

Configuring Monitoring

After clicking on a server or group from the dashboard, you can use the Monitoring link on the left-hand sidebar menu to configure the monitoring profile for that server or group. The Monitoring page lists the sensors currently configured in the server or group's monitoring profile.

Clicking the **Add Sensors** button will open a pop-up displaying the list of sensors not yet configured for this server or group. You can add custom sensors to this list from the Sensor Management page.

Click the **Edit** (pencil) icon to modify the run interval or data retention interval.

For servers or groups that inherit sensors from parent groups, there will also be a list of Inherited Sensors. The buttons next to the inherited sensors allow you to disable or override properties of the inherited sensors.

Configuring Alerting

Alerts in InformixHQ can be fully customized. You can configure not only what you want to be alerted on, but also the threshold or condition that should trigger that alert.

Alerts defined for a group are automatically inherited by the child groups and child servers of that group. Defining alerts at the group level simplifies the process of managing alerts for multiple servers.

To create alerts:

1. Click on a server or a group from the dashboard and then select the **Alerting** link on the left-hand sidebar menu. The **Alerting** page lists the alerts configured in the server or group's alerting profile.
2. Click the **Add Alert** to open a form that guides you through the process of defining an alert. Provide an alert name to identify it.
3. Select what you want to alert on: Informix server status, agent status, or data from a sensor metric.
4. Define the alerting condition. For example, alert me when the Informix server status is offline or alert me when the number of sessions connected to Informix is greater than 200.

For servers or groups that inherit alerts from parent groups, there will also be a list of Inherited Alerts. The buttons next to the inherited alerts allow you to disable inherited alerts. While you cannot override parent alerts like you can do for sensors (due to the complexities of alerts), there is a clone button provided to make it easy to clone and modify an inherited alert if changes are required.

Once alerts are defined, the InformixHQ server will evaluate the applicable alerting condition for each new data point collected by the InformixHQ agent in the process of monitoring your database server. If an alerting condition is met, an **alerting incident** is created. Alerting incidents are automatically displayed in the InformixHQ UI, both in the dashboard and the **Incidents** pages for the server and for the parent group(s) it belongs to.

You can enable alert notifications to be sent through email, PagerDuty, Twilio, or a custom alerting script. To enable alerting notifications, a system administrative user for InformixHQ must enable the desired alerting notification service on the System Settings->Alerting Configuration page. For certain alerting notification services, including email and Twilio, each individual user who wants to receive alerting notifications must enable it for their user on the My Settings->Alerting Configuration page where they will provide user specific settings like their email address or phone number.

Custom Dashboards

Creating a custom dashboard

You can create custom dashboard of sensor metric or custom query for one or more Informix database servers.

Dashboards can be created for a single server or for multiple servers. Multi-server dashboards can have up to 5 different servers. All dashboards will be saved in the current group. This allows you to open and view the dashboard for different database servers within that group, although you have the option of saving a default server or set of servers for any particular dashboard.

To create a custom dashboard:

1. Click on **Dashboards** from the sidebar menu for a server or a group.
2. Click the **New Dashboard** button.
3. Select a server or set of servers which will allow you to preview the dashboard as you build it.
4. Add one or more dashboard panels.
   Each dashboard panel can be defined to graph one or more sensor metrics from the repository. You can also define a graph based on custom query.

You can graph multiple metrics from multiple different sensors on the same graph. For multi-server dashboards, you can choose to associate each panel to a single server or you can graph metrics for each of your servers in the same graph.

You can customize the graph types (bar, pie, line and table), attributes of the left and right y-axes, colors and labels of graphed metrics.

5. Once you add graph/data source in the panel, it mandates to save the panel to apply the changes. You can also change the name of the panel by clicking on the pencil icon.
6. Arrange your dashboard panels. Panels can be re-sized and moved to different positions to create a custom layout for your dashboard.
   You can view and save your dashboard based on the timestamp selection also. Once you select your desired timestamp, the save button will be enabled/disable based on the action you performed with the timestamp selection.

   As you make changes to your dashboard throughout the process of defining and editing, it is automatically saved.

Viewing a custom dashboard

To view a custom dashboard:

1. Go to the **Dashboards** page of the group or server within the group.
2. Select the desired dashboard from the list of available dashboards.
3. Optionally, select or change the servers to show on the dashboard.
   If the selected dashboard has a set of default servers defined, it will open with the defined servers. Once opened however, you can change the servers shown in the dashboard by clicking on the server drop-down at the top of the dashboard.

   If the dashboard has no default servers defined, your context within InformixHQ will determine which servers get loaded into the dashboard when it opens. If you open a dashboard from the context of a server, then that dashboard will be automatically loaded for the current server. If you open a dashboard from the context of a group, then you will be prompted to select one or more servers from that group to show on a dashboard.

Importing a custom dashboard

To import a custom dashboard:

1. Go to the **Dashboards** page of the group or server within the group.
2. Click the **Import** button and select the JSON file that has been exported from Dashboards page in InformixHQ. New dashboard will be created with the same configuration that has been provided while exporting dashboards.

Exporting a custom dashboard

To export a custom dashboard:

1. Go to the **Dashboards** page of the group or server within the group.
2. Select the desired dashboard from the list of available dashboards.
3. Click the **Export** button. JSON file will be downloaded to system's Downloads folder.

Schema Manager

The **Schema Manager** page allows you to browse and view detailed information about the various tables and indexes in each of your databases.

Use the **Schema Manager** to:

- View detailed information about Databases
- View detailed information about Tables
- Create a Database
- Create a Demo Database
- Drop a Database
- Create a Table
- Drop a Table
- Create an Index
- Delete an Index

- Viewing Database Information
- Viewing Table Information
- Creating a Database
- Creating a Demo Database
- Dropping a Database
- Creating a Table
- Dropping a Table
- Creating an Index
- Deleting an Index

Viewing Database Information

You can view detailed information about database using different tabs like Stored Procedures, Sequences, User Defined Types, Data Blades etc.

To view database information:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select any table from the list.
3. Click on each tab to view the information like Stored Procedures, Sequences, User Defined Types, Data Blades.

Viewing Table Information

You can view detailed information about tables using different tabs like Indexes, References, Constraints, Triggers etc

To view information about table:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select any table from the list.
3. Click on each tab to view the information like Indexes, References, Constraints, Triggers.

Creating a Database

To create a database:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select any database from the list.
3. Click **...** and – Select **Create Database**
4. Enter the required details.
5. Click the **Finish** button. The new database will be created.

Creating a Demo Database

To create a demo database:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select any database from the list.
3. Click **...** and – Select **Create Demo Database**
4. Enter the required details.
5. Click the **Create** button. The demo database will be created.

Dropping a Database

To drop a database:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select the database you want to drop.
3. Click **...** and – Select **Drop Database**
4. Click **Yes** to confirm. The database will be dropped.

Creating a Table

This topic explains the steps to create a table using InformixHQ UI.

There are five types of tables:

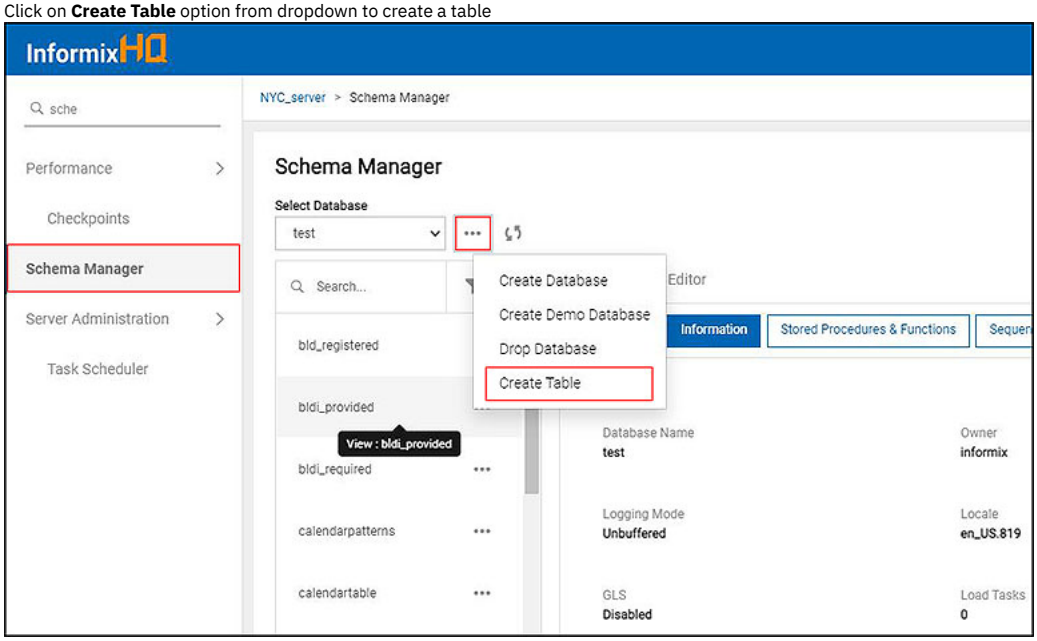
1. Standard table
2. Raw table
3. External Fixed table
4. External Delimited table

5. External Informix table

Standard & Raw table types are almost similar & three external table types are almost similar. This topic explains how to create both table types.

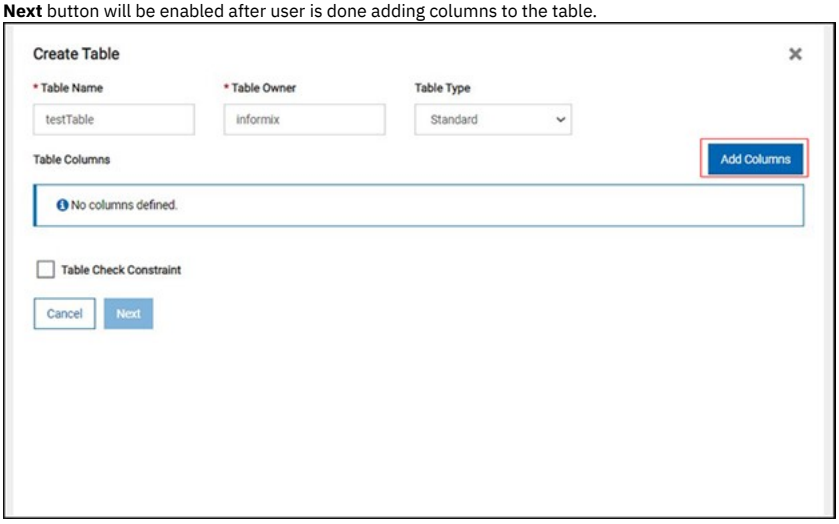
To create a table, follow the steps given below:

1. Go to the **Schema Manager** in InformixHQ.
2. Select desired database and click on menu option (3 dots) next to **Select Database** dropdown.
3. Click on **Create Table** option from dropdown to create a table

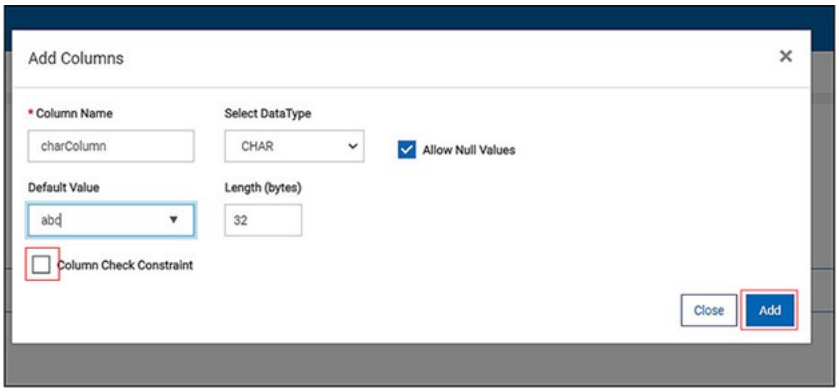


To create a Standard or Raw type table:

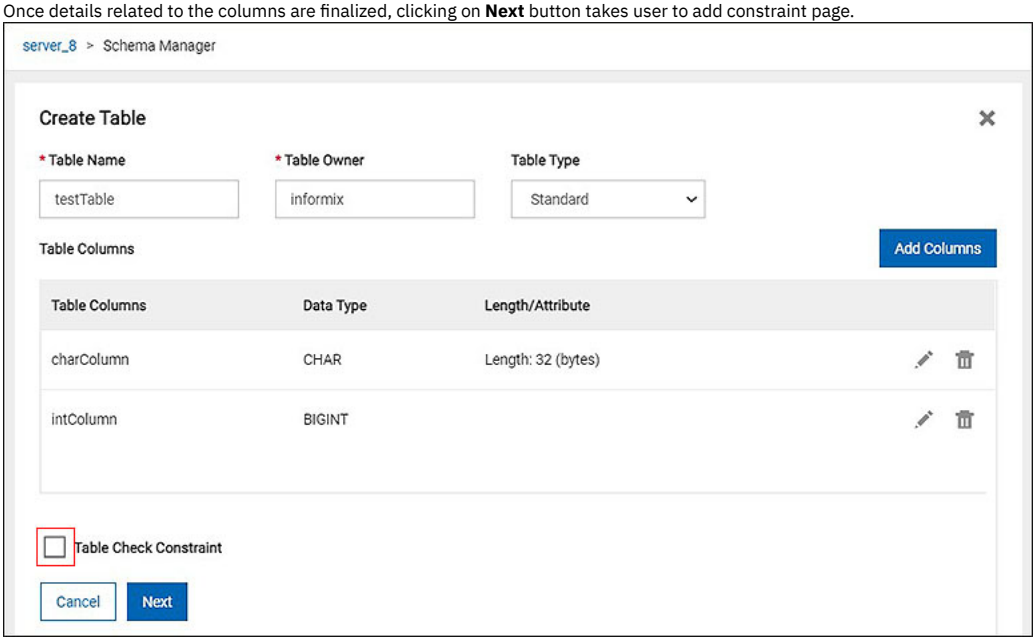
a. Enter mandatory fields such as **Table Name** and **Table Owner**.
b. Select Standard(default) or Raw type from **Table Type** dropdown.
c. Click on **Add Column** for adding columns to the table.
d. To cancel **Create Table** operation click on **Cancel** button.
e. **Next** button will be enabled after user is done adding columns to the table.



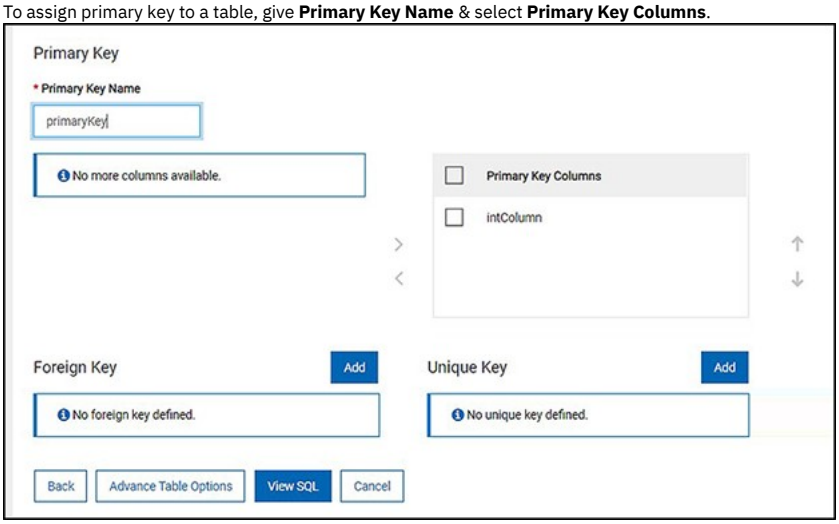
a. By clicking on **Add Columns** button pop up will appear to add column details.
b. Enter values for all mandatory fields(*).
c. To give a constraint on any column, click on **Column Check Constraint** checkbox.
d. Multiple columns can be added using the same pop up.
e. To go back to the main screen, click on **Close** button or **cross** icon in the upper right corner.



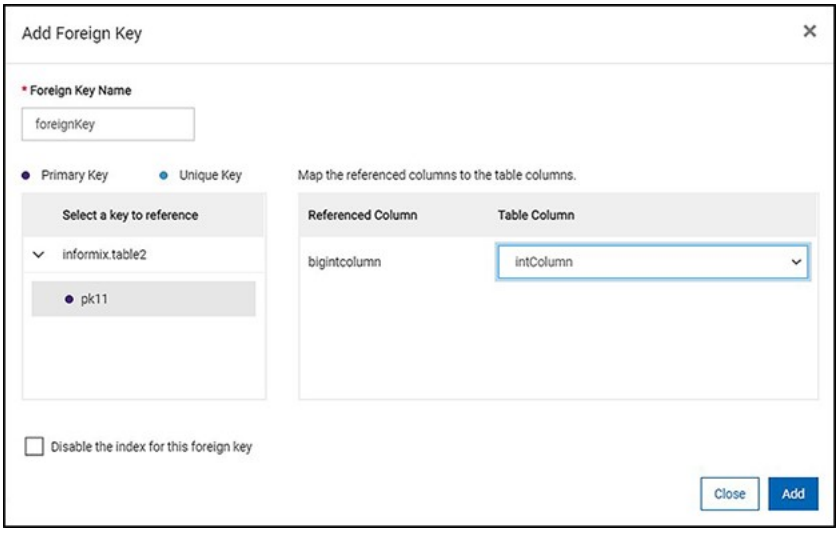
a. Once columns are added, user can view, edit, delete any of the columns.
b. Table level constraint can be added on this screen by clicking on **Table Check Constraint** checkbox.
c. Once details related to the columns are finalized, clicking on **Next** button takes user to add constraint page.



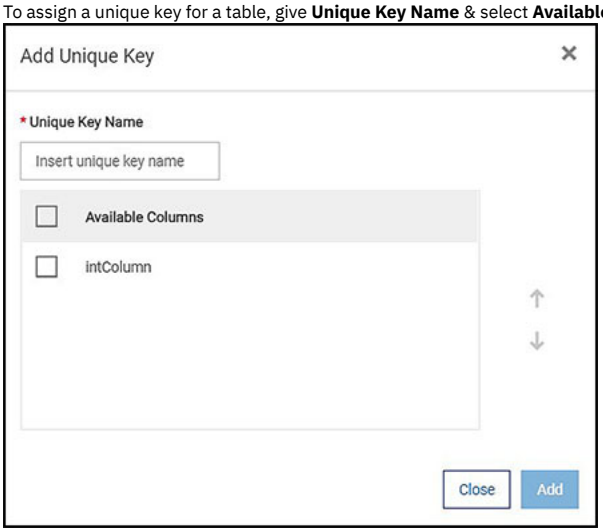
a. This screen is for adding a constraint like primary key, foreign key, unique key to a table.
b. To assign primary key to a table, give **Primary Key Name** & select **Primary Key Columns**.



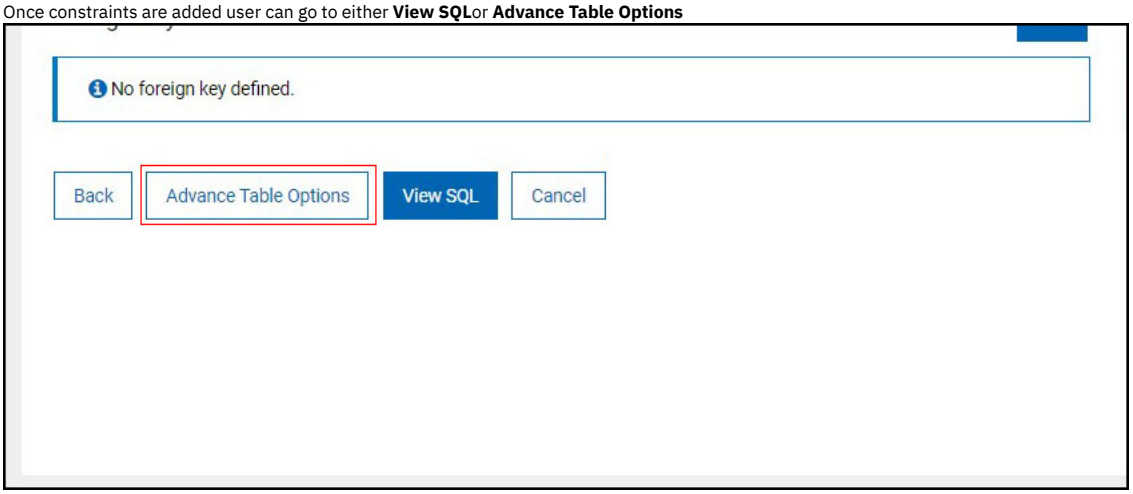
a. To add Foreign key constraint, click on **Add** button in Foreign key section.
b. To assign a foreign key for a table, give **Foreign Key Name** & map **Referenced Column** with **Table Column**.



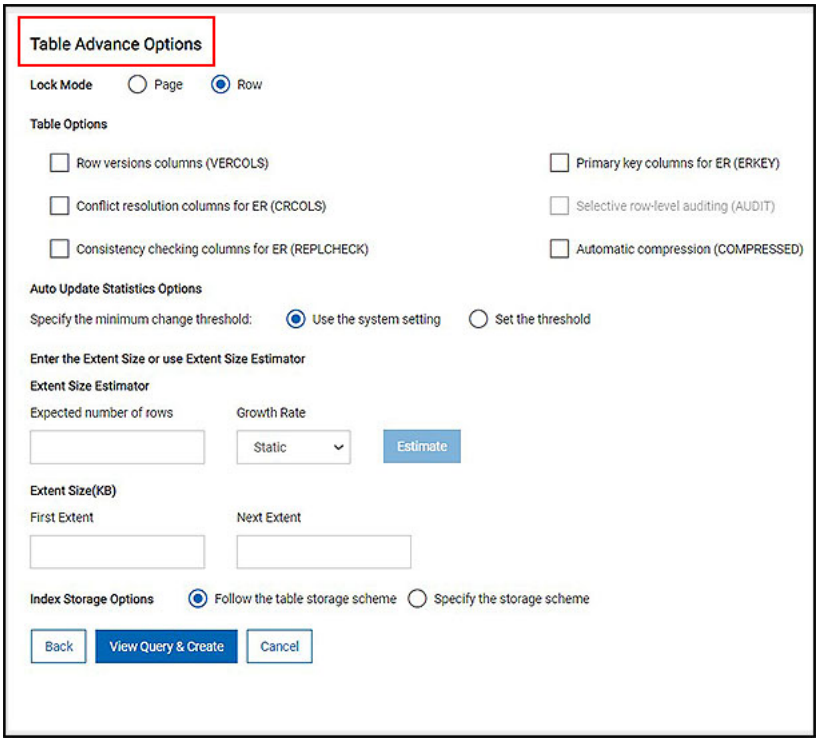
a. To add a Unique key constraint, click on **Add** button in **Unique key** section.
b. To assign a unique key for a table, give **Unique Key Name** & select **Available Columns** from the list. This will enable the **Add** button.



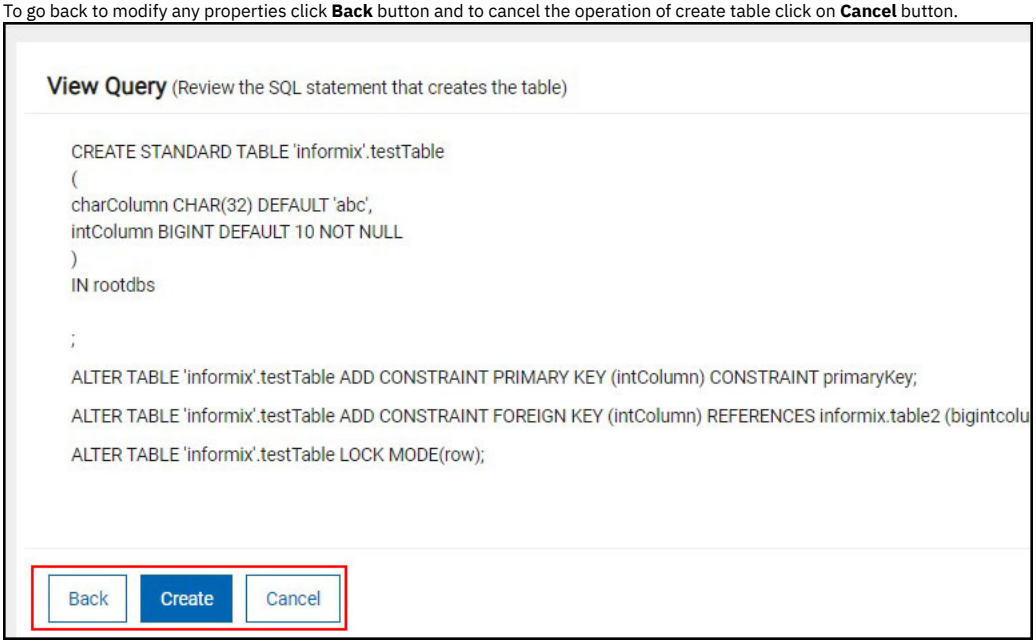
a. Once constraints are added user can go to either **View SQL**or **Advance Table Options**



a. Modify advance table level options using the screen given below. For example, changing lock mode, storage scheme, update statistics, etc.



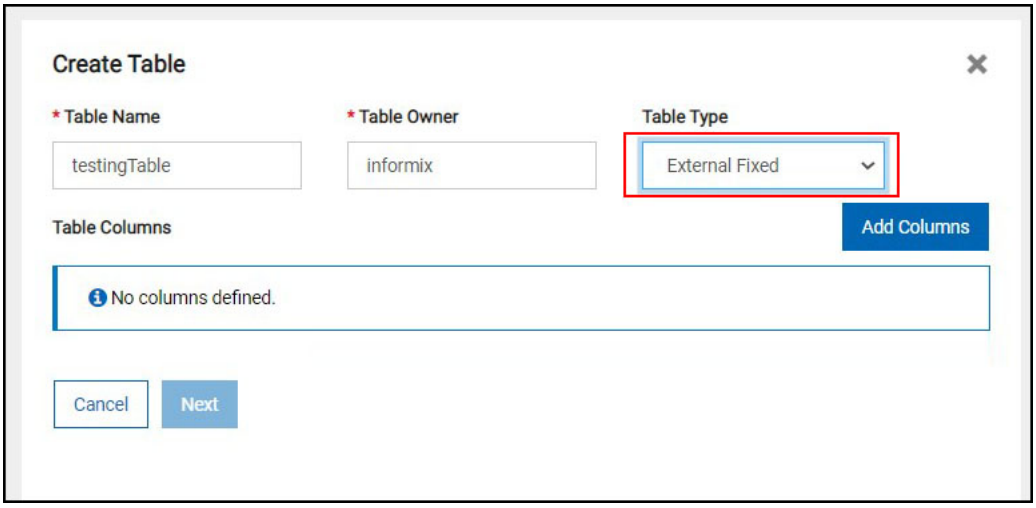
a. Once advance table option are set, click on **View Query & Create** button to view SQL query for creating the table.
b. After clicking on **View Query & Create** button from Advance Table Options or on **View SQL**button from **Add Constraints** screen, user will be able to view 'create table' query as shown in the screen given below.
c. Click **Create** button to create the table.
d. If table is created successfully, information status message will be shown and user will be taken back to **Schema Manager** page.
e. If table creation fails, error status message is displayed and all the create table related queries will be rolled back
f. To go back to modify any properties click **Back** button and to cancel the operation of create table click on **Cancel** button.



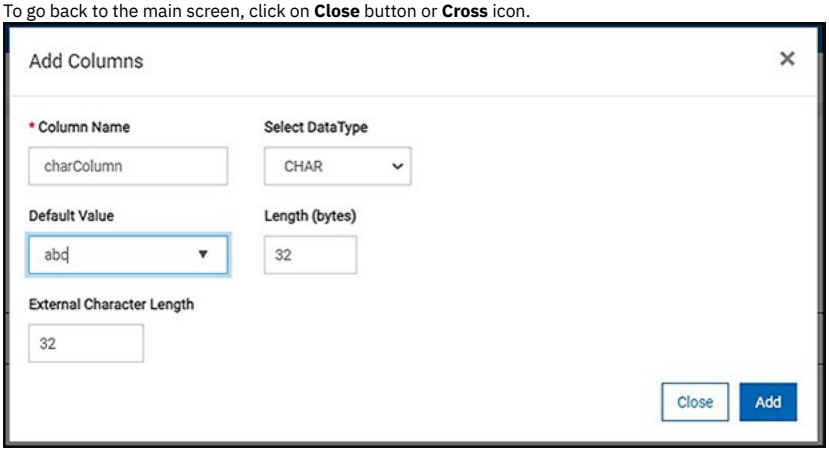
External table type:

To create an External type table:

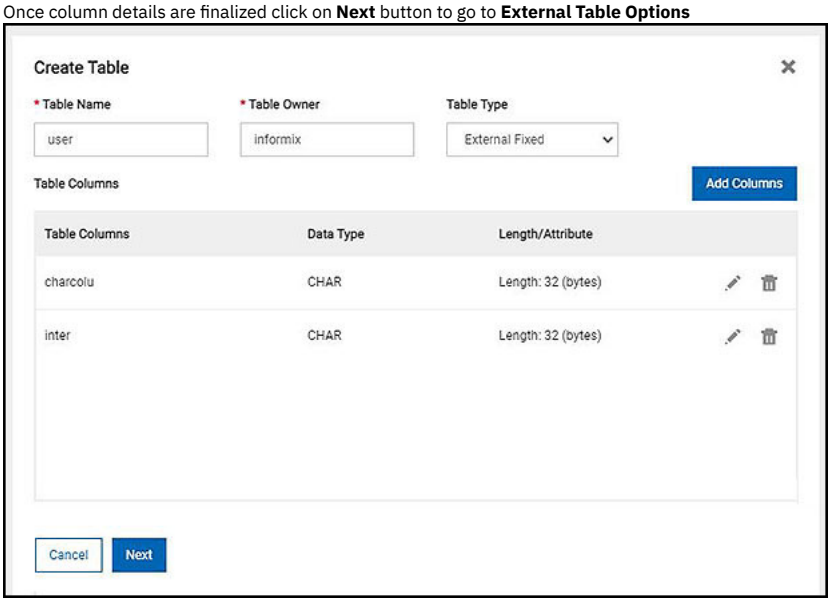
a. Enter mandatory fields such as **Table Name** and **Table Owner**.
b. Select one of **External Fixed**, **External delimited**, **External Informix** table type from **Table Type** dropdown.
c. Click on **Add Columns** for adding columns to the table.
d. To cancel **Create Table** operation, click on **Cancel** button.
e. **Next** button will be enabled once user adds columns to the table.



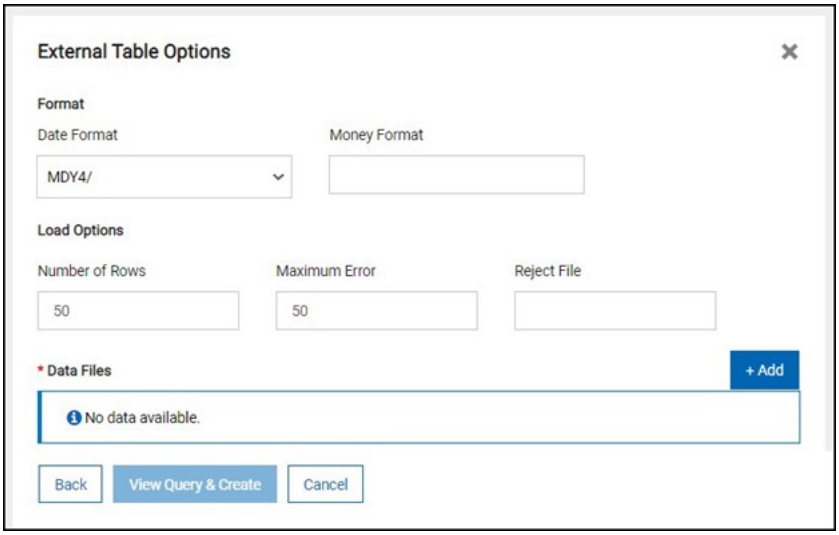
a. By clicking on **Add Column** button, a pop up will appear to add column details.
b. Enter all the mandatory field values(*).
c. User can add multiple columns using the same pop up.
d. To go back to the main screen, click on **Close** button or **Cross** icon.



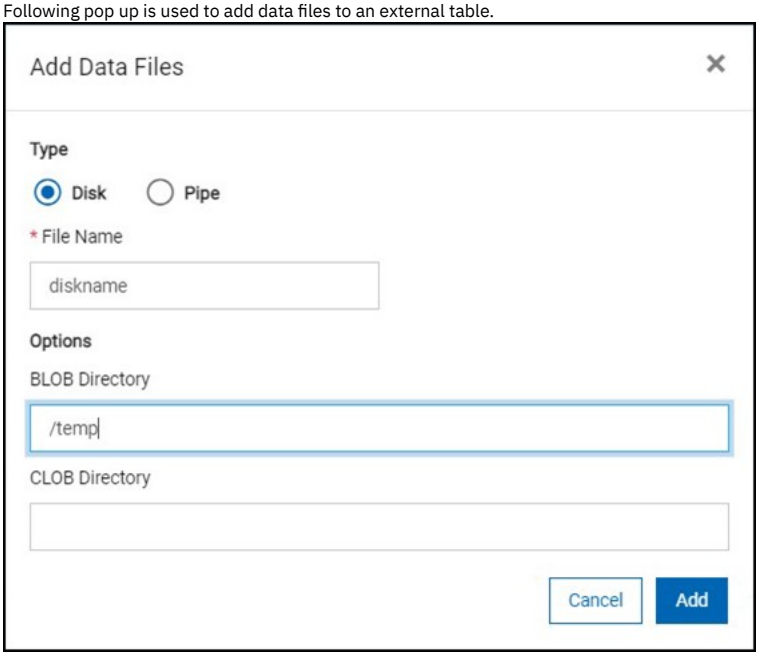
a. Once columns are added, user can view, edit , delete any of the columns.
b. Once column details are finalized click on **Next** button to go to **External Table Options**



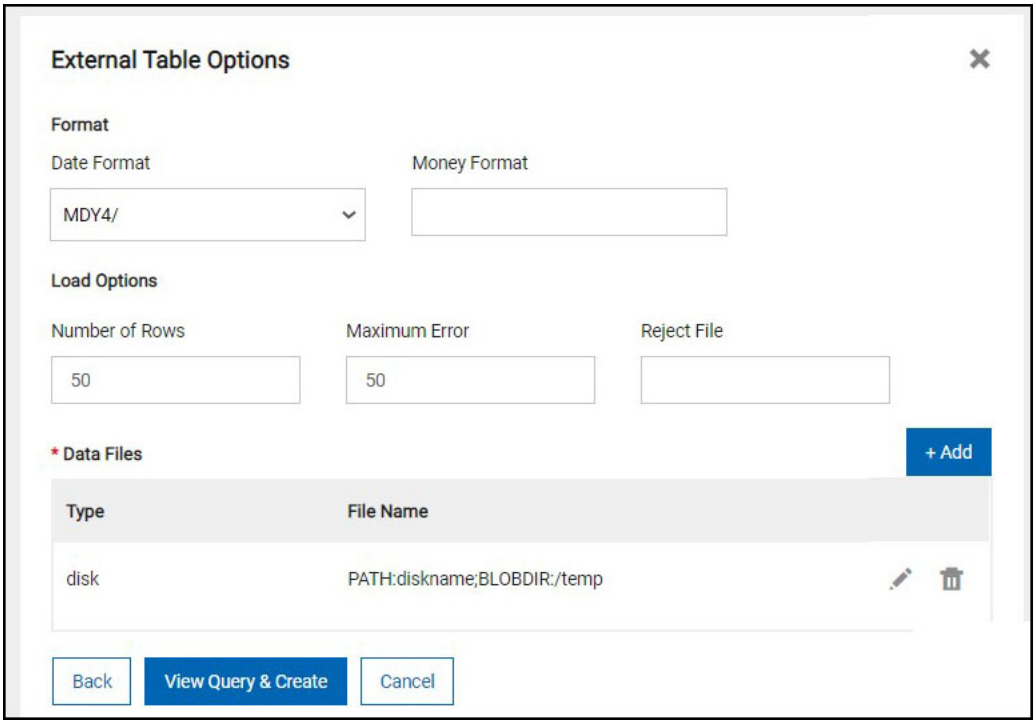
a. Provide information for external table options & add mandatory data file by clicking on **+Add** button.



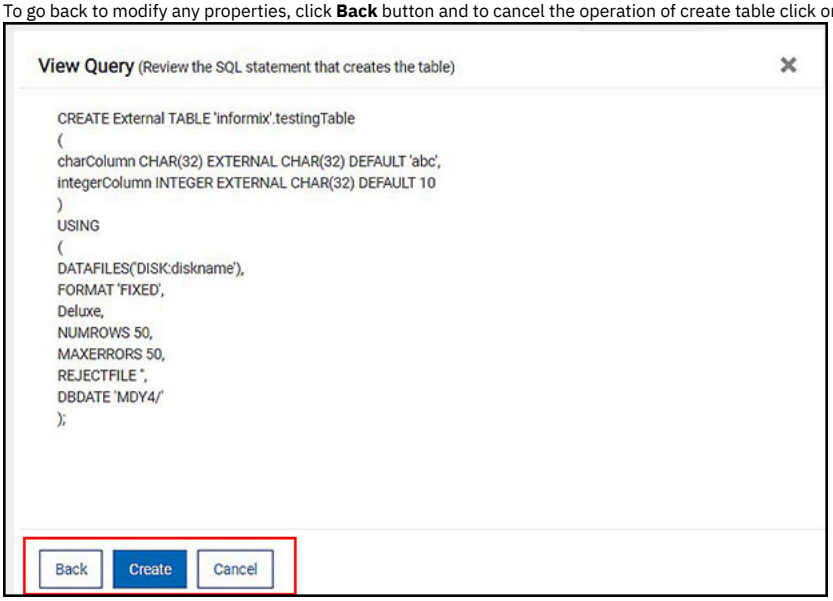
a. Following pop up is used to add data files to an external table.



a. Once data files are added, user can view, edit , delete any of the data files.
b. Once External table options are finalized click on **View Query & Create** button to view SQL query for creating the table.
c. User can either go **Back** or **Cancel** the operation using respective buttons.

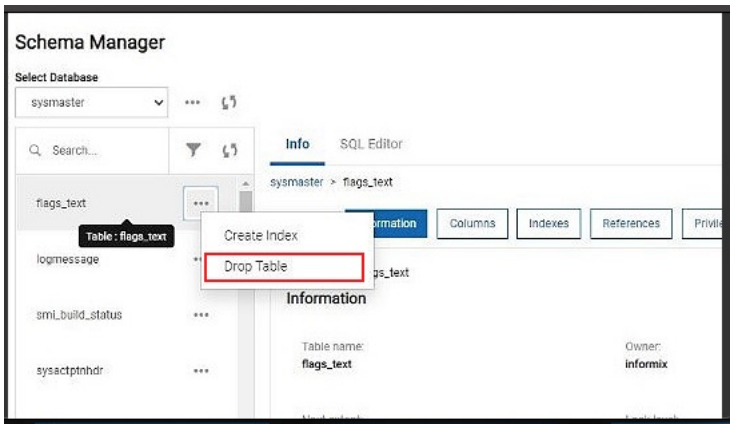


a. After clicking on **View Query & Create** button from the external table option, user will be able to view create table query as shown in the screen below.
b. Click **Create** button to create the table.
c. If table is created successfully information status message will be shown and user will be taken back to **Schema Manager** page.
d. If table creation fails, error status message will be displayed and all the create table queries will be rolled back.
e. To go back to modify any properties, click **Back** button and to cancel the operation of create table click on **Cancel** button.



Dropping a Table

1. Click on **Schema Manager** in InformixHQ.
2. Select a desired database from **Select Database** dropdown which contains the table to be dropped.
3. From the table list shown, locate the table to be dropped.
4. Click on menu option (3 dots) next to this table and select **Drop Table** from the dropdown menu to drop a table.
5. Confirm action on pop over (confirmation pop up for dropping a table).



Creating an Index

To create an index:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select the table.
3. Click **...** and – Select **Create Index**
4. Enter the required details.
5. Click the **View SQL** button to review the SQL statement.
6. Click the **Create** button, index will be created.

Note: You can also **enable** or **disable** the index.

Deleting an Index

To delete an index:

1. Go to the **Schema Manager** page in InformixHQ.
2. Select the table for which you want to delete the index.
3. Click the **Indexes** tab.
4. Select the index you want to delete.
5. Click the **Delete** icon.
6. Click **Yes** to confirm. Index will be deleted.

Connection Manager

. The **Connection Manager** page allows you to visualize and manage CM unit, SLA and FOC for any CM.

Use the **Connection Manager** to:

View all connection units

1. Go to the **Connection Manager** page in InformixHQ.
2. Click on any CM row. All CM units will be displayed.
3. Click on any CM unit name. CM unit detail page will be displayed.

View/add/modify/delete SLA within connection units

1. Go to the **Connection Manager** page in InformixHQ.
2. Click on any CM row. All CM units will be listed.
3. Click on any CM unit name. SLA table will be displayed.
4. Click the **Modify/Delete** button on any row to edit or delete the existing SLA.
5. Click the **Add SLA** button to create new SLA.

View/add/modify FOC within connection units

1. Go to the **Connection Manager** page in InformixHQ.
2. Click on any CM row. FOC details will be displayed in the Cluster CM unit detail page.
3. Click the **Modify** button to Modify FOC values. Use the Slider to enable or disable FOC.
4. To change the FOC order, click the **Click here** button in CM unit page.

View or get alerts for number of SLA connections

1. Go to the **Monitoring** page in InformixHQ.
2. Click the **+Add Sensor** button.
3. Search for SLA connections. The existing SLA Connections sensor details will be displayed in the Add sensor page.

InformixHQ Server Settings

Users with System Administrator privileges, including the initial admin user created when InformixHQ is started for the very first time, will have a **System Settings** link in the drop-down menu shown when they click on the user icon in the top right corner of the application title bar. This link is used for making changes to the global InformixHQ Configuration.

The InformixHQ Configuration page includes settings for:

- Alerting Configuration – enable and configure InformixHQ to be able to send alerts to users via email, Twilio, or PagerDuty
- Data Cleanup Configuration – configure the schedule and settings for when InformixHQ runs its repository data cleanup jobs
- Sensor Management – create and manage custom SQL sensors for monitoring your Informix database servers
- Server Configuration – configure system-wide settings for the InformixHQ server, including the REST SQL session timeout.
- User Management – add users and edit their InformixHQ privileges

- Configuring Alerting Notification
- Creating Custom Sensors

Configuring Alerting Notification

System administrative users for InformixHQ must enable which alerting notification services the InformixHQ server should use when an alerting incident occurs. If desired, you can configure and enable multiple alerting notification services.

InformixHQ supports the following alerting notification services, all of which can be configured and enabled on the **System Settings > Alerting Configuration** page.:

- **Email**
  InformixHQ can be configured to send emails through an external SMTP server when alerting incidents occur. To enable email notifications, the system administrative user must provide a SMTP server and port to use. Optionally, you can provide a user and password for authenticating to that SMTP server and a ***from email address*** that InformixHQ should use when sending alerting notification emails.

  Email notifications must first be enabled at the system level by the system administrative user. Then each individual InformixHQ user who wants email notifications must enable it for their email address on their My Settings->Alerting Configuration page.

- **Twilio**
  InformixHQ can be configured to send alerting incidents through Twilio. To enable Twilio notifications, the system administrative user must provide the Twilio account SID, authorization token, and phone number to send alerts from.

  Twilio notifications must be enabled at the system level by the system administrative user. Then each individual InformixHQ user who wants Twilio notifications must enable it for their phone number on their My Settings->Alerting Configuration page.

- **Pager Duty**
  InformixHQ can be configured to send alerting incidents through Pager Duty. Pager Duty alerts are enabled globally by the system administrative user of InformixHQ. To enable Pager Duty notifications, the system administrative user must provide a PagerDuty service key.

  Pager Duty alerts do not need to be enabled by each individual user of InformixHQ. When Pager Duty alerting notifications are enabled by the system administrative, all alerting incidents that occur in InformixHQ will be sent to the specified Pager Duty service key.

  InformixHQ sends PagerDuty notifications through REST using Pager Duty's Events API v1.

- **Run Script**
  The InformixHQ server can be configured to run a local script whenever an alerting incident occurs. Script notifications in InformixHQ provide an extensible way to integrate InformixHQ's alerting with any alerting mechanism used by your organization.

  Script notification is enabled globally by the system administrator on the **System Settings > Alerting Configuration** page. When script notification is enabled, the InformixHQ server will run the specified script whenever an alerting incident occurs on any server or group managed by InformixHQ.

  Before the InformixHQ server runs your script, it will set the following environment variables to contain information about the alerting incident that occurred:
  - ALERT_ID – id of the alerting incident
  - ALERT_TIMESTAMP – timestamp of the alerting incident
  - ALERT_SUMMARY – summary text describing the alerting incident
  - ALERT_MESSAGE – detailed message describing the alerting incident
  - SERVER_ID – id of the Informix server on which the alerting incident occurred
  - SERVER_ALIAS – alias of the Informix server on which the alerting incident occurred

- GROUP_ID – id of the parent group containing the Informix server on which the alerting incident occurred
- GROUP_NAME – name of the parent group containing the Informix server on which the alerting incident occurred
- EVENT_URL – an url link to view the alerting incident in InformixHQ

A sample use case for the “Run Script” alerting service would be, suppose your organization uses a third party alerting service that InformixHQ does not have native support for. That service requires you to POST a JSON document to a specific URL to generate an alert. You can write a script that reads in the environment variables set by InformixHQ, reformat that data into a JSON document as required, and then use curl to send a REST POST request to your organization’s alerting service.

Creating Custom Sensors

Use the **System Settings > Sensor Management** page to create and manage custom SQL sensors. Custom sensors allow you to define data to be collected by the InformixHQ agent. Data from custom sensors can be graphed on a custom dashboard. You can also use custom sensors to define alerting conditions in InformixHQ.

Each custom SQL sensor is based on a single SQL query to be run by the agent on the sysmaster database of the Informix server being monitored. Any data that can be returned by a SQL query against sysmaster can be monitored by the agent.

Only System Administrative users of InformixHQ can define custom SQL sensors. Once defined, custom sensors can be added to any server or group's monitoring profile.

To define a custom SQL sensor, go to the **System Settings > Sensor Management** page and click **Create Sensor**.

For each custom sensor, define the following:

- **SQL**
  - Define the **SQL query** to be run against the sysmaster database on the Informix server being monitored.
    - Select a sample server to run the query against to preview the data.
  - Optionally, specify the **Transpose** option to have the agent transpose (or pivot) the data returned by your SQL query based on a selected column.
  - Optionally, specify a **Primary Key** column for your query.
    - If your query returns multiple rows describing multiple objects (e.g. dbspaces), use the primary key column to define the unique identifier for each object.
- **Metrics**
  - Define a metric for each column returned by your query. Each metric will have:
    - **Name**: A display name for the metric which will be used to display this metric's data in the InformixHQ UI.
    - **Unit**: Optional. Defining the unit as percentage or bytes will direct the InformixHQ UI to format the sensor data values before they are displayed in the UI.
    - **Default Value**: Optional. A default value to be used for the metric if the query returns null for that column.
    - **Calculate Delta**: Optional. If enabled, the agent will store the difference per second between the latest reading and the previous reading.
- **Sensor Metadata**
  - A unique **id** for the sensor
  - A display **name** for the sensor
  - An optional **description** of the sensor
  - A default **run interval**
  - A default **data retention interval**

User Settings

Each user will have a Settings link in the drop-down menu shown when they click on the user icon in the top right corner of the application title bar. This link is where users can configure their own personal settings.

The user settings page includes links for:

- Alerting Configuration – enable alerts for your username and choose how to get alerts (email or Twilio). This requires that the InformixHQ administrator has configured the corresponding alerting services.
- Changing your password

- Configuring User Alerting Notification

Configuring User Alerting Notification

InformixHQ provides two alerting notification options that can be enabled for each individual user: email and Twilio. To enable these notification services, the system administrative user for InformixHQ must enable email and/or Twilio notifications at the system level on the InformixHQ System Settings > Alerting Configuration page. Then each individual user must enable email or Twilio notifications for their user on the **My Settings->Alerting Configuration** page, providing their email address or phone number respectively.

InfromixHQ also supports Pager Duty and Run Script based alerting notifications, but these are configured globally in InformixHQ and are not enabled for each individual user.

Users and Permissions

Only users with System Administrator privileges can add or delete users or modify their privileges. The initial **admin** user that is created when the InformixHQ server starts for the first time has System Administrator privileges. Additional users with System Administrator privileges can be created.

To add a user, delete a user, or modify their InformixHQ system privileges and permissions, a System Administrator user should click on their user icon in the top right corner of the title bar and go to the **System Settings > User Management** page.

When adding a user, you can optionally make the new user a System Administrator if you want them to manage users and configure InformixHQ. System Administrators automatically have full access (**Read**, **SQL**, and **Admin** permissions) on all servers and groups. When creating a new non-System Administrator, you have the option to grant **Read, SQL,** and/or **Admin** permissions for that user to any servers and groups that have been added to InformixHQ.

Read permissions allow a user to view information about a server in the UI, **Admin** permissions allow a user to execute administrative actions to make changes to a server, and **SQL** permissions allow a user to run SQL queries against that database server on the **Schema Manager** page.
Note: These three permissions are mutually exclusive. Granting **Admin** permissions does not automatically include **Read** permission.
In order to run administrative actions on an Informix database server, a user must have **Admin** permissions on that server and admin credentials must have been provided when adding that server to InformixHQ. You can review and edit server credentials by going to the server's **Setup** page or by going to the group dashboard, selecting the server's card, and clicking **Edit**.

Permissions for a server or a group in InformixHQ are inherited. If a user is granted **Read** or **Admin** privilege on a group, the same privilege will be granted on all servers and sub-groups within that group.

While the **System Settings > User Management** page allows you to view and edit the complete permissions for each individual user of InformixHQ, there is also a **Permissions** page specific to each server or group that allows to you to see (and edit) all users who have permissions to that particular server or group. You can access this page by navigating to a particular server or group and clicking on **Permissions** in the side-bar menu.

InformixHQ Agent

The InformixHQ agent is a lightweight Java 8 based monitoring agent. It should be installed on the host machine for each Informix database server that you want monitored by InformixHQ.

The agent runs alongside the database server, gathering database statistics through JDBC connections to sysmaster. The InformixHQ agent only needs read access to the database server.

By running the agent directly on the Informix server host machine, the agent is also able to gather and monitor OS statistics which can be just as critical in evaluating and tuning the Informix database server performance metrics.

- InformixHQ Agent Setup
  This topic explains how to configure InformixHQ agent from InformixHQ UI.
- InformixHQ Agent Configuration Parameters

InformixHQ Agent Setup

This topic explains how to configure InformixHQ agent from InformixHQ UI.

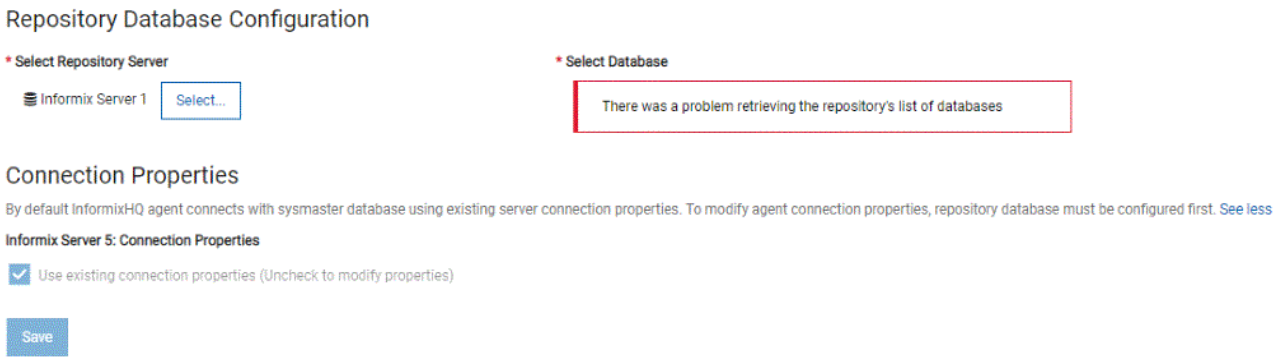
Select repository database

Repository server is the server which contains Repository database, which will be used to store all the monitoring data collected by InformixHQ agent. Without selecting a repository database, user is not allowed to save agent setup changes. Repository database server can be any Informix server specified in InformixHQ User Interface.

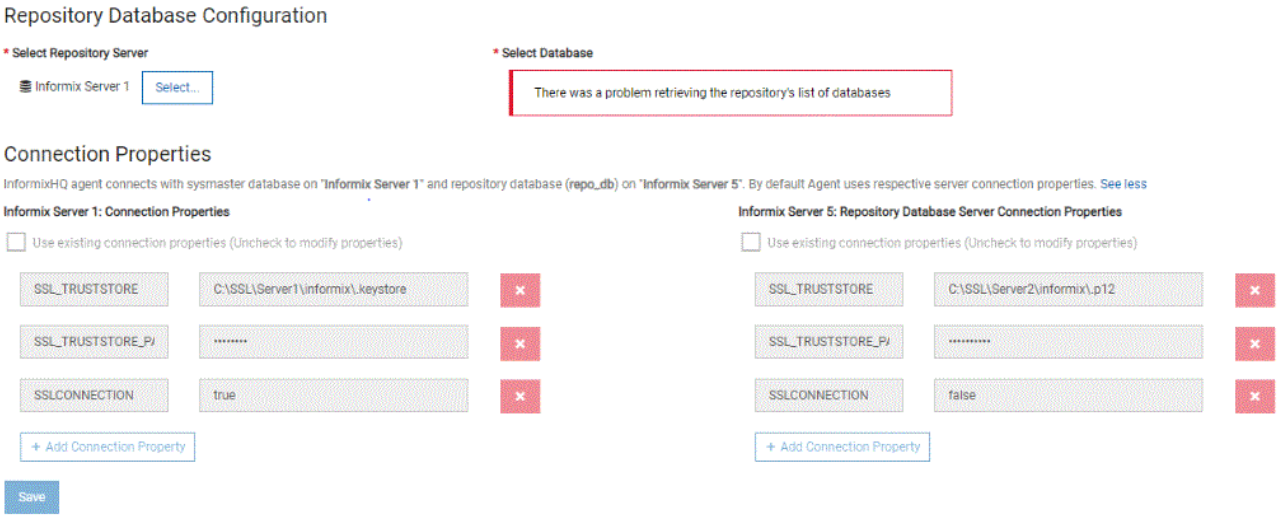
TIP: User should create a dedicated database to store metrics to be captured by InformixHQ agent. A new database can easily be created from Schema Manager page. Similarly metrics can be defined by adding Sensors from Motitoring page.

Following are two scenarios if repository database is not available for any reason:

1. ***InformixHQ Agent is not configured yet:***
   In this scenario, InformixHQ agent configuration is not set yet by the user, probably user is setting up InformixHQ agent for the first time. InformixHQ agent connects with monitoring server using existing Informix server connection properties.



2. ***InformixHQ Agent is already configured:***
   In this scenario, previously saved InformixHQ agent configuration will be shown in read only mode. Basically, whenever user runs agent jar, it will connect using already saved configuration.

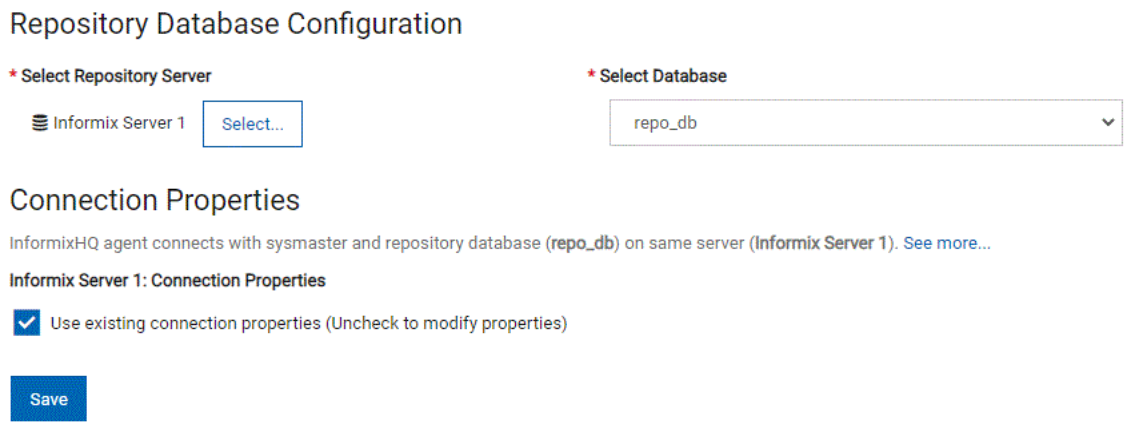


Add Connection Properties

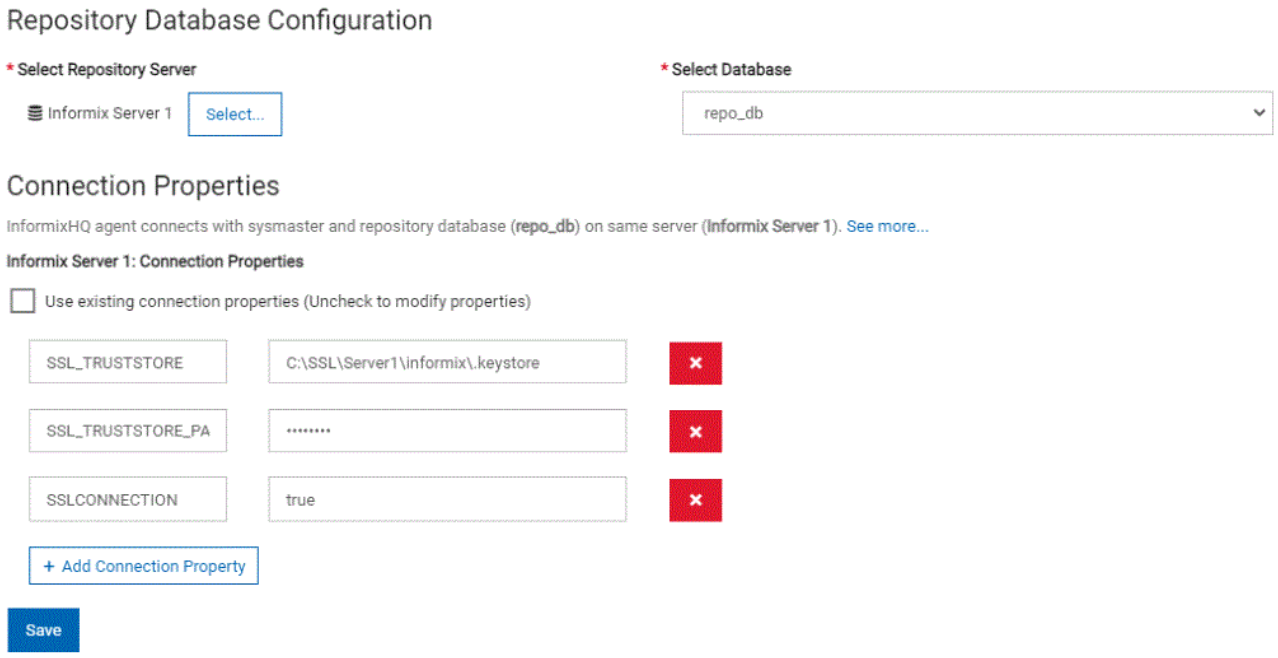
Separate connection properties can be specified for the InformixHQ agent. Users can specify connection properties for monitoring server and repository database server, respectively. Following are two cases while specifying connection properties.

1. ***Repository database is located on monitoring Server:***
   In this scenario, InformixHQ agent will connect with monitoring server and repository database using the same connection properties.

   By default, InformixHQ agent will use Informix server connection properties (from Informix server setup page) to connect with monitoring server and repository database. (When checkbox is checked)



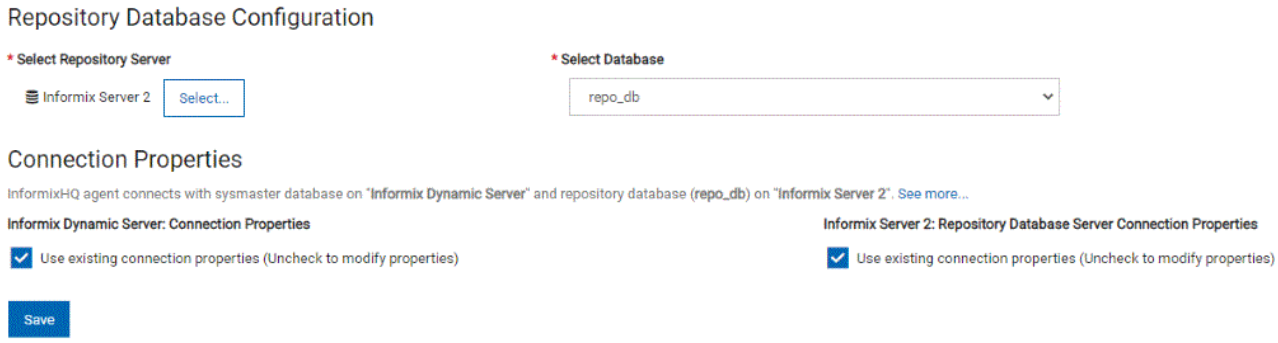
If needed user can add/modify InformixHQ agent connection properties by unchecking the checkbox as follows:



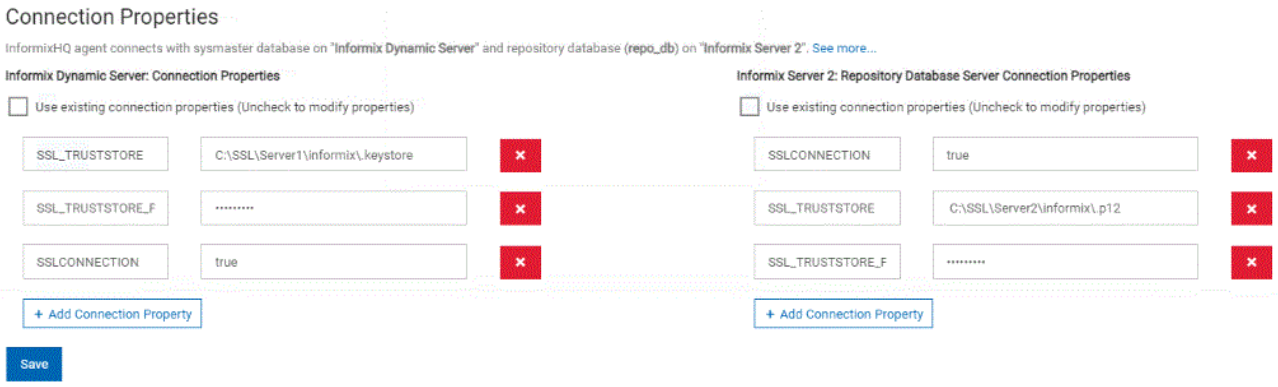
This will be specifically useful for providing different SSL keystore path for InformixHQ agent.

2. ***Repository database server is located on a different server:***
   In this scenario, InformixHQ agent will connect with monitoring server and repository database using different connection properties.

   By default, InformixHQ agent will use connection properties of respective Informix servers to connect with monitoring server and repository database, respectively. (When checkbox is checked)

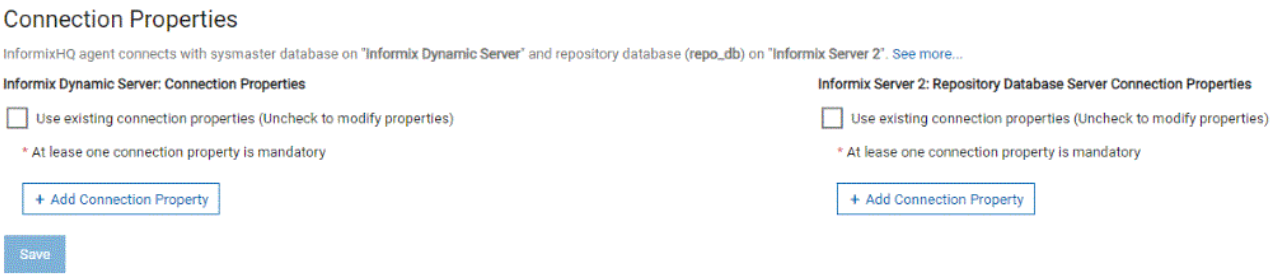


If needed user can add/modify InformixHQ agent connection properties for monitoring server and repository database separately by unchecking respective checkboxes as follows:



This will be specifically useful for providing different SSL keystore path for monitoring server and repository database, respectively. See (Compatibility matrix for Java with SSL Keystore format).

In both scenarios, if checkbox is unchecked, at least one connection property should be added to enable save button.



Note:
If Repository database is not configured or could not be connected due to any reason, then user will not be able to add/modify agent connection properties.

If already configured Repository server is removed from InformixHQ, then such server will not be visible in agent setup page under "Select Repository server". In this scenario, all the custom agent connection properties previously saved, will reset and user needs to re-enter all the custom connection properties in Agent Setup page once new repository server is selected.

Compatibility matrix for Java with SSL Keystore format

Following are keystore formats supported based on Java providers:

Java Provider	Type
IBM Java(v1.8)	JKS
Oracle Java(v1.8)	JKS, PKCS

Related information:
Schema Manager

InformixHQ Agent Configuration Parameters

A properties file is required to run the InformixHQ agent.

When starting the agent, you can pass the properties file name as part of the start command. Otherwise, the agent will look for a properties file named **agent.properties** in the classpath.

An example configuration file documenting the supported InformixHQ agent configuration properties can be found at $INFORMIXDIR/hq/informixhq-agent-example.properties.

- Required configuration properties
  - informixServer.id
  - server.host
  - server.port
- Optional configuration properties
  - dataSource.IFX_ISOLATION_LEVEL
  - pool.connectionTimeout
  - pool.idleTimeout
  - pool.maximumPoolSize
  - pool.minimumIdle
  - ssl.enable
  - ssl.keystore.file
  - ssl.keystore.password
  - target.informixdir
  - target.onlinelog
  - target.ping.frequency
  - user.password.minLength
  - user.password.requireLowerCase
  - user.password.requireUpperCase
  - user.password.requireNumber
  - user.password.requireSpecialCharacterFromSet

informixServer.id

The id of the Informix database server in InformixHQ. You find the server's id on the server's Setup page in the InformixHQ UI.

server.host

The host name on which the InformixHQ server is running.

server.port

The port on which the InformixHQ server is running.

dataSource.IFX_ISOLATION_LEVEL

Specifies the isolation level to set on JDBC connections to the target and repository Informix database servers.

The default value is 1 (DIRTY READ).

pool.connectionTimeout

Specifies the number of milliseconds to wait for a JDBC connection to the target or repository Informix database server to be established before it times out. The default value is 5000 (5 seconds).

pool.idleTimeout

Specifies the number of milliseconds that a JDBC connection can be idle in the connection pool before it is closed.

The default value is 60000 (1 minute).

pool.maximumPoolSize

The maximum number of JDBC connections in each connection pool. The InformixHQ agent will maintain a connection pool for the target database server and another a connection pool for the repository database server. The **pool.maximumPoolSize** puts a cap on the total number of open JDBC connections that can be established to each database. The default value is 3.

pool.minimumIdle

The minimum number of idle JDBC connections in each connection pool. The InformixHQ agent will maintain a connection pool for the target database server and another a connection pool for the repository database server. Setting **pool.minimumIdle** to zero indicates that all JDBC connections in the connection pool should be closed when they exceed the **pool.idleTimeout**. Setting **pool.minimumIdle** to a positive integer indicates the number of connections that should be kept open in the connection pool even when they exceed the **pool.idleTimeout**.

The default and recommended value is 0.

ssl.enable

Whether SSL should be enabled to secure web socket communication between the agent and the InformixHQ server. Set this value to true if the InformixHQ server port specified in **server.port** is an HTTPS port.
Note: if **redirectHTTPtoHTTPS** is set to true in the InformixHQ server configuration file, you must set this value to true in the agent configuration file.
If **ssl.enable** is set to true, you must also configure the **ssl.keystore.file** and **ssl.keystore.password** configuration properties.

The default value is false.

ssl.keystore.file

The path to the keystore file that contains the certificate to use for encrypting web socket communication between the agent and the InformixHQ server. This property must be set if **ssl.enable** is set to true.

ssl.keystore.password

The password to unlock the keystore file for used for encrypting web socket communication between the agent and the InformixHQ server.

This property must be set if **ssl.enable** is set to true.

target.informixdir

Optionally, specify the directory on the local machine where Informix is installed. If left empty, the agent will query the server for the INFORMIXDIR property. This property is used by sensors that gather data from onstat or other Informix utilities

target.onlinelog

Optionally specify the path to the **online.log** file for the target Informix database server.

If this is left empty and the Online Log monitoring sensor is enabled for this server, the agent will lookup the **online.log** file path by querying the database server.

There is no default value.

target.ping.frequency

Specifies the interval, in seconds, between pings to the target database server to see if it is still online. When the agent is running, it will regularly monitor whether the target database server is online or offline. This property controls the duration between these checks.

The default value is 1, indicating to check the server status every second.

user.password.minLength

Controls the minimum length for a user password. The default value is 8.

user.password.requireLowerCase

Controls whether user passwords are required to include at least one lowercase character. The default value is true.

user.password.requireUpperCase

Controls whether user passwords are required to include at least one uppercase character. The default value is true.

user.password.requireNumber

Controls whether user passwords are required to include at least one number. The default value is true.

user.password.requireSpecialCharacterFromSet

Controls whether user passwords are required to include at least one special character. An empty string indicates that no special characters are required. Setting this value to "!@#$%^&*()" would require user passwords to include at least one of those characters. The default value is an empty string.

Frequently asked questions (FAQs) about InformixHQ

These topics provide short answers to some frequently asked questions about InformixHQ.

- High level architecture and functionality
  This topic provides answers to some frequently asked questions about high level architecture and functionality.
- Getting Started
  This topic provides answers to some frequently asked questions on getting started with InformixHQ.
- Monitoring and the Repository Database
  This topic provides answers to some frequently asked questions on monitoring and the repository database in InformixHQ.
- Security
  This topic provides answers to some frequently asked questions on InformixHQ security.
- Users and Permissions
  This topic provides answers to some frequently asked questions on InformixHQ users and permissions.

High level architecture and functionality

This topic provides answers to some frequently asked questions about high level architecture and functionality.

What is the difference between the InformixHQ server and agent?

The InformixHQ server is a Java 8 based Jetty web server that hosts both the web user interface (UI) portion of the tool and the REST web services. The InformixHQ server also connects directly to the Informix database server instances to gather live data and run administration commands, manages connections to all agents, and evaluates and dispatches alerts when new monitored data comes in, among other things. You only need to run a single instance of the InformixHQ server to manage and monitor all of your Informix database server instances.

The InformixHQ agent is a lightweight Java program that runs alongside each of your Informix database server instances and gathers monitoring data. The agent is intended to be installed on the same host machine as the Informix database server that you want monitored by InformixHQ, which allows it to also gather operating system statistics about the host machine. Unlike the InformixHQ server, you will have one instance of the InformixHQ agent running for each Informix database server that you want the tool to monitor.

For information, see InformixHQ Architecture.

Is it necessary to start the agent to use InformixHQ?

The agent is not required to use InformixHQ. However, it is important to note that the agent process is the one responsible for gathering all of the monitoring data. Therefore, if you choose not to connect the agent, the tool will not be monitoring your Informix database servers when you close your web browser. You will not be able to see graphs in the UI of how various performance metrics are trending over time and you will not be able to configure alerting conditions if you are not using the agent.

I have multiple Informix database instances running on the same host machine. Do I need one agent per host or one agent per database server?

There is a one-to-one relationship between the InformixHQ agent and the Informix database server. Each agent monitors just one Informix database server instance. If you have multiple Informix database server instances on the same host and you want to monitor each of them, then you will need to have multiple agents on that same host machine, one for each database server instance.

Getting Started

This topic provides answers to some frequently asked questions on getting started with InformixHQ.

Where can I get InformixHQ?

InformixHQ is available as part of the Informix database server installation for versions 12.10.xC13 or higher.

For information, see Getting Started.

What should I do before upgrading to the latest version of InformixHQ?

Before upgrading to the latest version, it is suggested to make a backup copy of the old database file i.e. a backup copy of h2db.mv.db file. This is required in case if you ever want to revert to the previous version of InformixHQ with previous InformixHQ jar.

Where can I find sample configuration files for the server and agent?

Sample configuration files for both the InformixHQ server and agent are available in the **$INFORMIXDIR/hq** directory of your Informix database server installation.

For information, see InformixHQ Server Configuration and InformixHQ Agent Configuration.

How can I configure the logging for the server or agent? How do I change the logging level?

InformixHQ uses the logback library for logging. By default, the InformixHQ server and agent will log messages at INFO level to an informixhq-server.log file and an informixhq-agent.log file respectively.

You can customize the logging behavior by providing a server-logback.xml file in the current directory or classpath when starting the InformixHQ server or a agent-logback.xml file in the current directory or classpath when starting the InformixHQ agent. Use these logback configuration files to change the logging level (ERROR, WARN, INFO, or DEBUG), change the log file location, or enable rolling window logging. For more information, see logback and Logging in InformixHQ.

How do you stop the server and how do you stop the agent?

You can stop both the InformixHQ server and agent by terminating the java process that is running them.

Monitoring and the Repository Database

This topic provides answers to some frequently asked questions on monitoring and the repository database in InformixHQ.

Where is the monitored data stored?

The monitored data is stored in the repository database as defined in the InformixHQ UI. The repository database must be an Informix database, but it can be located wherever you choose. You can store the monitored data within the same Informix instance that is being monitored or you can choose to use a central repository database to store all of the monitored data for all of your Informix database server instances.

Does the repository database need to exist ahead of time? Do I need to run any DDL statements to initialize the repository database?

The repository database must exist before you define it as a repository database. You define a repository database in the InformixHQ UI on a server's **Setup > Agent** page.

You do not need to run any DDL statements to initialize the schema for the repository database. The agent will automatically create the tables it needs to store the monitored data.

If I add a new database server instance to a group and start an agent for that new server, do I have to do anything to enable all of the group's sensors on the new server?

No, this happens automatically. Sensors defined in a group's monitoring profile are automatically applied to all database servers within the group. When a new server is added to the group, it will automatically inherit all of the sensors enabled in the group's monitoring profile. No additional step is needed to make this happen. The same thing applies to a group's alerting profile.

Security

This topic provides answers to some frequently asked questions on InformixHQ security.

Do I need to keep the initialAdminPassword in the properties file after the InformixHQ server is started for the first time? Isn't it a security issue to keep the password in plain text in the properties file?

The initialAdminPassword property is only required in the InformixHQ server properties file the first time it is started. When the server is started for the very first time, it initializes its internal H2 database and creates the initial admin user. For all subsequent starts of the InformixHQ server, the admin user will already exist and therefore the initialAdminPassword will be ignored if it is present in the properties file. This means that after the server is started for the first time, you can safely remove the initialAdminPassword property from the properties file. This allows you to avoid having that password continue to sit around in plain text in your properties file.

How can I configure HTTPS and/or SSL for InformixHQ?

To use the Secure Sockets Layer (SSL) protocol to encrypt communication with InformixHQ, you will need a keystore and certificate. You can use the method that best fits your environment for creating the keystore and certificate, for example Java keytool, OpenSSL, or even the IBM Global Security Kit (GSKit).

- **Configuring HTTPS in the InformixHQ server**
  Once you have the keystore, secure the InformixHQ web user interface and REST API by configuring HTTPS in the InformixHQ server. To configure HTTPS in the InformixHQ server, in your InformixHQ server properties file, set the httpsPort, ssl.keystore.file, and ssl.keystore.password properties and potentially also the ssl.key.password property if your key password is different from the keystore password.

  Additionally, if you want to disable HTTP access to the InformixHQ so that all communication to and from the InformixHQ server uses HTTPS, set the httpPort to -1 in your properties file. If instead you would like the InformixHQ server to automatically redirect all HTTP traffic to the HTTPS port, set the redirectHTTPtoHTTPS property to true.

  Sample InformixHQ server properties file with HTTPS enabled:

```
# The initialAdminPassword is only required the first time the InformixHQ server is started
initialAdminPassword=myAdminPassword

# configure ports
httpPort=-1
httpsPort=8088
redirectHTTPtoHTTPS=false

# configure keystore
ssl.keystore.file=/opt/informixhq/mykeystore.jks
ssl.keystore.password=myStorePassword
# uncomment the following line if a separate key password is required for your keystore
#ssl.key.password=myKeyPassword
```

- **Configuring the InformixHQ agent to encrypt its web socket communication with SSL**

  Once you have HTTPS enabled in the InformixHQ server, you must configure your InformixHQ agents to encrypt their web socket communication with the InformixHQ server. If you use the **Deploy Agent** button in the UI to have the InformixHQ server automatically deploy the agent, it will automatically configure the agent to use SSL if the InformixHQ server has HTTPS enabled.

  If you are starting your agents manually to enable SSL, set the ssl.enable property to true in your agent configuration file and then set the ssl.keystore.file property, the ssl.keystore.password property.

  Sample agent configuration file with SSL enabled:

```
# host and port of the InformixHQ server
server.host=localhost
server.port=8088

# The id of the Informix database server as defined in InformixHQ
informixServer.id=1
```

```
# SSL configuration
ssl.enable=true
ssl.keystore.file=/opt/informixhq/mykeystore.jks
ssl.keystore.password=myStorePassword
```

How can I encrpyt the internal H2 database that the InformixHQ server uses?

The InformixHQ server creates an H2 database to store its internal metadata. The H2 database file, **h2db.mv.db** will be created in the directory where you start the InformixHQ server. It will store information about the groups and servers you define in the tool (including the database server connection credentials), the monitoring and alerting profiles, and alerting incidents.

You can configure encryption for this H2 database file by setting the following properties in your InformixHQ server configuration file.

```
h2.encrypt.enable=true
h2.encyrpt.password=some_password
```

Optionally, you can also set the h2.encrypt.algorithm property if you want to set the encryption algorithm to something other than AES.

Note: If you want to encrypt the H2 database, you must set these properties the first time you start the InformixHQ server when the H2 database is first created and initialized. You cannot change your H2 encryption configuration after the H2 database has been created. If you want to encrypt an H2 database that has already been created, you can use H2's ChangeFileEncryption tool as described in http://www.h2database.com/html/features.html#file_encryption or you can delete your h2db.mv.db file and have the InformixHQ server recreate it from scratch the next time you start it.

How can I configure InformixHQ to use SSL when connecting to my database server?

If your database supports or requires SSL connections, you can setup SSL using the connection properties on the **Add Server** page when adding the server or on the server's **Setup** page after it is created.

You must add the following connection properties in order to use SSL on InformixHQ's JDBC connections to your database server:

```
SSLCONNECTION=true
SSL_TRUSTSTORE=/path/to/truststore
SSL_TRUSTORE_PASSWORD=password
```

The truststore/keystore file that you specify must be present both where InformixHQ server is running as well as the machine where the InformixHQ agent is running.

For more information, see Adding Severs and Groups.

Users and Permissions

This topic provides answers to some frequently asked questions on InformixHQ users and permissions.

What's the difference between monitoring/admin credentials and Read/SQL/Admin permissions?

The monitoring and admin credentials are used by the InformixHQ server itself as part of the JDBC connection whenever it needs to connect to the database server. The Read, SQL, and Admin permissions are those that are assigned to users in the tool and therefore control what users can see and do in the UI.

- **Monitoring and admin credentials**

  When you click **Add Server** from a group dashboard page to add a new Informix database server to InformixHQ, you are asked to provide the InformixHQ tool with not only the host and port, but also user and password information that will be used when establishing a JDBC connection to that database server instance.

  The monitoring credentials are used by the InformixHQ server whenever it needs to query for data to be displayed in the UI or by the InformixHQ agent data it monitors your database server instance.

  The admin credentials are used whenever a user in the UI requests that an administration action be performed on the database server, for example creating a dbspace, editing an onconfig parameter, or deploying an agent. The user provided for the admin credentials should be a DBSA and needs access to the sysadmin database and permissions to run SQL Admin API commands on the database server.

  The required privileges for the monitoring and admin users can also be found in Adding Severs and Groups.

  It is required that you provide monitoring credentials when adding your Informix database server information to InformixHQ. Admin credentials need only be provided if you want to use InformixHQ to run administrative actions on your database server or if you want that server to be used as a repository database.

- **Read, SQL, and Admin permissions**

  The Read, SQL, and Admin permissions are the permissions assigned to the users that are created in InformixHQ itself. These permissions determine what kinds of access that a user has on various Informix servers and groups in the UI and also which REST services that user is authorized to run. Therefore, these permissions control what each user can see or do in the tool.

  Read permission provides the ability to view information about a server. SQL permission provides the ability to run any SQL query or statement (including DML) against the database server on its Schema Manager page in the UI. Admin permission provides the ability run administrative actions on the server, for example

creating a dbspace or editing and onconfig parameter. Read, SQL, and Admin permissions are mutually exclusive, so be sure to grant each user the full set of permissions that they will need.

Read, SQL, and Admin permissions can be granted to InformixHQ users by a System Administrator user in InformixHQ on the **System Settings > User Management** page or on the **Permissions** page for any server or group.

What privileges are required on the Informix database server?

The required privileges for the monitoring and admin users can be found in Adding Severs and Groups.

Is there any relationship between the users I create in InformixHQ and OS users?

There is no relationship between the users you create in InformixHQ and the operating system users on the host machine it is running on. Users that are created in InformixHQ are users that are specific to the InformixHQ tool. They have no relationship to operating system users or even Informix database server users.

Backup and restore

The backup and restore guides contain information about backing up and restoring data and managing storage devices and media.

External resources

- Knowledge Collection: Informix® Backup and Restore Utilities (Support document)
  This document helps you find the available resources that are related to backing up and restoring data with products.
- Using Optim™ with Informix Dynamic Server, Part 1: Configure Informix Dynamic Server to work together with Optim Solutions (IBM® developerWorks®)
  This tutorial describes how to use IBM Optim Solutions to segregate older data and maintain it in such a way that it is easily accessible for reporting or strategic decision making. In part 2 of this tutorial, follow scenarios to understand how you can use Optim to mask archived data and maintain privacy.
- Data archiving with Informix Dynamic Server table-level restore (IBM developerWorks)
  This article describes the most common options for archiving your IBM Informix data. It describes techniques to retrieve your data from a backup and store the data in an independent format, which is preserved against future changes in your environment.

Main resources

- Backup and Restore Guide
  These topics describe how to use the IBM Informix ON-Bar and ontape utilities to back up and restore database server data. These utilities enable you to recover your databases after data is lost or becomes corrupted due to hardware or software failure or accident.

Backup and Restore Guide

These topics describe how to use the IBM® Informix® ON-Bar and **ontape** utilities to back up and restore database server data. These utilities enable you to recover your databases after data is lost or becomes corrupted due to hardware or software failure or accident.

These topics are of interest to the following users:

- Database administrators
- System administrators
- Backup operators
- Technical support personnel

These topics are written with the assumption that you have the following background:

- Some experience with storage managers, which are applications that manage the storage devices and media that contain backups
- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with database server administration, operating-system administration, or network administration

- Overview of backup and restore
  These topics provide an overview of backup and restore concepts. They also provide information about planning for backup and restore operations.
- ON-Bar backup and restore system
- ontape backup and restore system
- Backup and restore a Remote Secondary Server(RSS)
- Integrated Backup Encryption
  These topics provide information about Integrated Backup Encryption.
- Informix Primary Storage Manager
  The IBM Informix Primary Storage Manager manages storage for ON-Bar backup and restore operations, including parallel backups, that use file devices (disks).
- archecker table level restore utility
- Backup and restore configuration parameter reference
- Cloud Backup
  These topics provide information about storing and retrieving the backups directly to the ecosystem of selected cloud providers, namely Amazon S3 and Softlayer Object Storage.
- Appendixes

Overview of backup and restore

These topics provide an overview of backup and restore concepts. They also provide information about planning for backup and restore operations.

- Backup and restore concepts
  IBM Informix provides two utilities for backing up and restoring database server data. Both utilities back up and restore storage spaces and logical logs. However, they support different features and it is important to know the differences. These topics explain basic backup and restore concepts for IBM Informix database servers and compares the ON-Bar and **ontape** utilities.
- Plan for backup and restore
  These topics describe the planning for backup and restore, for example by planning your recovery strategy and backup system.

Backup and restore concepts

IBM® Informix® provides two utilities for backing up and restoring database server data. Both utilities back up and restore storage spaces and logical logs. However, they support different features and it is important to know the differences. These topics explain basic backup and restore concepts for IBM Informix database servers and compares the ON-Bar and **ontape** utilities.

ON-Bar backs up and restores storage spaces (dbspaces) and logical file, by using a storage manager, whereas **ontape** does not use a storage manager.

- Recovery system
  A *recovery system*, which includes backup and restore systems, enables you to back up your database server data and later restore it if your current data becomes corrupted or inaccessible.
- Comparison of the ON-Bar and ontape utilities
  This topic contains information to help you compare the ON-Bar and **ontape** utilities, so you can determine when to use each utility.

Recovery system

A *recovery system*, which includes backup and restore systems, enables you to back up your database server data and later restore it if your current data becomes corrupted or inaccessible.

The causes of data corruption or loss can range from a program error to a disk failure to a disaster that damages the entire facility. A recovery system enables you to recover data that you already lost due to such mishaps.

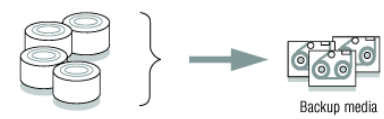
- Backup systems
  A *backup* is a copy of one or more *dbspaces* (also called *storage spaces*) and logical logs that the database server maintains. You can also back up *blobspaces* and *sbspaces*.
- Backup levels
  To provide flexibility, the ON-Bar and **ontape** utilities support three backup levels.
- Logical-log backup
  A *logical-log backup* is a copy to disk or tape of all full logical-log files. The logical-log files store a record of database server activity that occurs between backups.
- Restore systems
  A *restore* recreates database server data from backed-up storage spaces and logical-log files.

Backup systems

A *backup* is a copy of one or more *dbspaces* (also called *storage spaces*) and logical logs that the database server maintains. You can also back up *blobspaces* and *sbspaces*.

The backup copy is typically written to a *secondary storage* medium such as disk or magnetic tape. Store the media offline and keep a copy off site if possible.

Important: Database backups do not replace ordinary operating-system backups, which back up files other than IBM® Informix® database files.
The following figure illustrates the basic concept of a database backup.
Figure 1. A backup of database server data



You do not always have to back up all the storage spaces. If some tables change daily but others rarely change, it is inefficient to back up the storage spaces that contain the unchanged tables every time that you back up the database server. You need to plan your backup schedule carefully to avoid long delays for backing up or restoring data.

Related concepts:
Backup levels
Logical-log backup
Restore systems

Backup levels

To provide flexibility, the ON-Bar and **ontape** utilities support three backup levels.

Level 0
: Level 0 backs up all used pages that contain data for the specified storage spaces.
You need all these pages to restore the database to the state that it was in at the time that you made the backup.

  Level-0 backups can be time-consuming because ON-Bar writes all the disk pages to back up media. Level-1 and level-2 backups might take almost as much time as a level-0 backup because the database server must scan all the data to determine what has changed since the last backup. It takes less time to restore data from level-0, level-1, and level-2 backups than from level-0 backups and a long series of logical-log backups.

Level 1
: Level 1 backs up only data that has changed since the last level-0 backup of the specified storage spaces.
All changed table and index pages are backed up, including those pages with deleted data. The data that is copied to the backup reflects the state of the changed data at the time that the level-1 backup began.

  A level-1 backup takes less space and might take less time than a level-0 backup because only data that changed since the last level-0 backup is copied to the storage manager.

Level 2
: Level 2 backs up only data that has changed since the last level-1 backup of the specified storage spaces.
A level-2 backup contains a copy of every table and index page in a storage space that has changed since the last level-1 backup.

  A level-2 backup takes less space and might take less time than a level-1 backup because only data that changed since the last level-1 backup is copied to the storage manager.

Important: If disks and other media are destroyed and need to be replaced, you need at least a level-0 backup of all storage spaces and relevant logical logs to restore data completely on the replacement hardware.

Related concepts:
Backup systems
Logical-log backup
Restore systems

Related reference:
Schedule backups

Logical-log backup

A *logical-log backup* is a copy to disk or tape of all full logical-log files. The logical-log files store a record of database server activity that occurs between backups.

To free full logical-log files, back them up. The database server reuses the freed logical-log files for recording new transactions. For a complete description of the logical log, see your *IBM® Informix® Administrator's Guide*.

Restriction: Even if you do not specify logging for databases or tables, you need to back up the logical logs because they contain administrative information such as checkpoint records and additions and deletions of chunks. When you back up these logical-log files, you can do warm restores even when you do not use logging for any of your databases.

- Manual and continuous logical-log backups
  You can manually back up logical logs or you can enable continuous logical-log backup.
- Log salvage
  When the database server is offline, you can perform a special logical-log backup, called a *log salvage*. In a log salvage, the database server accesses the log files directly from disk. The log salvage backs up any logical logs that have not yet been backed up and are not corrupted or destroyed.
- Save logical-log backups
  You should perform frequent logical-log backups and then save the logical-log backups from at least the last two level-0 backups so that you can use them to complete a restore.

Related concepts:
Backup systems
Backup levels
Restore systems

Manual and continuous logical-log backups

You can manually back up logical logs or you can enable continuous logical-log backup.

A *manual logical-log backup* backs up all the full logical-log files and stops at the current logical-log file. You must monitor your logical logs carefully and start logical-log backups as needed.

To find out if a logical-log file is ready to be backed up, check the flags field of **onstat -l**. After the logical-log file is marked as backed up, it can be reused. When the flags field displays any of the following values, the logical-log file is ready to be backed up:

```
U------
U-----L
```

The value `U` means that the logical-log file is used. The value `L` means that the last checkpoint occurred when the indicated logical-log file was current. The value `C` indicates the current log. If `B` appears in the third column, the logical-log file is already backed up and can be reused.

```
U-B---L
```

The flag values `U---C-L` or `U---C--` represent the current logical log. While you are allowed to back up the current logical log, doing so forces a log switch that wastes logical-log space. Wait until a logical-log file fills before you back it up.

If you turn on *continuous logical-log backup*, the database server backs up each logical log automatically when it becomes full. If you turn off continuous logical-log backup, the logical-log files continue to fill. If all logical logs are filled, the database server hangs until the logs are backed up. You can start continuous logical log backups by setting the ALARMPROGRAM configuration parameter in the onconfig file or by running an ON-Bar or **ontape** command.

Related concepts:
Log salvage
Related reference:
Save logical-log backups

Log salvage

When the database server is offline, you can perform a special logical-log backup, called a *log salvage*. In a log salvage, the database server accesses the log files directly from disk. The log salvage backs up any logical logs that have not yet been backed up and are not corrupted or destroyed.

The log salvage enables you to recover all of your data up to the last available and uncorrupted logical-log file and the last complete transaction.

Related concepts:
Manual and continuous logical-log backups
Related reference:
Save logical-log backups

Save logical-log backups

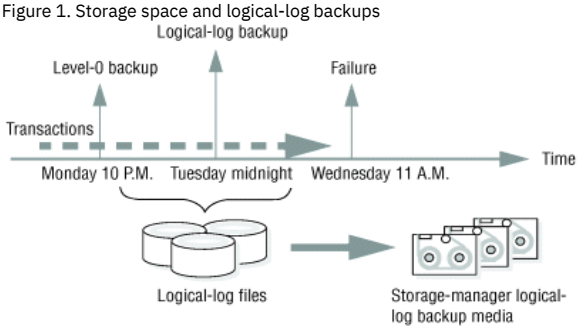
You should perform frequent logical-log backups and then save the logical-log backups from at least the last two level-0 backups so that you can use them to complete a restore.

Perform frequent logical-log backups for the following reasons:

- To free full logical-log files
- To minimize data loss if a disk that contains logical logs fails
- To ensure that restores contain consistent and the latest transactions

You should save the logical-log backups from the last two level-0 backups because if a level-0 backup is inaccessible or unusable, you can restore data from an older backup. If any of the logical-log backups are also inaccessible or unusable, however, you cannot roll forward the transactions from those logical-log files or from any subsequent logical-log files.
Important: You lose transactions in logical-log files that are not backed up or salvaged.
To illustrate, as the following figure shows, suppose you perform a level-0 backup on Monday at 10 p.m. and then back up the logical logs on Tuesday at midnight. On Wednesday at 11 a.m., you suffer a mishap that destroys your databases. You would be unable to restore the transactions that occurred between midnight on Tuesday and 11 a.m. on Wednesday unless you had continuous logical-log backup setup.

If the disks that contain the storage spaces with the logical logs are damaged, the transactions after midnight on Tuesday might be lost. To restore these transactions from the last logical-log backup, try to salvage the logical logs before you repair or replace the bad disk and then perform a cold restore.
Figure 1. Storage space and logical-log backups



Related concepts:
Manual and continuous logical-log backups
Log salvage

Restore systems

A *restore* recreates database server data from backed-up storage spaces and logical-log files.

A restore recreates database server data that has become inaccessible because of any of the following conditions:

- You need to replace a failed disk that contains database server data.
- A logic error in a program has corrupted a database.
- You need to move your database server data to a new computer.
- A user accidentally corrupted or destroyed data.

To restore data up to the time of the failure, you must have at least one level-0 backup of each of your storage spaces from before the failure and the logical-log files that contain all transactions since these backups.

- Physical and logical restores
  ON-Bar and **ontape** restore database server data in two phases. The first phase is the *physical restore*, which restores data from backups of all or selected storage spaces. The second phase is the *logical restore*, which restores transactions from the logical-log backups.
- Warm, cold, and mixed restores
  When you restore data, you must decide whether to do so while the database server is in quiescent, online, or offline mode. The type of restore depends on which of these operating modes the server is in.
- Continuous log restore
  A *continuous log restore* keeps a second system available to replace the primary system if the primary system for restoring logs fails.

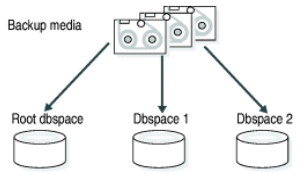
Related concepts:
Backup systems
Backup levels
Logical-log backup

Physical and logical restores

ON-Bar and **ontape** restore database server data in two phases. The first phase is the *physical restore*, which restores data from backups of all or selected storage spaces. The second phase is the *logical restore*, which restores transactions from the logical-log backups.

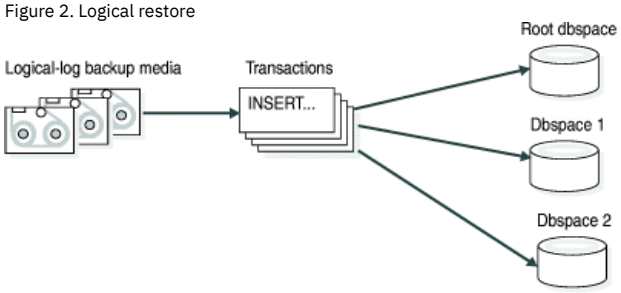
Physical restore

During a physical restore, ON-Bar or **ontape** restores the data from the most recent level-0, level-1, and level-2 backups. When you suffer a disk failure, you can restore to a new disk only those storage spaces with chunks that resided on the failed disk. The following figure illustrates a physical restore.
Figure 1. Physical restore



Logical restore

As the following figure shows, the database server *replays* the logical logs to reapply any database transactions that occurred after the last backup. The logical restore applies only to the physically restored storage spaces.
Figure 2. Logical restore



The database server automatically knows which logical logs to restore.

For more information, see Restore data with ON-Bar and Restore with ontape.

Related concepts:
Warm, cold, and mixed restores
Continuous log restore

Warm, cold, and mixed restores

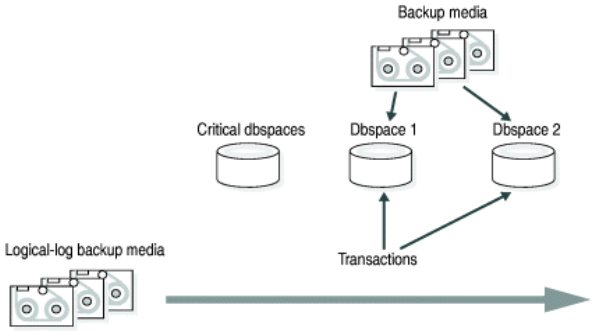
When you restore data, you must decide whether to do so while the database server is in quiescent, online, or offline mode. The type of restore depends on which of these operating modes the server is in.

The types of restores are as follows:

- If you restore noncritical dbspaces while the database server is online or quiescent, that process is called a *warm restore*.
- When IBM® Informix® is offline, you can perform only a *cold restore*.
- A *mixed restore* is a cold restore of some storage spaces followed by a warm restore of the remaining storage spaces.

Warm restore

As the following figure shows, a warm restore restores noncritical storage spaces. A warm restore consists of one or more physical restores, a logical-log backup, and a logical restore.
Figure 1. Warm restore

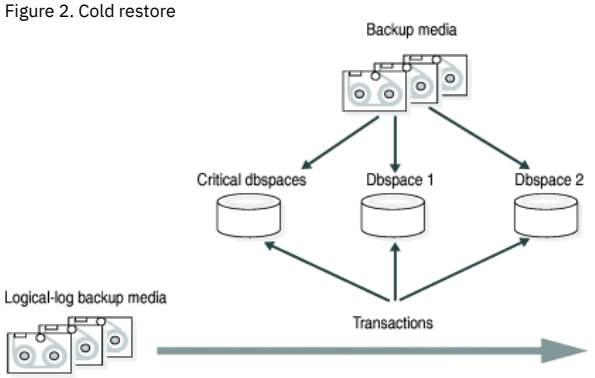


You cannot perform more than one simultaneous warm restore.

Cold restore

As the following figure shows, a cold restore salvages the logical logs, and restores the critical dbspaces (root dbspace and the dbspaces that contain the physical log and logical-log files), other storage spaces, and the logical logs.

Figure 2. Cold restore



You can perform a cold restore onto a computer that is not identical to the one on which the backup was performed by giving any chunk a new path name and offset during the restore.

When restoring a whole-system backup, it is not necessary to restore the logical logs. A whole-system backup contains a snapshot of the entire instance at the moment the backup was performed, which is logically consistent across all dbspaces.

When restoring a standard backup, you must restore the logical logs by performing a logical restore.

A cold restore starts by physically restoring all critical storage spaces, then the noncritical storage spaces, and finally the logical logs. The database server goes into recovery mode after the reserved pages of the root dbspace are restored. When the logical restore is complete, the database server goes into quiescent mode. Use the **onmode** command to bring the database server online.

Tip: If you mirror the critical dbspaces, you are less likely to have to perform a cold restore after a disk failure because the database server can use the mirrored storage space. If you mirror the logical-log spaces, you are more likely to be able to salvage logical-log data if one or more disks fail.

Required: Cold restores are required for Enterprise Replication servers before resuming replication.

Mixed restores

A mixed restore makes the critical data available sooner, however, the complete restore takes longer because the logical logs are restored and replayed several times, once for the initial cold restore and once for each subsequent warm restore.

The initial set of storage spaces you restore in the cold restore must include all critical storage spaces in the server. To the extent that you do not restore all storage spaces during the initial cold restore and avoid the time necessary to restore them, you can bring the server online faster than if you were to perform a cold restore of the entire server. You can then restore the remaining storage spaces in one or more warm restores.

The storage spaces that you do not restore during the cold restore are not available until after you restore them during a warm restore, although they might not have been damaged by the failure.

Related concepts:
Physical and logical restores
Continuous log restore
Related reference:
Determine failure severity

Continuous log restore

A *continuous log restore* keeps a second system available to replace the primary system if the primary system for restoring logs fails.

Normal log restore restores all of the available log file backups and applies the log records. After the last available log is restored and applied, the log restore finishes. Transactions that are still open are rolled back in the transaction cleanup phase, then the server is brought into quiescent mode. After the server is quiesced, no more logical logs can be restored.

With continuous log restore, instead of transaction clean up the server is put into log restore suspended state after the last available log is restored. The restore client (**ontape** or ON-Bar) exits and returns control to you. With the server in this state, you can start another logical restore after additional logical logs become available. As long as you start each log restore as a continuous log restore, you can continue this cycle indefinitely.

One use of continuous log restore is to keep a second system available in case the primary system fails. You can restore logical logs backed up on the primary system on the secondary system as they become available. If the primary system fails, you can restore remaining available logical logs on the secondary system and bring that secondary system online as the new primary system.

Continuous log restore requires much less network bandwidth than High-Availability Data Replication (HDR) and enterprise data replication (ER). Continuous log restore is more flexible than HDR and ER because you can start continuous log restore at any time. As a result, continuous log restore is more robust than HDR or ER in unpredictable circumstances, such as intermittent network availability.

For more information, see Configuring a continuous log restore by using ON-Bar and Configuring continuous log restore with ontape.

Related concepts:
Physical and logical restores
Warm, cold, and mixed restores
Related tasks:
Configuring a continuous log restore by using ON-Bar
Configuring continuous log restore with ontape

Comparison of the ON-Bar and ontape utilities

This topic contains information to help you compare the ON-Bar and **ontape** utilities, so you can determine when to use each utility.

ON-Bar
: Backs up and restores storage spaces (dbspaces) and logical files, by using a storage manager to track backups and storage media. Use this utility when you need to:

  - Select specific storage spaces
  - Back up to a specific point in time
  - Perform separate physical and logical restores
  - Back up and restore different storage spaces in parallel
  - Use multiple tape drives concurrently for backups and restores
  - Perform imported restores
  - Perform external backups and restores

ontape
: Logs, backs up, and restores data, and enables you to change the logging status of a database. It does not use a storage manager. Use this utility when you need to:

  - Back up and restore data without a storage manager
  - Back up without selecting storage spaces
  - Change the logging mode for databases

Important: The backup that **ontape** and ON-Bar produce are not compatible. You cannot create a backup with **ontape** and restore it with ON-Bar, or vice versa.
The following table compares ON-Bar and **ontape**.

Table 1. Differences between ON-Bar and ontape

Can the utility...	ON-Bar	ontape
Use a storage manager to track backups and storage media?	yes	no
Back up all database server data?	yes	yes
Back up selected storage spaces?	yes	no
Back up logical-log files?	yes	yes
Perform continuous logical-log backups?	yes	yes
Perform continuous logical-log restore?	yes	yes
Back up while the database server is online?	yes	yes
Back up while the database server is in quiescent mode?	yes	yes
Restore all database server data?	yes	yes
Restore selected storage spaces?	yes	yes
Back up and restore storage spaces serially?	yes	yes
Perform cold restores with the database server offline?	yes	yes
Initialize high availability data replication?	yes	yes
Restore data to a specific point in time?	yes	no
Perform separate physical and logical restores?	yes	yes
Back up and restore different storage spaces in parallel?	yes	no
Use multiple tape drives concurrently for backups and restores?	yes	no
Restart a restore?	yes	no
Rename a chunk path name or device during a cold restore?	yes	yes
Perform imported restores?	yes	yes

Can the utility...	ON-Bar	ontape
Perform external backups and restores?	yes	yes
Monitor performance?	yes	no
Change logging mode for databases?	no	yes
Transform data with external programs?	yes	yes
Back up to or restore from cloud storage?	no	yes
Encrypt or decrypt a storage space during a restore?	yes	yes

Additional differences:

- Emergency boot files and sysutils database
  The **ontape** utility does not use the **sysutils** database or the emergency boot files.
- Simultaneous sessions
  ON-Bar, with IBM® Informix® Primary Storage Manager, supports simultaneous sessions.
- Device support and storage management
  The **ontape** utility supports remote backup devices on other hosts.

  ON-Bar, with the Informix Primary Storage Manager, supports the export of backup generations into specified directories and devices.

  You can also use ON-Bar with the IBM Spectrum Protect or third-party storage managers to obtain device support and storage management.
- Changing the logging mode of a database
  You cannot change the logging mode for ON-Bar; however you can use the **ondblog** utility to do this task when using ON-Bar.

  You can also use the SQL administration API alternative, ALTER LOGMODE to change the logging mode.

For details about each utility, see Back up with ON-Bar and Back up with ontape.

Related reference:
Configure ontape

Plan for backup and restore

These topics describe the planning for backup and restore, for example by planning your recovery strategy and backup system.

- Plan a recovery strategy
  Before you use ON-Bar or **ontape**, plan your recovery goals.
- Plan a backup system for a production database server
  To plan for adequate backup protection for your data, analyze your database server configuration and activity and the types of backup media available at your installation.

Plan a recovery strategy

Before you use ON-Bar or **ontape**, plan your recovery goals.

- Types of data loss
  The first step in planning a recovery strategy is to determine how much data loss, if any, is acceptable.
- Determine failure severity
  After you determine your recovery goals, create your recovery plan. The plan should include recovery goals for multiple levels of failure.
- Data use determines your backup schedule
  After you develop your recovery plan, create a backup plan based on how you use your data.
- Schedule backups
  You recovery strategy should include a schedule of backups. Tailor your backup plan to the requirements of your system. The more often the data changes and the more important it is, the more frequently you need to back it up.
- Security requirements for label-based access control
  For label-based access control (LBAC), the person who runs ON-Bar or **ontape** does not require an exemption to security policies or an additional privilege to back up or restore data.

Types of data loss

The first step in planning a recovery strategy is to determine how much data loss, if any, is acceptable.

The following types of data loss can occur:

- Deletion of the following:
  - Rows, columns, tables, or databases
  - Chunks, storage spaces, or logical logs
- Data corruption or incorrect data created
- Hardware failure (such as a disk that contains chunk files fails or a backup tape that wears out)
- Database server failure

- Natural disaster

Determine failure severity

After you determine your recovery goals, create your recovery plan. The plan should include recovery goals for multiple levels of failure.

The following table shows recovery plans for failures with amounts of lost data.

Table 1. Sample recovery plans

Failure severity	Data loss	Suggested recovery plan
Small	Noncritical data is lost.	Restore of the data can wait until a nonpeak time. Use a warm restore.
Medium	The data that is lost is critical for your business but does not reside in a critical dbspace.	Perform a warm restore of this data as soon as possible.
Large	Critical dbspaces are lost.	Use a mixed restore to restore the critical data right away and a warm restore to restore noncritical data during off-peak hours.
Disaster	All data is lost.	Perform a cold or mixed restore as soon as possible.

Related concepts:
Warm, cold, and mixed restores

Data use determines your backup schedule

After you develop your recovery plan, create a backup plan based on how you use your data.

How you use the data determines how you plan your backup schedule, as follows:

- Data usage
  How do users use the data?
  - Critical dbspaces (root dbspace and dbspaces that contain the physical log and at least one logical-log file)
  - Critical business application data
  - Long-term data storage for legal or record-keeping reasons
  - Data sharing among groups
  - Test data
- Transaction Time
  How much transaction time can be lost? Also, how long might it take to re-enter lost transactions manually? For example, can you afford to re-enter all transactions that occurred over the past three hours?
- Quantity and Distribution
  How much data can you afford to lose? For example, you lost one fourth of your customer profiles, or you lost the Midwest regional sales figures but the West Coast figures are intact.

Ask the following questions to assist in deciding how often and when you want to back up the data:

- Does your business have downtime where the system can be restored?
- If your system is 24x7 (no downtime), is there a nonpeak time where a restore could occur?
- If a restore must occur during a peak period, how critical is the time?
- Which data can you restore with the database server online (warm restore)? Which data must be restored offline (cold restore)?
- How many storage devices are available to back up and restore the data?

Schedule backups

You recovery strategy should include a schedule of backups. Tailor your backup plan to the requirements of your system. The more often the data changes and the more important it is, the more frequently you need to back it up.

Your backup plan should also specify the backup level.

The following table shows a sample backup plan for a small or medium-sized system.

Table 1. Sample backup plan

Backup level	Backup schedule
Complete backup (level-0)	Saturday at 6 p.m.
Incremental backup (level-1)	Tuesday and Thursday at 6 p.m.
Incremental backup (level-2)	Daily at 6 p.m.
Level-0 backup of storage spaces that are updated frequently	Hourly

Important: Perform a level-0 backup after you change the physical schema, such as adding a chunk to a storage space. (See Preparing to back up data.)
Related concepts:
Backup levels

Security requirements for label-based access control

For label-based access control (LBAC), the person who runs ON-Bar or **ontape** does not require an exemption to security policies or an additional privilege to back up or restore data.

LBAC protection remains intact after you restore data with ON-Bar or **ontape**.

Plan a backup system for a production database server

To plan for adequate backup protection for your data, analyze your database server configuration and activity and the types of backup media available at your installation.

Also, consider your budget for storage media, disks, computers and controllers, and the size of your network.

Actions after which to perform a level-0 back up

You must perform a level-0 backup of, at minimum, the root dbspace and the modified storage spaces after you perform any of the following actions:

- Add or drop mirroring.
- Move, drop, or resize a logical-log file.
- Change the size or location of the physical log.
- Change your storage-manager configuration.
- Add, move, or drop a dbspace.
- Add, move, or drop a chunk to any type of storage space.
- Add, move, or drop a blobspace or sbspace.

For example, if you add a new dbspace **dbs1**, you see a warning in the message log that asks you to perform a level-0 backup of the root dbspace and the new dbspace. If you attempt an incremental backup of the root dbspace or the new dbspace instead, ON-Bar automatically performs a level-0 backup of the new dbspace.

Tip: Although you no longer need to back up immediately after adding a log file, your next backup should be level-0 because the data structures have changed.
If you create a storage space with the same name as a deleted storage space, perform a level-0 backup twice:

1. Back up the root dbspace after you drop the storage space and before you create the storage space with the same name.
2. After you create the storage space, back up the root dbspace and the new storage space.

Actions before which to perform a level-0 back up

You must perform a level-0 backup of the modified storage spaces before you perform any of the following actions:

- Convert a nonlogging database to a logging database.
- Before you alter a RAW table to type STANDARD. This backup ensures that the unlogged data is restorable before you switch to a logging table type.

- Evaluate hardware and memory resources
  When planning your backup system, evaluate your hardware and memory resources.
- Evaluate backup and restore time
  Several factors. including database server configuration and the size of your database, affect the amount of time that the system needs to back up and restore data.
- Evaluate logging and transaction activity
  When planning your backup system, also consider logging and transaction activity.
- Compress row data
  Compressing row data can make backing up and restoring data more efficient.
- Transform data with external programs
  You can use external programs as filter plug-ins to transform data to a different format before a backup and transform it back after the restore.

Related reference:
onbar -b syntax: Backing up

Evaluate hardware and memory resources

When planning your backup system, evaluate your hardware and memory resources.

Evaluate the following database server and hardware configuration elements to determine which storage manager and storage devices to use:

- The number of I/O virtual processors
- The amount of memory available and the distribution of processor activity

Also consider temporary disk space needed for backup and restore. The database server uses temporary disk space to store the before images of data that are overwritten while backups are occurring and overflow from query processing that occurs in memory.

When preparing to back up data, make sure that you correctly set the DBSPACETEMP environment variable or parameter to specify dbspaces with enough space for your needs. If there is not enough room in the specified dbspaces, the backup will fail, root dbspace will be used, or the backup will fail after filling the root dbspace.

Evaluate backup and restore time

Several factors. including database server configuration and the size of your database, affect the amount of time that the system needs to back up and restore data.

How long your backup or restore takes depends on the following factors:

- The speed of disks or tape devices
  The faster the storage devices, the faster the backup or restore time.
- The number of incremental backups that you want to restore if a disk or system failure requires you to rebuild the database
  Incremental backups use less storage space than full backups and also reduce restore time.
- The size and number of storage spaces in the database
  Backups: Many small storage spaces take slightly longer to back up than a few large storage spaces of the same total size.

  Restores: A restore usually takes as long to recover the largest storage space and the logical logs.
- Whether storage spaces are mirrored
  If storage spaces are mirrored, you reduce the chance of having to restore damaged or corrupted data. You can restore the mirror at nonpeak time with the database server online.
- The length of time users are interrupted during backups and restores
  If you perform backups and warm restores while the database server is online, users can continue their work but might notice a slower response. If you perform backups and warm restores with the database server in quiescent mode, users must exit the database server. If you perform a cold restore with the database server offline, the database server is unavailable to users, so the faster the restore, the better. An external backup and restore eliminates system downtime.
- The backup schedule
  Not all storage spaces need to be included in each backup or restore session. Schedule backups so that you can back up more often the storage spaces that change rapidly than those storage spaces that seldom or never change. Be sure to back up each storage space at level-0 at least once.
- The layout of the tables across the dbspaces and the layout of dbspaces across the disks
  When you design your database server schema, organize the data so that you can restore important information quickly. For example, you isolate critical and frequently used data in a small set of storage spaces on the fastest disks. You also can fragment large tables across dbspaces to balance I/O and maximize throughput across multiple disks. For more information, see your *IBM® Informix® Performance Guide*.
- The database server and system workload
  The greater the workload on the database server or system, the longer the backup or restore time.
- The values of backup and restore configuration parameters
  For example, the number and size of data buffers that ON-Bar uses to exchange data with the database server can affect performance. Use the BAR_NB_XPORT_COUNT and BAR_XFER_BUF_SIZE configuration parameters to control the number and size of data buffers.

Evaluate logging and transaction activity

When planning your backup system, also consider logging and transaction activity.

The following database server usage requirements affect your decisions about the storage manager and storage devices:

- The amount and rate of transaction activity that you expect
- The number and size of logical logs
  If you need to restore data from a database server with little transaction activity, define many small logical logs. You are less likely to lose data because of infrequent logical-log backups.
- How fast the logical-log files fill
  Back up log files before they fill so that the database server does not hang.
- Database and table logging modes
  When you use many nonlogging databases or tables, logical-log backups might become less frequent.

Compress row data

Compressing row data can make backing up and restoring data more efficient.

Compressing row data before backing it up can improve the speed of backing up and restoring and requires less backup media. A smaller size of data results in the following advantages over uncompressed data during backup and restore:

- Backing up is quicker.
- Restoring is quicker.
- The logical logs are smaller.
- The backup image is smaller.

Using an external compression utility to compress a backup image of compressed row data might not reduce the size of the backup image, because already compressed data usually cannot be further compressed. In some cases, the size of the backup image of compressed row data might be larger than the size of the backup image that was compressed by an external utility.

Transform data with external programs

You can use external programs as filter plug-ins to transform data to a different format before a backup and transform it back after the restore.

To compress or transform data, use the BACKUP_FILTER and RESTORE_FILTER configuration parameters to call external programs.

Tip: If you compress row data before backing it up, compressing the backup image with an external utility might not result in a smaller backup image.

The filter can be owned by anyone, but cannot have write access to non-privileged users. Permission on the filters is the same as that of permission on any other executable file that is called by the IBM® Informix® server or utilities.

ON-Bar backup and restore system

- Overview of the ON-Bar backup and restore system
  ON-Bar consists of various components and it works with a storage manager to back up and restore data.
- Configure the storage manager and ON-Bar
  The topics in this section provide the information that you need to plan and to set up ON-Bar with a storage manager.
- Back up with ON-Bar
  You can use the ON-Bar utility to back up and verify storage spaces (dbspaces, blobspaces, and sbspaces) and logical-log files.
- Restore data with ON-Bar
  You can use the ON-Bar utility to restore data that was backed up by the ON-Bar utility.
- External backup and restore
- Customize and maintain ON-Bar
- ON-Bar catalog tables
  These topics describe the ON-Bar tables that are stored in the **sysutils** database.
- ON-Bar messages and return codes
  ON-Bar prints informational, progress, warning, and error messages to the ON-Bar activity log file. ON-Bar return codes indicate the status of the command.

Related reference:
Backup and restore configuration parameters

Overview of the ON-Bar backup and restore system

ON-Bar consists of various components and it works with a storage manager to back up and restore data.

- ON-Bar components
  ON-Bar components include a command-line utility, catalog tables, an activity log, and an emergency boot file. You use ON-Bar with a storage manager and the XBSA shared library for the storage manager.

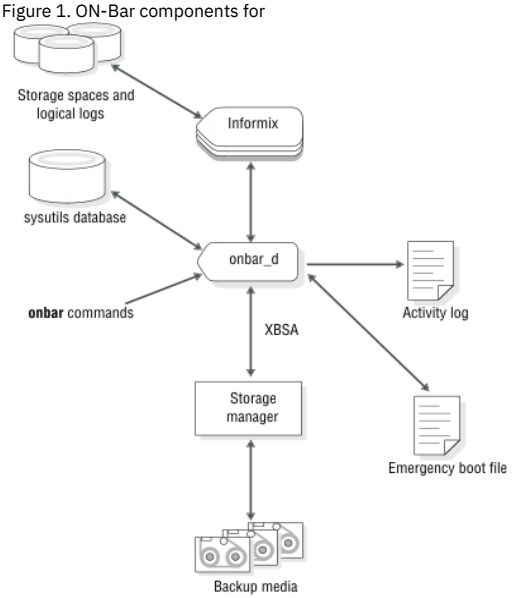
ON-Bar components

ON-Bar components include a command-line utility, catalog tables, an activity log, and an emergency boot file. You use ON-Bar with a storage manager and the XBSA shared library for the storage manager.

The following figure shows the ON-Bar and database server components:

- The storage spaces (dbspaces, blobspaces, and sbspaces) and logical logs from the database server
- The **sysutils** database, which includes ON-Bar catalog tables
- The **onbar** and the **onbar-d** command-line utilities
- The XBSA shared library for the storage manager on your system
- The storage media for storing backups
- The ON-Bar activity log
- The ON-Bar emergency boot file

Figure 1. ON-Bar components for



ON-Bar communicates with both the database server and the storage manager. You use the **onbar** command to start a backup or restore operation. By default, ON-Bar backs up and restores storage spaces in parallel. ON-Bar always processes log files serially.

For a backup session, ON-Bar requests the contents of storage spaces and logical logs from the database server and passes them to the storage manager. The storage manager stores the data on storage media. For a restore session, ON-Bar requests the backed up data from the storage manager and restores it on the database server.

ON-Bar backs up the critical dbspaces first, then the remaining storage spaces, and finally the logical logs. The *critical dbspaces* are the **rootdbs** and the dbspaces that contain the logical logs and physical log.

ON-Bar also places the following critical files in the archive during backups:

- The onconfig file
- UNIX: The sqlhosts file
- The ON-Bar emergency boot file: ixbar.*servernum*
- The server boot file: oncfg_*servername*.*servernum*

You can restore storage spaces stored in both raw and cooked files. If your system contains primary and mirror storage spaces, ON-Bar writes to both the primary and mirror chunks at the same time during the restore, except for an external restore.

ON-Bar status and error messages are written to the activity log file: bar_act.log.

- Backup Services API (XBSA)
  ON-Bar and the storage manager communicate through the Backup Services Application Programming Interface (XBSA), which enables the storage manager to manage media for the database server. By using an open-system interface to the storage manager, ON-Bar can work with various storage managers that also use XBSA.
- ON-Bar catalog tables
  ON-Bar uses the catalog tables in the **sysutils** database to track backup and restore operations. The **onsmsync** utility uses other catalog tables to track its operations.
- ixbar file: ON-Bar emergency boot file
  The emergency boot file is automatically updated after every backup. It contains the information that ON-Bar needs to perform a cold restore.
- bar_act.log file: ON-Bar activity log
  ON-Bar writes informational, progress, warning, error, and debugging messages to the ON-Bar activity log, bar_act.log.
- ON-Bar script
  The ON-Bar utility includes a shell script on UNIX and a batch script on Windows for customizing backup and restore operations.

Backup Services API (XBSA)

ON-Bar and the storage manager communicate through the Backup Services Application Programming Interface (XBSA), which enables the storage manager to manage media for the database server. By using an open-system interface to the storage manager, ON-Bar can work with various storage managers that also use XBSA.

Each storage manager develops and distributes a unique version of the XBSA shared library. You must use the version of the XBSA shared library provided with the storage manager. For example, if you use IBM® Informix® Primary Storage Manager, you must also use the XBSA shared library provided with ON-Bar. ON-Bar and the XBSA shared library must be compiled the same way (32-bit or 64-bit).

ON-Bar uses XBSA to exchange the following types of information with a storage manager:

Control data
: ON-Bar exchanges control data with a storage manager to verify that ON-Bar and XBSA are compatible, to ensure that objects are restored in the correct order to the correct instance of the database server, and to track the history of backup objects.

Backup or restore data
: During backups and restores, ON-Bar and the storage manager use XBSA to exchange data from specified storage spaces or logical-log files.

ON-Bar uses XBSA transactions to ensure data consistency. All operations included in a transaction are treated as a unit. All operations within a transaction must succeed for objects transferred to the storage manager to be restorable.

Related concepts:
IBM Informix Primary Storage Manager

ON-Bar catalog tables

ON-Bar uses the catalog tables in the **sysutils** database to track backup and restore operations. The **onsmsync** utility uses other catalog tables to track its operations.

ON-Bar uses the following catalog tables in the **sysutils** database to track backup and restore operations:

- The **bar_server** table tracks instances of the database server.
- The **bar_object** table tracks backup objects. A *backup object* is a backup of a dbspace, blobspace, sbspace, or logical-log file.
- The **bar_action** table tracks all backup and restore attempts against each backup object, except some log salvage and cold restore events.
- The **bar_instance** table describes each object that is backed up during a successful backup attempt.

The **onsmsync** utility uses and maintains the following tables to track its operations:

- The **bar_ixbar** table contains the history of all unexpired successful backups in all timelines.
- The **bar_syncdeltab** table is normally empty except when **onsmsync** is running.

For a description of the content of these tables, see ON-Bar catalog tables.

ixbar file: ON-Bar emergency boot file

The emergency boot file is automatically updated after every backup. It contains the information that ON-Bar needs to perform a cold restore.

Important: Do not modify the emergency boot file. Doing so might cause ON-Bar to select the wrong backup as part of a restore, possibly leading to data corruption or system failure.
The file name for the boot file is ixbar.*servernum*, where *servernum* is the value of the SERVERNUM configuration parameter.

The ON-Bar emergency boot file is in the $INFORMIXDIR/etc directory on UNIX and in the %INFORMIXDIR%\etc directory on Windows. You can override the default path and name of the boot file by changing the information specified in the BAR_IXBAR_PATH configuration parameter.

bar_act.log file: ON-Bar activity log

ON-Bar writes informational, progress, warning, error, and debugging messages to the ON-Bar activity log, bar_act.log.

ON-Bar backup and restore errors do not appear in standard output. If an error occurs when you back up and restore data, check information in the ON-Bar activity log

You can also use the activity log to:

- Monitor backup and restore activities such as, which storage spaces and logical logs were backed up or restored, the progress of the operation, and approximately how long it took.
- Verify whether a backup or restore succeeded.
- Track errors from the **ondblog** utility.
- Track ON-Bar performance statistics

The ON-Bar activity log is in the /tmp directory on UNIX and in the %INFORMIXDIR%\etc directory on Windows. You specify the location of the ON-Bar activity log with the BAR_ACT_LOG configuration parameter.

Related reference:
BAR_ACT_LOG configuration parameter
ON-Bar messages and return codes
View ON-Bar backup and restore performance statistics
onbar -m syntax: Monitoring recent ON-Bar activity

ON-Bar script

The ON-Bar utility includes a shell script on UNIX and a batch script on Windows for customizing backup and restore operations.

When you install ON-Bar with the database server, a default script is included. The name and location of the script depends on the operating system:

UNIX
: The **onbar** shell script is in the $INFORMIXDIR/bin directory.

Windows
: The **onbar.bat** batch script is in the %INFORMIXDIR%\bin directory.

When you issue ON-Bar commands from the command line, the arguments are passed to the script, and then to the **onbar_d** utility.

Table 1. ON-Bar utilities

Utility	Description
onbar_d utility	Transfers data between the database server and the storage manager. The **onbar** command calls the **onbar_d** utility that starts the **onbar-driver**. The **onbar-driver** starts and controls backup and restore activities.
onsmsync utility	Synchronizes the contents of the **sysutils** database, the emergency boot files, and the storage manager catalogs. Use this utility to purge backups that are no longer needed.
ondblog utility	Changes the database-logging mode. The **ondblog** utility logs its output in the ON-Bar activity log, bar_act.log.
archecker utility	Verifies backups, and restores table-level data from an archive.

Related concepts:
Overview of the archecker utility
Related reference:
The onsmsync utility

Configure the storage manager and ON-Bar

The topics in this section provide the information that you need to plan and to set up ON-Bar with a storage manager.

- Configure a storage manager
  ON-Bar backup and restore operations require a storage manager that integrates with ON-Bar through an XBSA shared library interface.
- Validating your storage manager
  When you convert or revert the IBM® Informix® database server, the storage manager that you used on the old version might not be validated for the version that you are migrating to. Verify that the storage-manager vendor successfully completed the IBM Informix validation process for the database server version and platform.

- Configuring ON-Bar
  Before you begin your first backup, review the default ON-Bar parameters in the onconfig file and adjust the values as needed. You can also set an environment variable.
- Verifying the configuration of ON-Bar and your storage manager
  Before you begin using ON-Bar and your storage manager, make sure that ON-Bar and your storage manager are set up correctly.
- Files that ON-Bar and storage managers use
  ON-Bar, IBM Informix Primary Storage Manager, and IBM Spectrum Protect use particular files in your installation.

Related tasks:
Preparing to back up data

Configure a storage manager

ON-Bar backup and restore operations require a storage manager that integrates with ON-Bar through an XBSA shared library interface.

You can choose to use the IBM® Informix® Primary Storage Manager, the IBM Spectrum Protect, or a third-party storage manager with ON-Bar. The Informix Primary Storage Manager is bundled with Informix. If you are using Spectrum Protect, the XBSA shared library needed for ON-Bar to communicate with Spectrum Protect is bundled with Informix.

The Informix Primary Storage Manager manages storage for ON-Bar backup and restore operations, including parallel backups, that use file devices (disks) only, not tapes. By default, the IBM Informix Primary Storage Manager is automatically configured with the information specified in Informix Primary Storage Manager and some ON-Bar configuration parameters. This storage manager is also automatically configured when you use the **onpsm** utility. You can change the configuration. For information, see IBM Informix Primary Storage Manager andConfiguring Informix Primary Storage Manager

- Storage-manager definitions in the sm_versions file
  Most storage managers must have an entry in the sm_versions file.
- Configuring Spectrum Protect
  To use IBM Spectrum Protect with IBM Informix databases, you must install and configure the IBM Spectrum Protect client on your database server computer and IBM Spectrum Protect on your storage computer.
- Configuring a third-party storage manager
  Storage managers have slightly different installation and configuration requirements. If you use a third-party storage manager, make sure that you follow the manufacturer instructions carefully. If you have difficulty with the storage-manager installation and configuration, contact the manufacturer directly.

Related concepts:
IBM Informix Primary Storage Manager
Setting up Informix Primary Storage Manager

Storage-manager definitions in the sm_versions file

Most storage managers must have an entry in the sm_versions file.

The IBM® Informix® Primary Storage Manager and IBM Spectrum Protect do not require an entry in the sm_versions file.

The storage-manager definition in the sm_versions file uses this format:

```
1|XBSA_ver|sm_name|sm_ver
```

In the format, *XBSA_ver* is the release version of the XBSA shared library for the storage manager, *sm_name* is the name of the storage manager, and *sm_ver* is the storage-manager version. The maximum field length is 128 characters.

Before ON-Bar starts a backup or restore process with the IBM Spectrum Protect and third-party storage managers, ON-Bar calls the currently installed version of the storage-manager-specific XBSA shared library to get its version number. If this version is compatible with the current version of ON-Bar and is defined in the sm_versions file, ON-Bar begins the requested operation.

Related tasks:
Updating the storage-manager definition in the sm_versions file for Spectrum Protect
Configuring a third-party storage manager

Configuring Spectrum Protect

To use IBM® Spectrum Protect with IBM Informix® databases, you must install and configure the IBM Spectrum Protect client on your database server computer and IBM Spectrum Protect on your storage computer.

You must also configure IBM Informix Interface for Spectrum Protect and perform other Spectrum Protect configuration tasks on your IBM Informix database server computer.

Starting with versions 8.1.2 and 7.1.8, IBM Spectrum Protect requires SSL communication between the client and the server, and therefore additional configuration steps must be followed:

To configure Spectrum Protect:

1. Edit the Spectrum Protect client options files.
2. Assign a Spectrum Protect management class for the server to use for backups.
3. Set the IBM Informix Interface for Spectrum Protect environment variables.
4. Register with the Spectrum Protect server.
5. Initialize the IBM Informix Interface for Spectrum Protect.

6. Optional. Configure ON-Bar to support optional Spectrum Protect features.

The version of the XBSA shared library for Spectrum Protect is 1.0.3.

For details about Spectrum Protect, read the following manuals:

- *IBM Spectrum Protect Backup-Archive Clients Installation and User's Guide*
- *IBM Spectrum Protect Using the Application Program Interface*
- *IBM Spectrum Protect Administrator's Guide*
- *IBM Spectrum Protect Administrator's Reference*

- Editing the Spectrum Protect client options files
  The IBM Informix Interface for Spectrum Protect communicates with the Spectrum Protect server with the Spectrum Protect API. By default, IBM Informix Interface for Spectrum Protect uses the client user options file (dsm.opt) and, on UNIX systems, the client system options file (dsm.sys), both of which are located in the Spectrum Protect API installation directory.
- Assigning a Spectrum Protect management class for a backup
  When you back up a database, the default management class for your node is *used*. You can override the default value with a different value that is specified in the INCLUDE option.
- Setting the IBM Informix Interface for Spectrum Protect environment variables
  When you use the IBM Informix Interface for TSM, you need to set certain environment variables in the environment of the user.
- Registering with the Spectrum Protect server
  Before backing up to and recovering from an IBM Spectrum Protect server, you must have a Spectrum Protect registered node name and a password. The process of setting up a node name and password is called *registration*.
- Initializing the IBM Informix Interface for Spectrum Protect password
  To initialize the password for IBM Informix Interface for Spectrum Protect, use the **txbsapswd** program. This program sets up a connection with the server instance that you specified in the dsm.opt file.
- Updating the storage-manager definition in the sm_versions file for Spectrum Protect
  You must update the storage-manager definition in sm_versions file for ON-Bar to use with IBM Spectrum Protect.
- Configuring ON-Bar for optional Spectrum Protect features
  You can configure ON-Bar to enable or disable optional features in IBM Spectrum Protect.

Editing the Spectrum Protect client options files

The IBM® Informix® Interface for Spectrum Protect communicates with the Spectrum Protect server with the Spectrum Protect API. By default, IBM Informix Interface for Spectrum Protect uses the client user options file (dsm.opt) and, on UNIX systems, the client system options file (dsm.sys), both of which are located in the Spectrum Protect API installation directory.

On UNIX systems, edit both the dsm.opt and the dsm.sys files as the **root** user:

- Specify the Spectrum Protect server to use in the client user options file, dsm.opt.
- Identify the Spectrum Protect server name, communication method, and server options in the client system options file, dsm.sys.

Use the sample dsm.opt.smp and dsm.sys.smp files distributed with the Spectrum Protect API to help you get started quickly.

On Windows systems, specify the Spectrum Protect server name, communication method, and server options in the dsm.opt file.

The following example shows a simple dsm.sys on UNIX/Linux systems:

```
SErvername  TSMSRV01
    TCPPort  9027
    TCPADMINPort 9028
    TCPServeraddress    tsmcentaur.myhcl.com
    PASSWORDACCESS      GENERATE
    PASSWORDDIR /work/TSM/PAX12/TSMPsswdDir
    NODENAME            pax12
    ERRORLOGName         /work/TSM/PAX12/dsierror.log

The TCPPort in this file should be same as the SSLTCPPORT used in the Spectrum Protect server.
The TCPADMINPort in this file should be same as the SSLTCPADMINPort used in the Spectrum Protect server.
```

See *Spectrum Protect Installing the Clients* and *Spectrum Protect Trace Facility Guide* for information regarding options you can specify in these files.

- Editing the Spectrum Protect client user options file
  You can edit the IBM Spectrum Protect client user options file, dsm.opt. This file must refer to the correct Spectrum Protect server instance, as listed in the dsm.sys file.
- Editing the Spectrum Protect client system options file
  You can edit the IBM Spectrum Protect client systems options file, dsm.sys. This file must refer to the correctSpectrum Protect server address and communication method.

Editing the Spectrum Protect client user options file

You can edit the IBM® Spectrum Protect client user options file, dsm.opt. This file must refer to the correct Spectrum Protect server instance, as listed in the dsm.sys file.

Set the following options in the dsm.opt file:

SERVERNAME
: Identifies which Spectrum Protect server instance, as listed in the dsm.sys file, that IBM Informix® Interface for Spectrum Protect contacts for services.

TRACEFILE
: Sends trace output information to a designated file.

TRACEFLAG

Sets specific trace flags

Editing the Spectrum Protect client system options file

You can edit the IBM® Spectrum Protect client systems options file, dsm.sys. This file must refer to the correctSpectrum Protect server address and communication method.

The following Spectrum Protect options are the most important to set in the dsm.sys file:

SERVERNAME
: Specifies the name that you want to use to identify a server when it is referred to in the dsm.opt file and to create an instance that contains options for that server.

COMMMETHOD
: Identifies the communication method.

TCPSERVERADDRESS
: Identifies the Spectrum Protect server.

PASSWORDACCESS
: Specifies GENERATE to store the Spectrum Protect password.

PASSWORDDIR
: Directory where the Spectrum Protect client will store the SSL encryption related files and keystore.

NODENAME
: Specifies the name of the Spectrum Protect client.
Note: Use the INFORMIXSERVER / DBSERVERNAME values in such a way that, it will be easy to identify in the Spectrum Protect catalog.

The SERVERNAME option in the dsm.opt and dsm.sys files define server instance names only. The TCPSERVERADDRESS option controls which server is contacted.

You can enable deduplication by including the DEDUP=CLIENTORSERVER option in the client system options file. You must also set the IFX_BAR_USE_DEDUP environment variable in the database server environment and restart the database server. See the *IBM Spectrum Protect Backup-Archive Client Installation and User's Guide* for information about configuring deduplication.

You can set up multiple server instances in the dsm.sys file. See the *IBM Spectrum Protect Backup-Archive Client Installation and User's Guide* for information about multiple server instances.

Related concepts:
Configuring ON-Bar for optional Spectrum Protect features
Related reference:
IFX_BAR_USE_DEDUP environment variable

Assigning a Spectrum Protect management class for a backup

When you back up a database, the default management class for your node is *used*. You can override the default value with a different value that is specified in the INCLUDE option.

The INCLUDE option is placed in the include-exclude options file. The file name of the include-exclude options file is in the client system options file (dsm.sys). For more information, see the *IBM Spectrum Protect Backup-Archive Client Installation and User's Guide*.

Use the following naming conventions for ON-Bar files:

- A database backup:
  /dbservername/dbservername/dbspacename/level
- A log backup:
  /dbservername/dbservername/server_number/unique_logid

For a database backup, an example of the INCLUDE statement is as follows:

```
Include /dbserverA/dbserverA/dbspaceA/* InformixDbMgmt
```

For a logical log backup, an example of the INCLUDE statement is as follows:

```
Include /dbserverA/dbserverA/55/* InformixLogMgmt
```

where the number `55` is the value of the SERVERNUM parameter in the onconfig file.

Setting the IBM Informix Interface for Spectrum Protect environment variables

When you use the IBM® Informix® Interface for TSM, you need to set certain environment variables in the environment of the user.

The following table describes these environment variables.

Table 1. IBM Informix Interface for Spectrum Protect environment variables

Environment variable	Description
DSMI_CONFIG	The fully qualified name for the client user option file (dsm.opt). The default value is dsm.opt in the TSM API installation directory.

Environment variable	Description
DSMI_DIR	On UNIX, points to the TSM API installed path. This environment variable needs to be defined only if the TSM API is installed in a different path from the default path. The DSMI_DIR environment variable is also used to find the dsm.sys file. On Windows, specifies the installation location of the TSM Backup-Archive Client. Typically, the TMS Backup-Archive Client is installed in the C:\Tivoli\TSMClient\baclient directory.
DSMI_LOG	Points to the directory that contains the API error log file (dsierror.log). For error log files, create a directory for the error logs to be created in, then set the DSMI_LOG environment variable to that directory. The user **informix** or the backup operator should have write permission on this directory.

The following example shows how to set up these environment variables for Solaris 32-bit if the TSM API is installed in the /opt/Tivoli/tsm/client/api directory:

```
export DSMI_CONFIG=/opt/Tivoli/tsm/client/api/bin/dsm.opt
export DSMI_DIR=/opt/Tivoli/tsm/client/api/bin
export DSMI_LOG=/home/user_a/logdir
```

The following example shows how to set up these environment variables for Windows if the TSM API is installed in the C:\Tivoli\TSMClient\api directory:

```
set DSMI_CONFIG=C:\Tivoli\TSMClient\api\BIN\dsm.opt
set DSMI_DIR=C:\Tivoli\TSMClient\baclient
set DSMI_LOG=C:\logdir
```

Registering with the Spectrum Protect server

Before backing up to and recovering from an IBM® Spectrum Protect server, you must have a Spectrum Protect registered node name and a password. The process of setting up a node name and password is called *registration*.

After the IBM Informix® Interface for Spectrum Protect node is registered with a Spectrum Protect server, you can begin using the IBM Informix Interface for Spectrum Protect to back up and restore your IBM Informix storage spaces and logical logs. If your workstation has a node name assigned to the Spectrum Protect backup-archive client, you should have a different node name for IBM Informix Interface for Spectrum Protect. For information about performing the registration process, see the *IBM Spectrum Protect Backup-Archive Client Installation and User's Guide*.

Initializing the IBM Informix Interface for Spectrum Protect password

To initialize the password for IBM® Informix® Interface for Spectrum Protect, use the **txbsapswd** program. This program sets up a connection with the server instance that you specified in the dsm.opt file.

You must run the **txbsapswd** program as user **root** before using IBM Informix Interface for TSM.

To initialize the password:

1. Create the directory for the SSL-related files as user "informix" (PASSWORDDIR in the dsm.sys file):

   ```
   $ sudo -u informix mkdir /work/TSM/PAX12/TSMPsswdDir
   ```

2. As user root, start the **txbsapswd** program located in $INFORMIXDIR/bin directory.
3. Enter the password and press Return. To retain your current password, press Return without a value.
4. Replace ownership and permissions in the PASSWORDDIR:

   ```
   $ sudo chown -R informix:informix  /work/TSM/PAX12/TSMPsswdDir
   ```

   ```
   $ sudo chmod -R 750 /work/TSM/PAX12/TSMPsswdDir
   ```

Updating the storage-manager definition in the sm_versions file for Spectrum Protect

You must update the storage-manager definition in sm_versions file for ON-Bar to use with IBM® Spectrum Protect.

Before ON-Bar starts a backup or restore process, it calls the currently installed version of the storage-manager-specific XBSA shared library to get its version number. If this version is compatible with the current version of ON-Bar and is defined in the sm_versions file, ON-Bar begins the requested operation.

To update the storage-manager definition in sm_versions file:

1. Copy the sm_versions.std template to a new file, sm_versions in the $INFORMIXDIR/etc directory on UNIX or the %INFORMIXDIR%\etc directory on Windows.
2. Put `tsm` in the **sm_name** field of the sm_versions file. The value `adsm` is also valid but will be deprecated in a future release.
3. Stop any ON-Bar processes (**onbar_d**, **onbar_w**, or **onbar_m**) that are currently running and restart them for the changes to take effect.

The following example shows the IBM Spectrum Protect definition in the sm_versions file:

```
1|5.3|tsm|5
```

Related reference:
Storage-manager definitions in the sm_versions file

Configuring ON-Bar for optional Spectrum Protect features

You can configure ON-Bar to enable or disable optional features in IBM® Spectrum Protect.

Apply all patches and updates to Spectrum Protect before you use the following Spectrum Protect features.

Deduplication
: Deduplication eliminates redundant data in backups. To enable the database server to support deduplication, set the IFX_BAR_USE_DEDUP environment variable in the Informix® environment and restart the database server. Update the Spectrum Protect client systems option file.

Large transfer buffer size
: The default transfer buffer size is 64 KB. Set the BAR_XFER_BUF_SIZE configuration parameter to specify a transfer buffer of up to 65 MB.
To limit the transfer buffer size to 64 KB regardless of the value of the BAR_XFER_BUF_SIZE configuration parameter, set the IFX_NO_LONG_BUFFERS environment variable to 1.

Replicate, import, and export backup objects
: Replicating, importing, or exporting backup objects between Spectrum Protect servers requires unique IDs for backup objects. ON-Bar automatically stores IDs in the metadata of backup objects that are unique for all Spectrum Protect server with version 12.10.xC2 or later.
To disable the ability to restore backup objects that are moved between Spectrum Protect servers, set the IFX_TSM_OBJINFO_OFF environment variable to 1.

Related reference:
IFX_BAR_USE_DEDUP environment variable
BAR_XFER_BUF_SIZE configuration parameter
Editing the Spectrum Protect client system options file
IFX_TSM_OBJINFO_OFF environment variable
IFX_BAR_NO_LONG_BUFFERS environment variable

Configuring a third-party storage manager

Storage managers have slightly different installation and configuration requirements. If you use a third-party storage manager, make sure that you follow the manufacturer instructions carefully. If you have difficulty with the storage-manager installation and configuration, contact the manufacturer directly.

For the list of certified storage managers for your ON-Bar version, consult your marketing representative.
Important: Some storage managers let you specify the data to back up to specific storage devices. Configure the storage manager to back up logical logs to one device and storage spaces to a different device for more efficient backups and restores.
To configure a third-party storage manager:

1. Set ON-Bar configuration parameters and environment variables.
2. Configure the storage manager so that ON-Bar can communicate correctly with it. For information, see your storage-manager documentation.
3. Configure your storage devices by following the instructions in your storage-manager documentation. The storage manager must know the device names of the storage devices that it uses.
4. Label your storage volumes.
5. Mount the storage volumes on the storage devices.
6. Create the storage-manager definition in the sm_versions file. Use the definition provided by the vendor of the third-party storage manager.
    a. Copy the sm_versions.std template to a new file, sm_versions in the $INFORMIXDIR/etc directory on UNIX or the %INFORMIXDIR%\etc directory on Windows.
    b. Create your own sm_versions file with the correct data for the storage manager by using the format in sm_versions.std as a template. To find out which code name to use in sm_versions for third-party storage managers, see the storage-manager documentation.
    c. Stop any ON-Bar processes (**onbar_d**, **onbar_w**, or **onbar_m**) that are currently running and restart them for the changes to take effect.
7. Verify that the BAR_BSALIB_PATH configuration parameter points to the correct XBSA shared library for your storage manager.
8. If you enabled deduplication for your storage manager, set the IFX_BAR_USE_DEDUP environment variable and restart the database server.
9. ON-Bar uses the value of the SERVERNUM configuration parameter as part of the storage path for the logical logs in the storage manager. If the storage manager does not use a wildcard for the server number, set the appropriate server number environment variable for the storage manager.

After you configure the storage manager and storage devices and label volumes for your database server and logical-log backups, you are ready to initiate a backup or restore operation with ON-Bar.

Related reference:
IFX_BAR_USE_DEDUP environment variable
Storage-manager definitions in the sm_versions file

Validating your storage manager

When you convert or revert the IBM® Informix® database server, the storage manager that you used on the old version might not be validated for the version that you are migrating to. Verify that the storage-manager vendor successfully completed the IBM Informix validation process for the database server version and platform.

If not, you need to install a validated storage manager before you perform backups with ON-Bar.

Configuring ON-Bar

Before you begin your first backup, review the default ON-Bar parameters in the onconfig file and adjust the values as needed. You can also set an environment variable.

You can configure the behavior of ON-Bar by setting the following configuration parameters and environment variable.

Table 1. ON-Bar configuration parameters and environment variable

Behavior	Configuration parameters

Behavior	Configuration parameters
Control the number and size of data buffers and the number of parallel processes.	BAR_NB_XPORT_COUNT configuration parameter BAR_XFER_BUF_SIZE configuration parameter IFX_BAR_NO_LONG_BUFFERS environment variable BAR_MAX_BACKUP configuration parameter BAR_MAX_RESTORE configuration parameter
Set the debugging level and the location of debug log file.	BAR_DEBUG configuration parameter BAR_DEBUG_LOG configuration parameter
Change the path of the ON-Bar boot file.	BAR_IXBAR_PATH configuration parameter
Maintain a history of expired backups.	BAR_HISTORY configuration parameter
Change the location and contents of ON-Bar activity log.	BAR_IXBAR_PATH configuration parameter
Maintain a history of expired backups.	BAR_ACT_LOG configuration parameter BAR_PROGRESS_FREQ configuration parameter BAR_PERFORMANCE configuration parameter
Set automatic retrying of failed back ups or restores.	BAR_RETRY configuration parameter
Allow a failed restore to be restarted.	RESTARTABLE_RESTORE configuration parameter
Increase backup size estimate sent to the storage manager.	BAR_SIZE_FACTOR configuration parameter
Extend the time an RS secondary server waits for a checkpoint during an external backup.	BAR_CKPTSEC_TIMEOUT configuration parameter
Configure continuous log backup.	ALARMPROGRAM configuration parameter
Filter or transform backed up data with an external program.	BACKUP_FILTER configuration parameter RESTORE_FILTER configuration parameter
Optimize the deduplication capabilities of storage managers.	IFX_BAR_USE_DEDUP environment variable
Disable the ability to replicate, import, or export backup objects among TSM servers.	IFX_TSM_OBJINFO_OFF environment variable
Force the use of the sm_versions file.	IFX_BAR_NO_BSA_PROVIDER environment variable

Do not set the LTAPEDEV configuration parameter to /dev/null or NUL because logical-log backups would be disabled and you can restore only whole-system backups.

- ON-Bar security
  By default, only the **informix** or **root** users on UNIX system or members of the **Informix-Admin** group on Windows systems can run ON-Bar commands.

ON-Bar security

By default, only the **informix** or **root** users on UNIX system or members of the **Informix-Admin** group on Windows systems can run ON-Bar commands.

To enable additional users to run ON-Bar commands:

- On UNIX systems, create a **bargroup** group and add users to the group. For instructions on how to create a group, see your UNIX documentation.
- On Windows systems, add the users to the **Informix-Admin** group.

Restriction: For security, it is recommended that ON-Bar commands not be run by the **root** user.
Related reference:
onbar -r syntax: Restoring data
onbar -b syntax: Backing up
onbar -v syntax: Verifying backups
onbar -m syntax: Monitoring recent ON-Bar activity
onbar -P syntax: Printing backed-up logical logs
onbar -RESTART syntax: Restarting a failed restore

Verifying the configuration of ON-Bar and your storage manager

Before you begin using ON-Bar and your storage manager, make sure that ON-Bar and your storage manager are set up correctly.

Verify your configuration by checking the items in the following list:

- The storage manager is installed and configured to manage specific storage devices.
- For UNIX, make sure that the BAR_BSALIB_PATH configuration parameter specifies correctly the XBSA shared library or it is not set and the library is in the default location.
- For Windows, make sure that the BAR_BSALIB_PATH configuration parameter specifies correctly the XBSA shared library.
- The sm_versions file contains a row that identifies the version number of the storage-manager-specific XBSA shared library.

After you verify that ON-Bar and your storage manager are set up correctly, run ON-Bar on your test database to make sure that you can back up and restore data. For more information, follow the instructions in Back up with ON-Bar.

Files that ON-Bar and storage managers use

ON-Bar, IBM® Informix® Primary Storage Manager, and IBM Spectrum Protect use particular files in your installation.

The following table lists the files that ON-Bar and IBM Spectrum Protect use and the directories where the files are. These names and locations change if you set up the onconfig file to values different from the defaults.

Table 1. List of files that ON-Bar and Spectrum Protect use

File name	Directory	Purpose
ac_config.std	UNIX: $INFORMIXDIR/etc Windows: %INFORMIXDIR%\etc	Template for **archecker** parameter values. The ac_config.std file contains the default **archecker** (archive checking) utility parameters. To use the template, copy it into another file and modify the values.
ac_msg.log	/tmp %INFORMIXDIR%\etc	The **archecker** message log. When you use **archecker** with ON-Bar to verify a backup, it writes brief status and error messages to the ON-Bar activity log and writes detailed status and error messages to the **archecker** message log. Technical Support uses the **archecker** message log to diagnose problems with backups and restores. Specify the location of the **archecker** message log with the AC_MSGPATH configuration parameter.
bar_act.log	/tmp %INFORMIXDIR%	ON-Bar activity log. For more information, see bar_act.log file: ON-Bar activity log.
bldutil.*process_id*	/tmp \tmp	When the **sysutils** database is created, error messages appear in this file.
dsierror.log	$DSMI_LOG	Spectrum Protect API error log.
dsm.opt	$DSMI_CONFIG	Spectrum Protect client user option file.
dsm.sys	$DSMI_DIR	Spectrum Protect client system option file.
Emergency boot files (ixbar* files)	$INFORMIXDIR/etc %INFORMIXDIR%\etc	Used in a cold restore. For more information, see ixbar file: ON-Bar emergency boot file.
oncfg_*servername.servernum*	$INFORMIXDIR/etc %INFORMIXDIR%\etc	Configuration information for ON-Bar restores. The database server creates the oncfg_*servername.servernum* file when you initialize disk space. The database server updates the file every time that you add or delete a dbspace, a logical-log file, or a chunk. The database server uses the oncfg* file when it salvages logical-log files during a cold restore. The database server uses the oncfg* files, so do not delete them.
save, savegrp, savefs	$INFORMIXDIR/bin	
sm_versions	$INFORMIXDIR/etc %INFORMIXDIR%\etc	Identifies the version of a third-party storage manager. To update the storage-manager version, edit the sm_versions file directly. The Informix Primary Storage Manager does not use the **sm_versions.std** file.

Back up with ON-Bar

You can use the ON-Bar utility to back up and verify storage spaces (dbspaces, blobspaces, and sbspaces) and logical-log files.

To perform a backup with ON-Bar:

1. Prepare for backup.
2. Back up with ON-Bar
3. Monitor backup progress.
4. Verify backups.
5. Back up storage manager information.

You can customize ON-Bar and storage manager commands in a shell or batch script. You can call ON-Bar from a job-scheduling program.

- Preparing to back up data
  Before you back up storage spaces and logical logs, you must prepare the system and copy critical administrative files.
- onbar -b syntax: Backing up
  Use the **onbar -b** command to back up storage spaces and logical logs.
- onbar -m syntax: Monitoring recent ON-Bar activity
  You can monitor recent ON-Bar activity with the **onbar -m** command. Only users who have permission to perform backup and restore operations can use this option.
- Viewing a list of registered backups
  You can create a list of the registered ON-Bar backups performed on your system.
- onbar -P syntax: Printing backed-up logical logs
  You can use the **onbar -P** command to print logical logs that are backed up using the ON-Bar utility.
- onbar -v syntax: Verifying backups
  Use the **onbar -v** command to verify that backups that were created by the ON-Bar utility are complete and can be restored.

Related reference:
Customizing ON-Bar and storage-manager commands

Preparing to back up data

Before you back up storage spaces and logical logs, you must prepare the system and copy critical administrative files.

To prepare to back up data:

1. Configure ON-Bar and your storage manager.
2. Ensure that you have enough logical log space. ON-Bar checks for available logical-log space at the beginning of a backup. If the logs are nearly full, ON-Bar backs up and frees the logs before attempting to back up the storage spaces. If the logs contain ample space, ON-Bar backs up the storage spaces, then the logical logs.
3. Verify that you have enough temporary disk space. The database server uses temporary disk space to store the before images of data that are overwritten while backups are occurring and overflow from query processing that occurs in memory. Verify that the DBSPACETEMP environment variable and DBSPACETEMP configuration parameter specify dbspaces that have enough space for your needs. If there is not enough room in the specified dbspaces, the backup will fail, root dbspace will be used, or the backup will fail after filling the root dbspace.
4. Back up administrative files to a different location.
5. Run the **oncheck -cD** command to verify that all database server data is consistent. You do not need to check for consistency before every level-0 backup. Do not discard a backup that is known to be consistent until the next time that you verify the consistency of your databases.

- Administrative files to back up
  Although ON-Bar backs up some critical administrative files, you must also include critical files in normal operating-system backups of important configuration files.

Related reference:
Configure the storage manager and ON-Bar
onbar -b syntax: Backing up
Related information:
oncheck -cd and oncheck -cD commands: Check pages

Administrative files to back up

Although ON-Bar backs up some critical administrative files, you must also include critical files in normal operating-system backups of important configuration files.

Files that ON-Bar backs up

When you back up a storage space, ON-Bar also backs up the following critical files:

- The onconfig file
- UNIX: The sqlhosts file
- The ON-Bar emergency boot file: ixbar.*servernum*
- The server boot file: oncfg_*servername.servernum*

You must restore these files if you need to replace disks or if you restore to a second computer system (imported restore).
Look at the bar_act.log file to determine whether critical files are successfully backed up. The return code for the **onbar -b** command indicates only whether storage spaces are successfully backed up. The following lines from the bar_act.log file show that the ON-Bar emergency boot file, ixbar.0, is backed up:

```
Begin backup of critical file '/opt/informix-11.70.fc7/etc/ixbar.0'.
Completed backup of critical file '/opt/informix-11.70.fc7/etc/ixbar.0'
```

Files that you must manually back up

In addition to the critical files, you must also manually back up the following administrative files:

- The sm_versions file
- Storage-manager configuration and data files
- Simple-large-object data in blobspaces that are stored on disks
- Externally stored data such as external tables that a DataBlade maintains
- The keystore and stash files for encrypting storage spaces, as specified by the DISK_ENCRYPTION configuration parameter

Tip: Even though ON-Bar includes the critical files with the files it backs up, it is a good practice to also include the critical files in your system archive. Having the critical files included in both the IBM® Informix® and system archives gives you more options if you need them.

Files that ON-Bar re-creates

Although ON-Bar does not back up the following items, ON-Bar automatically re-creates them during a restore. You do not need to make backup copies of these files:

- The dbspace pages that are allocated to the database server but that are not yet allocated to a tblspace extent
- Mirror chunks, if the corresponding primary chunks are accessible
- Temporary dbspaces
  ON-Bar does not back up or restore the data in temporary dbspaces. Upon restore, the database server re-creates empty temporary dbspaces.

onbar -b syntax: Backing up

Use the **onbar -b** command to back up storage spaces and logical logs.

To run ON-Bar commands, you must be user **root**, user **informix**, a member of the **bargroup** group on UNIX, or a member of the Informix®-Admin group on Windows.

- Usage
- Example: Back up a whole system
- Example: Back up all online storage spaces and logical logs
- Example: Perform an incremental backup

- Example: Back up specified storage spaces and all logical logs
- Example: Back up a list of storage spaces specified in a file
- Example: Back up logical logs
- Example: Physical backup

Syntax for backing up with ON-Bar

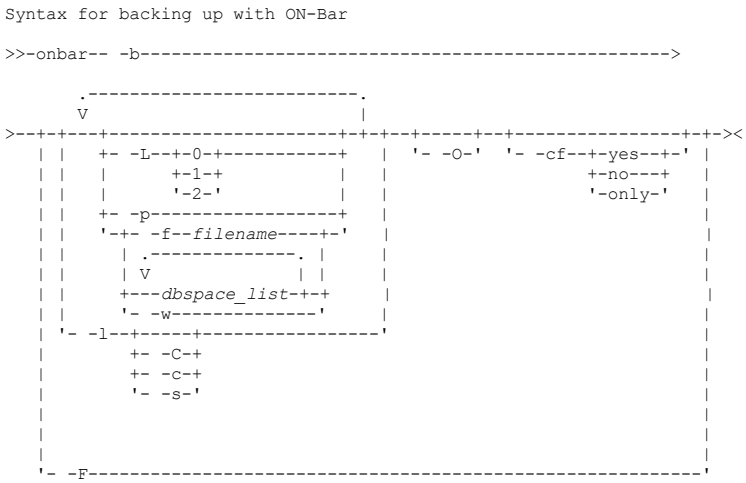


Table 1. Options for the onbar -b command

Option	Description
-b	Specifies a backup Backs up the storage spaces and logical logs, including the current logical log.
dbspace_list	Specifies the storage spaces to be backed up, separated by blank spaces. If you do not enter *dbspace_list* or -f *filename*, ON-Bar backs up all online storage spaces on the database server.
-c	Closes and backs up the current logical log and the other full logical logs.
-C	Starts a continuous log backup. Reserve a dedicated storage device and terminal window because the continuous log backups run indefinitely waiting for logical logs to fill. To stop a continuous log backup, stop the ON-Bar process with an interrupt command, such as CTRL-C or SIGTERM.
-cf	Specifies whether the critical files are backed up. The critical files are the onconfig file, the sqlhosts file, and the ixbar.*servernum* file. Valid values are: • **yes** = Backs up the critical files. Default when performing a level 0, 1, or 2 backup. • **no** = Does not back up the critical files. Default when backing up the logical log files. • **only** = Backs up only the critical files.
-f *filename*	Backs up the storage spaces that are listed in the text file that is specified by the *filename* value. Use this option to avoid entering a long list of storage spaces every time that you back up. For more information, see List of storage spaces in a file.
-F	Performs a fake backup A storage-manager application is not necessary. No backup actually occurs, so no restore is possible from a fake backup. Use fake backups sparingly, if at all. Fake backups might be appropriate in the following situations: • Change database logging modes • Change a RAW table to a STANDARD table • Allow the user to use new logs, chunks, or mirrors without performing a backup • In special situations when you, the administrator, judge that a backup is not needed
-L	Specifies the level of backup to perform on storage spaces: • 0 = a complete backup (Default) • 1 = changes since the last level-0 backup • 2 = changes since the last level-1 backup If you request an incremental backup and ON-Bar finds that no previous level backup was performed for a particular storage space, ON-Bar backs up that storage space at the previous level. For example, if you request a level-1 backup, and ON-Bar finds no level-0 backup, it makes a level-0 backup instead.
-l	Performs a backup of full logical-log files. The current logical-log file is not backed up.

Option	Description
-O	Overrides normal backup restrictions. Use this option to back up logical logs when blobspaces are offline. If a log backup occurs when blobspaces are offline, return code 178 displays in the ON-Bar activity log.
-p	Backs up only physical storage spaces without logical logs. A warning message is written to the activity log listing the log unique ID of the latest log file that is required for a restore of the storage spaces. Use this option if logical logs are being continuously backed up. If necessary, a log switch is initiated, so that this log can be backed up. If the current log is already newer than the log with the archive checkpoint of the last storage space, then no log switch is initiated.
-s	Salvages any logical logs that are still on disk after a database server failure. You can run the **onbar -l -s** command while the server is offline. If possible, use this option before you replace a damaged disk. If you use **onbar -r** to perform a cold restore on an undamaged disk, ON-Bar automatically salvages the logical logs.
-w	Backs up a whole system, which includes all storage spaces and logical logs based on a single checkpoint. The time of the backup is stored with the backup information. The data in all storage spaces is consistent in a whole-system backup, therefore, you do not need to restore the logical logs to make the data consistent. If you do not save the logical logs, you must use the **-w** option.

Usage

Before you back up your data, make sure that your data is consistent by running the **oncheck -cD** command.

To run ON-Bar commands, you must be user **root**, user **informix**, or a member of the **bargroup** group on UNIX, or a member of the **Informix-Admin** group on Windows. For more information, see ON-Bar security.

You can back up storage spaces and logical logs when the database server is in online, quiescent, or fast-recovery mode.

The storage-space chunks can be stored on raw disk storage space, in cooked files, or on an NTFS file system (Windows).

Only online storage spaces are backed up. Use the **onstat -d** command to determine which storage spaces are online. During a backup, if ON-Bar encounters a down dbspace, it skips it and later returns an error. If a storage space is offline, restart the backup when the storage space is back online.

After you begin the backup, monitor its progress in the ON-Bar activity log and database server message log.

You can either back up the logical logs separately or with storage spaces. Back up the logical logs as soon as they fill so that you can reuse them and to protect against data loss if the disks that contain the logs are lost. If the log files fill, the database server pauses until you back up the logical logs. You can either back up the logical logs manually or start a continuous logical-log backup by running the **onbar -b -C** command. Logical-log backups are always level 0. After you close the current logical log, you can back it up.

If you perform whole-system backups and restores, you do not need to restore logical logs. However, back up the logical logs when you use whole-system backups. These log backups allow you to recover your data to a time after the whole-system backup, minimizing data loss.

If you are running continuous logical log backup and then start a whole system backup, the ON-Bar process attempts to save the logical logs. Because the continuous logical log backup is running, an error message is returned indicating that a logical log backup is already running, and the whole system backup returns a non-zero error code. In this case the logical logs are backed up only one time. To avoid the error, create a physical backup with the **onbar -b -w -p** command.

To back up a specific table or set of tables in ON-Bar, store these tables in a separate dbspace and then back up this dbspace. Alternatively, you can perform table level restores with the **archecker** utility.

Example: Back up a whole system

The following command performs a level-0 whole system backup after taking a checkpoint of all online storage spaces and logical logs:

```
onbar -b -w
```

The following command performs a level-1 whole system backup:

```
onbar -b -w -L 1
```

Example: Back up all online storage spaces and logical logs

The following command performs a standard, level-0 backup of all online storage spaces and used logical logs:

```
onbar -b
```

Example: Perform an incremental backup

The following command performs a standard, level-1 backup:

```
onbar -b -L 1
```

Example: Back up specified storage spaces and all logical logs

The following command performs a level-0 backup of the dbspaces named **fin_dbspace1** and **fin_dbspace2** and all logical logs:

```
onbar -b fin_dbspace1 fin_dbspace2
```

Example: Back up a list of storage spaces specified in a file

The following sample file named listfile3 contains a list of storage spaces to be backed up:**blobsp2.1**, **my_dbspace1**, **blobsp2.2**, **dbsl.1**, **rootdbs.1**, and **dbsl.2**.

```
blobsp2.1
# a comment                      ignore this text

       my_dbspace1            # back up this dbspace
; another comment
blobsp2.2                      dbsl.1
rootdbs.1      dbsl.2    ; backing up two spaces
```

The following command backs up the storage spaces listed in the listfile3 file:

```
onbar -b -f listfile3
```

Example: Back up logical logs

The following command starts a manual logical-log backup:

```
onbar -b -l
```

The following command backs up the current logical-log file:

```
onbar -b -l -c
```

Example: Physical backup

The following command backs up all storage spaces without backing up any logical logs:

```
onbar -b -p -L 0
```

A warning message is written to the ON-Bar activity log file stating that log file backup was not initiated. The message also contains the log unique ID of the latest log file that is required for a restore of the storage spaces. The latest required log file contains the archive checkpoint of the last dbspace backed up.

Example message:

```
2011-12-14 09:30:35 14277  14275 (-43354) WARNING: Logical logs were
     not backed up as part of this operation. Logs through log unique ID 9
     are needed for restoring this backup. Make sure these logs are backed
     up separately.
```

- List of storage spaces in a file
  You can list storage spaces to back up or restore in a file.
- Backing up blobspaces
  You can back up blobspaces in a database that uses transaction logging.

Related tasks:
Configuring a continuous log restore by using ON-Bar
Replacing disks during a restore
Preparing to back up data
Related reference:
Plan a backup system for a production database server
ON-Bar security

List of storage spaces in a file

You can list storage spaces to back up or restore in a file.

The *filename* value can be any valid UNIX or Windows file name:

- Simple file names, for example: listfile_1)
- Relative file names, for example: ../backup_lists/listfile_2 or ..\backup_lists\listfile2
- absolute file names, for example: /usr//backup_lists/listfile3 or c:\\backup_lists\listfile3

The format rules for the file are:

- If you are restoring chunks, list storage space names without paths. Each line can list more than one storage space, separated by spaces or a tab.
- If you are renaming chunks, list the old chunk path name, the old offset, the new chunk path name, and the new offset. Put a blank space or a tab between each item. Put information for each chunk on a separate line.
- Comments begin with a # or a ; symbol and continue to the end of the current line.
- ON-Bar ignores all comment or blank lines in the file.

Backing up blobspaces

You can back up blobspaces in a database that uses transaction logging.

Before you back up a new blobspace, make sure that the log file that recorded the creation of the blobspace is no longer the current log file. You can run the **onstat -l** command to verify the logical-log status.

When users update or delete simple large objects in blobspaces, the blobpages are not freed for reuse until the log file that contains the delete records is freed. To free the log file, you must back it up.

Important: If you perform a warm restore of a blobspace without backing up the logical logs after updating or deleting data in it, that blobspace might not be restorable. To back up blobspaces:

1. Verify the logical-log status by running the **onstat -l** or **xctl onstat -l** command.
2. Switch to the next log file by running the **onmode -l** command.
3. Back up the logical logs:
   - If the blobspace is online, run the **onbar -b -l -c** command.
   - If the blobspace is offline, run the **onbar -b -l -O** or **onbar -b -O** command. If this backup is successful, ON-Bar returns `178`.
4. Back up the blobspaces by running the **onbar -b** or **onbar -b -w** command.

Related information:
onstat -L command: Print the number of free locks

onbar -m syntax: Monitoring recent ON-Bar activity

You can monitor recent ON-Bar activity with the **onbar -m** command. Only users who have permission to perform backup and restore operations can use this option.

```
Monitor recent ON-Bar activity

             .-20--------.
>>-onbar -m--+-+--------+-+--+----------------------+-----------><
               '-lines-'    |       .-5-----------. |
                            '- -r--+-+---------+-+-'
                                   '-seconds-'
```

Table 1. Options for the onbar -m command

Option	Description
-m	Prints the recent activity of ON-Bar from the activity log file.
lines	Specifies the number of lines to output. Default is 20 lines.
-r	Causes the **onbar -m** command to repeat.
seconds	Specifies the number of seconds to wait before repeating. Default is 5 seconds.

Related concepts:
bar_act.log file: ON-Bar activity log
Related reference:
ON-Bar messages and return codes
Message format in the ON-Bar message log
ON-Bar security

Viewing a list of registered backups

You can create a list of the registered ON-Bar backups performed on your system.

To view the list of registered backups:

1. Create a view in the **sysutils** database that contains information from the **bar_action**, **bar_instance**, and **bar_object** catalog tables. Include the following fields in the view:
   - Backup_ID: The internally generated ID for the backup
   - Type: Defines whether the backup is a whole system backup, dbspace backup, or logical log backup.
   - Object_Name: The name of the object backed up.
   - Ifx_Time: Time at which the object was created. For dbspace backups, the checkpoint time that started the backup. For logical logs, the time when the log become full.
   - CopyID_HI: The High part of the ID to locate the object in the storage manager.
   - CopyID_LO: The Low part of the ID to locate the object in the storage manager.
   - Backup_Start: Date and time when the backup started for this object
   - Backup_End: Date and time when the backup ended for this object.
   - Verify_Date: The time of the last verification made to this object, if any.
2. Run a SELECT statement against the view.

Example

The following statement creates a view that contains backup information:

```
CREATE VIEW list_backups(Backup_ID, Type, Object_Name, Ifx_Time, CopyID_HI,
            CopyID_LO, Backup_Start, Backup_End, Verify_Date)
AS SELECT * FROM (
SELECT
        act_aid AS backup_id,
        DECODE(act_type, 5, "Whole-System", DECODE(obj_type, "L",
           "Logical log", "Dbspace")) AS Type,
        substr(obj_name,1, 8) AS Object_Name,
```

```
        min(DBINFO ('utc_to_datetime', seal_time)) AS Ifx_Time,
        ins_copyid_hi AS CopyID_HI,
        ins_copyid_lo AS CopyID_LO,
        act_start AS Backup_Start,
        act_end AS Backup_End,
        ins_verify_date AS Verify_Date

FROM
        bar_action A,
        bar_instance I,
        bar_object O
WHERE
        A.act_aid = I.ins_aid AND
        A.act_oid = O.obj_oid AND
        A.act_oid = I.ins_oid AND
        O.obj_type in ("R", "CD", "ND", "L")
GROUP BY 1,2,3,5,6,7,8,9
ORDER BY Ifx_Time, Backup_ID) AS view_list_backups
```

The following query returns all the backups:

```
SELECT * FROM list_backups
```

Related information:
The sysutils Tables

onbar -P syntax: Printing backed-up logical logs

You can use the **onbar -P** command to print logical logs that are backed up using the ON-Bar utility.

To run ON-Bar commands, you must be user **root**, user **informix**, a member of the **bargroup** group on UNIX, or a member of the Informix®-Admin group on Windows.

- Usage
- Example: Print a specific transaction
- Example: Print multiple logical log files

Print backed-up logical logs

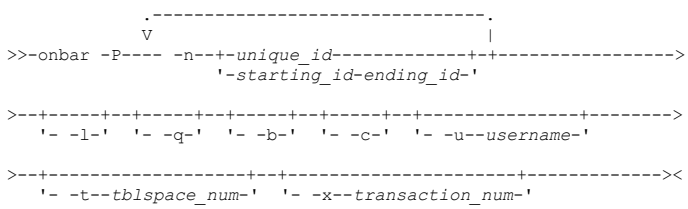


Table 1. Options for the onbar -P command

Option	Purpose
-b	Print logical-log records associated with blobspace blobpages. The database server stores these records on the logical-log backup media as part of blobspace logging.
-c	Use the compression dictionary to expand compressed data.
-l	Print the long listing of the logical-log record. The long listing of a log record includes complex hexadecimal and ASCII dumps of the entire log record.
-n *starting_id-ending_id*	Print the logical-log records contained in the specified range of log files. The *starting_id* option is the ID of the first log to print. The *ending_id* option is ID of the last log to print. The value of the *starting_id* option must be smaller than the value of the *ending_id* option. Separate the starting and ending ID values with a hyphen. Do not include blank spaces.
-n *unique_id*	Print the logical-log records contained in the specified log file. The *unique_id* option is the unique ID number of the logical log. To determine the unique ID of a specific logical-log file, use the **onstat -l** command.
-P	Print backed-up logical log information
-q	Do not print the program header
-t *tblspace_num*	Print the records associated with the tblspace that you specify with the *tblspace_num* option. Specify the *tblspace_num* value as either an unsigned integer or hexadecimal value. If you do not use a prefix of `0x`, the value is interpreted as an integer. The integer must be greater than zero and must exist in the **partnum** column of the **systables** system catalog table.
-u *username*	Print the records for a specific user. The user name must be an existing login name and conform to operating-system-specific rules for login names.

Option	Purpose
-x *transaction_num*	Print only the records associated with the transaction that you specify. The *transaction_num* must be an unsigned integer between zero and TRANSACTIONS -1, inclusive. Additional Information: Use the **-x** option only in the unlikely situation of an error being generated during a roll-forward. When this situation occurs, the database server sends a message to the message log that includes the transaction ID of the offending transaction. You can use this transaction ID with the **-x** option to investigate the cause of the error.

Usage

To view the backed-up logical logs, the storage manager must be running.

The output of this command is printed to **stdout**.

Example: Print a specific transaction

The following command prints information about a single transaction that was performed by the user **informix** against the tblspace 1048722 and is contained in the logical log file 2:

```
onbar -P -n 2 -l -q -b -u "informix" -t 1048722 -x 1
```

The output for this command might be:

```
log uniqid: 2.
1665d0   120  DPT      1        2  0        5
         00000078 0002006c 00000010 0000fefe ...x...l ........
         00000001 00000000 000077e3 00000000 ........ ..w.....
         00000005 00000005 00002a24 00000001 ........ ..*$....
         00100004 0a0c21b8 00002a48 00000001 ......!. ..*H....
         00100006 0a0c2288 00002ea1 00000001 ......". ........
         0010001b 0a0c3810 00002bee 00000001 ......8. ..+.....
         00100015 0a0c3a18 00002a3d 00000001 ......:. ..*=....
         00100005 0a0c57c0                   ......W.
166648   60   CKPOINT  1        0  1665d0   1
         0000003c 00000042 00000010 0000fefe ...<...B ........
         00000001 001665d0 000077e3 00000000 ......e. ..w.....
         00010005 00000002 00000002 001665a0 ........ ......e.
         00000007 ffffffff 00084403          ........ ..D.
```

Example: Print multiple logical log files

The following command prints the logical log records for the logical logs files that have IDs of 2, 3, 4, 5, 10, 11, and 12:

```
onbar -P -n 2-5 -n 10-12
```

Related reference:
ON-Bar security
Related information:
onstat -l command: Print physical and logical log information
onstat -L command: Print the number of free locks
SYSTABLES

onbar -v syntax: Verifying backups

Use the **onbar -v** command to verify that backups that were created by the ON-Bar utility are complete and can be restored.

To run ON-Bar commands, you must be user **root**, user **informix**, a member of the **bargroup** group on UNIX, or a member of the Informix®-Admin group on Windows.

Sufficient temporary space must be available. For more information, see Temporary space for backup verification.


```
Verify backups

>>-onbar - -v--+-----+--+-------------+--+---------------+-----><
              '- -p-'  '- -t--"time"-'  +- -f--filename-+
                                        | .-------.     |
                                        | V       |     |
                                        +---space-+-----+
                                        '- -w-----------'
```

Table 1. Options for the onbar -v command

Option	Description

Option	Description
-v	Verifies a backup. The server can be in any mode. If verification is successful, you can restore the storage spaces safely. You can verify a whole-system or physical-only backup. You cannot verify the logical logs.
space	Names of storage spaces to verify. If you enter more than one storage-space name, use a space to separate the names.
-f *filename*	Verifies the storage spaces that are listed in the text file whose path name *filename* provides. Use this option to avoid entering a long list of storage spaces every time that you verify them. You can use any valid UNIX or Windows path name and file name. For the format of this file, see List of storage spaces in a file. The file can list multiple storage spaces per line.
-p	Verifies a physical-only backup.
-t "*time*"	Specifies the date and time to which dbspaces are verified. Must be surrounded by quotation marks. How you enter the time depends on your current GLS locale convention. If the GL_DATETIME environment variable is set, you must specify the date and time according to that variable. If the GLS locale is not set, use ANSI-style date format: `YYYY-MM-DD HH:MM:SS`.
-w	Verifies a whole-system backup.

Usage

The **onbar -v** command runs the **archecker** utility. The **archecker** utility verifies that all pages required to restore a backup exist on the media in the correct form. After you successfully verify a backup, you can restore it safely.

When you verify a backup, ON-Bar writes summary messages to the bar_act.log that report which storage spaces were verified and whether the verification succeeded or failed. The **archecker** utility writes detailed messages to the ac_msg.log. Software Support uses the ac_msg.log to diagnose problems with backups and restores.

The **onbar -v** command verifies only the smart-large-object extents in an sbspace. For a complete check, use the **oncheck -cS** command.

The **onbar -v** command cannot verify the links between data rows and simple large objects in a blobspace. Use the **oncheck -cD** command instead to verify the links in a blobspace.

Example: Perform a point-in-time verification of a backup

The following command verifies a backup at a point-in-time:

```
onbar -v -t "2011-12-10 10:20:50"
```

Example: Verify backups of storage spaces listed in a file

The following command verifies the backed-up storage spaces that are listed in the file bkup1:

```
onbar -v -f /usr/backups/bkup1
```

Example: ON-Bar activity log verification messages

The following examples show messages about verification in the ON-Bar activity log:

The level-0 backup of dbspace **dbs2.2** passed verification, as follows:

```
Begin backup verification of level0 for dbs2.2 (Storage Manager Copy ID:##)
Completed level-0 backup verification successfully.
```

The level-0 backup of **rootdbs** failed verification, as follows:

```
Begin backup verification of level0 for rootdbs (Storage Manager Copy ID:##).
ERROR: Unable to close the physical check: error_message.
```

Example: archecker message log verification messages

More detailed information is available in the **archecker** message log, as follows:

```
STATUS: Scan PASSED
STATUS: Control page checks PASSED
STATUS: Starting checks of dbspace dbs2.2.
STATUS: Checking dbs2.2:TBLSpace
.
.
STATUS: Tables/Fragments Validated: 1
Archive Validation Passed
```

- Temporary space for backup verification
  When you verify backups, 15-25 MB of temporary space must be available.
- Verification failures
  The verification of a backup can fail for various reasons. If a backup fails verification, do not attempt to restore it.

Related tasks:

Verifying an expired backup
Related reference:
ON-Bar security

Temporary space for backup verification

When you verify backups, 15-25 MB of temporary space must be available.

During backup verification, the **archecker** utility requires about 15 MB of temporary space for a medium-size system (40-50 GB) and 25 MB for a large system. This temporary space is stored on the file system in the directory that the AC_STORAGE parameter specifies, not in the dbspaces. The temporary files contain bitmap information about the backup and copies of partition pages, free pages in a chunk, reserved pages, and optionally, free pages in a blobspace and debugging information. The **archecker** utility must have permissions to the temporary directory.

If the backup is verified successfully, these files are deleted. If the backup fails verification, these files remain. Copy them to another location so that Software Support can review them.

If your database server contains only dbspaces, use the following formula to estimate the amount of temporary space in KB for the **archecker** temporary files:

```
space = (130 KB * number_of_chunks) + (pagesize * number_of_tables) +
(.05 KB * number_of_logs)
```

For IBM® Informix®, if your database server contains blobspaces or sbspaces, use the following formula to estimate the amount of temporary space for the **archecker** temporary files:

```
space = (130 KB * number_of_chunks) + (pagesize * number_of_tables) +
(.05 KB * number_of_logs) + (pagesize * (num_of_blobpages/252))
```

number_of_chunks
: The maximum number of chunks that you estimate for the database server.

pagesize
: The system page size in KB.

number_of_tables
: The maximum number of tables that you estimate for the database server.

number_of_logs
: The number of logical logs on the database server.

num_of_blobpages
: The number of blobpages in the blobspaces or the number of sbspaces. (If your database server contains sbspaces, substitute *num_of_blobpages* with the number of sbspaces.)

For example, you would need 12.9 megabytes of temporary disk space on a 50-gigabyte system with a page size of 2 KB. This system does not contain any blobspaces, as the following statement shows:

```
13,252 KB = (130 KB * 25 chunks) + (2 KB * 5000 tables) +
            (.05 KB * 50 logs) + (2 KB * 0)
```

To convert KB to MB, divide the result by 1024:

```
12.9 MB = 13,252/1024
```

Verification failures

The verification of a backup can fail for various reasons. If a backup fails verification, do not attempt to restore it.

The causes of a verification failure are unpredictable and range from corruption of the database server to a failed restore because ON-Bar cannot find the backup object on the storage manager. In fact, the restore might appear to be successful but it hides the real problem with the data or media.

Backups with corrupted pages

If the pages are corrupted, the problem is with the databases rather than with the backup or the media.

Run **oncheck -cd** on any tables that produce errors and then redo the backup and verification. To check extents and reserved pages, run **oncheck -ce** and **oncheck -cr**.

Backups with corrupted control information

In this case, all the data is correct, but some of the backup control information is incorrect, which might cause problems with the restore. Ask Software Support for assistance.

Backups with missing data

When a backup is missing data, it might not be recoverable. After a data loss, try to restore from an older backup. Then restore the current logical logs.

Backups of inconsistent database server data

There are cases where **archecker** returns “success” to ON-Bar but shows “failure” in the **archecker** message logs. This situation occurs when **archecker** verifies that ON-Bar backed up the data correctly, but the database server data was invalid or inconsistent when it was backed up.

- Diagnosing why a backup failed verification
  If a backup failed verification, you can take steps to diagnose and attempt to fix the problem.
- Verifying an expired backup
  You can verify an expired backup in case subsequent backups are not valid.
- Restoring when a backup is missing data
  If a backup fails verification because of missing data, you can perform a restore from an older backup.

Diagnosing why a backup failed verification

If a backup failed verification, you can take steps to diagnose and attempt to fix the problem.

To diagnose why a backup failed verification:

1. Verify that the AC_CONFIG environment variable and the contents of the **archecker** configuration file are set correctly. If these variables are set incorrectly, the ON-Bar activity log prints a message.
2. Back up the data onto different media.
   Do not reuse the original backup media because it might be damaged.

   Do not use any backups based on this backup. If the level-0 backup failed verification, do not use the corresponding level-1 and level-2 backups.
3. Verify this new backup. If verification succeeds, you can restore the storage spaces.
4. Use your storage manager to expire the backup that failed verification and then run the **onsmsync** utility without arguments to remove the bad backup from the **sysutils** and emergency boot files.
5. If verification fails again, call Software Support and provide them with the following information:
   - Your backup tool name (ON-Bar)
   - The database server online.log
   - The **archecker** message log
   - The AC_STORAGE directory that contains the bitmap of the backup and copies of important backed-up pages

   If only part of the backup is corrupted, Software Support can help you determine which portion of the backup can be restored in an emergency.

   Software Support might advise you to run **oncheck** options against a set of tables.

Verifying an expired backup

You can verify an expired backup in case subsequent backups are not valid.

To verify an expired backup:

1. Check that the status of the backup save set on the storage manager. If the storage manager expired the backup save set, the **archecker** utility cannot verify it.
2. Use the storage-manager commands for activating the expired backup save set. See your storage-manager documentation.
3. Run the **onbar -v** command again.

Related reference:
onbar -v syntax: Verifying backups

Restoring when a backup is missing data

If a backup fails verification because of missing data, you can perform a restore from an older backup.

To restore when a backup is missing data:

1. Choose the date and time of an older backup than the one that failed. To perform a point-in-time verification, use the **onbar -v -t** *time space* command.
2. If the older backup passes verification, perform a point-in-time physical restore by using the same *time* value, then perform a log restore, as follows:

   ```
   onbar -r -p -t time space
   onbar -r -l
   ```
3. Expire the corrupted backup at your storage manager.
4. Run the **onsmsync** command without arguments. The **onsmsync** utility removes backups that are no longer held by the storage manager from the emergency boot file and the **sysutils** database, preventing ON-Bar from attempting to use such backups.

Related reference:
The onsmsync utility
onbar -r syntax: Restoring data

Restore data with ON-Bar

You can use the ON-Bar utility to restore data that was backed up by the ON-Bar utility.

Before you restore data, use the pre-restore checklist to determine if whether a restore is needed and to prepare for a restore.

To perform a restore with the ON-Bar utility:

1. Make the storage devices that were available during the backup available for the restore.
2. If necessary, add enough temporary space to perform the restore. The logical log restore portion of a warm restore requires temporary space. The minimum amount of temporary space is equal to the smaller of the total amount of allocated logical-log space and the number of log files to be replayed.
3. Run the **onbar -r** command with the appropriate options to restore the data.
4. Monitor the ON-Bar activity log.
5. After the restore is complete, run the **onstat -d** command to verify that all storage spaces are restored. The letter O in the flags column indicates that the chunk is online.

- Pre-restore checklist
  Use this checklist to determine if a restore is necessary and to prepare for a restore.
- Storage device availability
  Verify that the storage devices and files used in the backup are available for the restore.
- onbar -r syntax: Restoring data
  To run a complete restore, use **onbar -r** command.
- Replacing disks during a restore
  You can replace disks during a restore by renaming chunks. You rename chunks by specifying new chunks paths and offsets during a cold restore with ON-Bar. This option is useful if you need to restore storage spaces to a different disk from the one on which the backup was made. You can rename any type of chunk, including critical chunks and mirror chunks.
- Restoring to a different computer
  You can back up data on one computer and restore the data on a different computer. Importing a restore is useful for disaster recovery or upgrading a database server. After you back up your data and move over the storage-manager objects, you can perform an imported restore. An imported restore involves copying files from the source to the target computer and performing the restore in one of several ways.
- onbar -RESTART syntax: Restarting a failed restore
  If a failure occurs with the database server, media, storage manager, or ON-Bar during a restore, you can restart the restore from the place that it failed. To restart a failed restore, the RESTARTABLE_RESTORE configuration parameter must be set to ON in the onconfig file when the restore fails.
- Resolve a failed restore
  How you resolve a failed restore depends on the cause of the failure.

Pre-restore checklist

Use this checklist to determine if a restore is necessary and to prepare for a restore.

To prepare for a restore:

- Determine if you need to restore. If one or more of these problems is true, you perform a restore to fix the problem:
  - Has data been lost or corrupted?
  - Does a committed transaction error need to be undone?
  - Is the database server down or has a disk failed?
  - Is a storage space or chunk down or inconsistent?
- Diagnose the problem by using database server monitoring tools.
- If the root dbspace or the dbspaces that contain the physical log and logical-log files need to be restored, you must perform a cold restore. The database server must be offline during a cold restore. Ask your client users to log off the system.
- Contact the appropriate vendor to resolve the following types of problems before doing a restore:
  - The storage manager
  - The XBSA connection
  - The operating system
  - The storage media

- Storage space status and required actions
  To determine the state of each storage space and its chunks, examine the output of the **onstat -d** command. The storage space status determines the action you need to take to solve the problem. The database server must be online.

Related information:
Database server monitoring

Storage space status and required actions

To determine the state of each storage space and its chunks, examine the output of the **onstat -d** command. The storage space status determines the action you need to take to solve the problem. The database server must be online.

The following table describes **onstat -d** command output about chunk status and the actions required to solve the problems. The chunk status information is in the second position of the **flags** column in the first (storage spaces) and second (chunks) sections of the output.

Table 1. Chunk flag descriptions and required actions

chunk flag	Storage space or chunk state	Action required
(No flag)	Storage space no longer exists.	Perform a point-in-time cold restore to a time before the storage space was dropped.
D	Chunk is down or storage space is disabled.	Perform a warm restore of the affected storage space.
I	Chunk is physically restored, but needs a logical restore.	Perform a logical restore.
L	Storage space is being logically restored.	Try the logical restore again.
N	Chunk is renamed and either down or inconsistent.	Perform a warm restore of the chunk when the physical device is available.

chunk flag	Storage space or chunk state	Action required
O	Chunk is online.	No action required.
P	Storage space is physically restored.	Perform a logical restore, if one is not already in progress.
R	Storage space is being restored.	Perform a physical or logical restore.
X	Storage space or chunk is newly mirrored.	No action required.

Related reference:
onbar -r syntax: Restoring data
Related information:
onstat -d command: Print chunk information

Storage device availability

Verify that the storage devices and files used in the backup are available for the restore.

If you drop a dbspace or mirror device after a level-0 backup, the dbspace or mirror device must be available to the database server when you begin the restore. If the storage device is not available, the database server cannot write to the chunk and the restore fails.

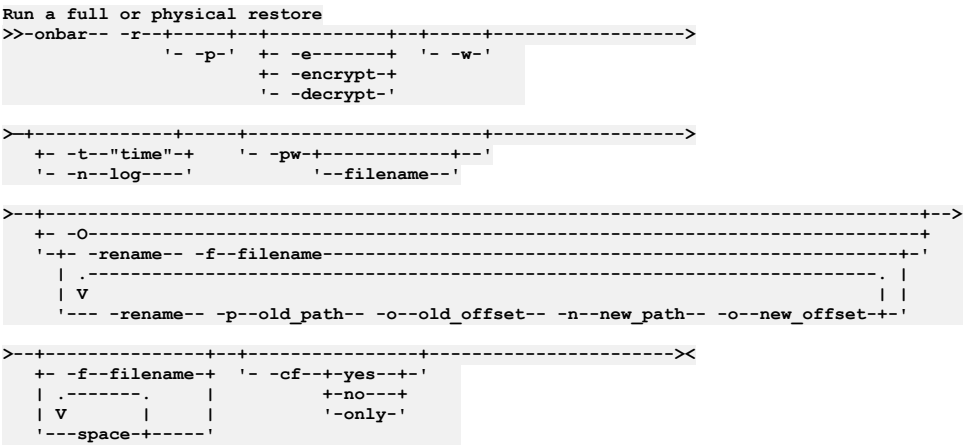
If you add a chunk after your last backup, the chunk device must be available to the database server when it rolls forward the logical log.

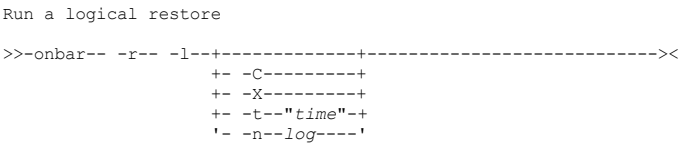
onbar -r syntax: Restoring data

To run a complete restore, use **onbar -r** command.

To run ON-Bar commands, you must be user **root**, user **informix**, a member of the **bargroup** group on UNIX, or a member of the Informix®-Admin group on Windows.

- Usage
- Example: Perform a whole-system restore
- Example: Restore specific storage spaces
- Example: Perform a warm restore in stages
- Example: Point-in-time restore
- Example: Point-in-time restore with a French locale
- Example: Point-in-time restore in stages
- Example: Restore a dropped storage space and chunks
- Example: Restore critical files





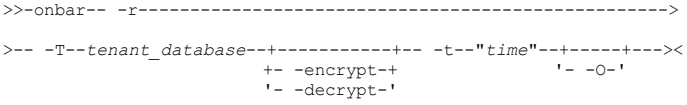


Table 1. Options for the onbar -r command.

Option	Description
-r	Specifies a restore. If the database server is offline, ON-Bar performs a cold restore. If the database server is in online, quiescent, or fast recovery mode, ON-Bar performs a warm restore. In a cold restore, the **-r** option restores all storage spaces and salvages and restores the logical logs. In a warm restore, the **-r** option restores all storage spaces that are offline and restores the logical logs. You must specify the **-r** option first.
space	Specifies which storage spaces to back up as a list of one or more dbspace, blobspace, or sbspace names, separated by blank spaces. ON-Bar restores only the storage spaces listed. If the database server is offline, you must list all the critical dbspaces. You cannot specify temporary spaces.
-C	Continuously restores logical logs from the current logical log tape without sending prompts to mount the tape. The server is placed in suspend log restore state, and the command exits after the last applicable log is restored. The server sends a prompt if a log spans tapes. The configuration parameter RESTARTABLE_RESTORE does not affect continuous log restoration.
-cf	Specifies whether the critical files are restored during a cold restore. The critical files are the onconfig file, the sqlhosts file (on UNIX), the oncfg_*servername*.*servernum* file, and the ixbar.*servernum* file. Valid values are: • **yes** = Restores the critical files. • **no** = Default. Does not restore the critical files. • **only** = Restores only the critical files.
-decrypt	Specifies to decrypt any encrypted storage spaces during the physical restore of the spaces.
-encrypt	Specifies to encrypt storage spaces during the physical restore of the spaces. Storage space encryption must be enabled by the DISK_ENCRYPTION configuration parameter. Otherwise, the restore fails. For more information about storage space encryption, see Changing storage space encryption during a restore.
-e	Specifies an external restore. Run the **onbar -r -e** command after you externally restore the storage spaces. Marks storage spaces as physically restored, restores the logical logs, and brings the storage spaces online.
-f *filename*	Specifies the path and file name of a text file that lists the storage spaces to restore or rename. Use this option to avoid entering a long list of storage spaces. For more information, see List of storage spaces in a file.
-l	Specifies a logical restore only. Restores and rolls forward the logical logs. The logical restore applies only to those storage spaces that are already physically restored. Important: To improve performance, replay logical-log transactions in parallel during a warm restore. Use the ON_RECVRY_THREADS configuration parameter to set the number of parallel threads. To replay logical-log transactions in parallel during a cold restore, use the OFF_RECVRY_THREADS configuration parameter. For more information, see your *IBM® Informix Performance Guide*.
-n *log*	Indicates the unique ID of the last logical log to be restored in a cold restore. The database server must be offline. To find the unique ID, use the **onstat -l** command. A point-in-log restore is a special point-in-time restore. You must restore all storage spaces in a point-in-log restore so that the data is consistent. If any logical logs exist after the specified log, ON-Bar does not restore them and their data is lost. If the specific logical log applies to more than one timeline, ON-Bar uses the latest one. Cannot be combined with the **-t** option.
-n *new_path*	Specifies the new path of the chunk. Use with the **-rename** option.
-O	Overrides internal error checks. Allows the restore of online storage spaces. Forces the recreation of chunk files that no longer exist. Used to override internal error checks to perform the following tasks: • Force the restore of online storage spaces. If a storage space in the list of storage spaces to restore is online, ON-Bar takes the storage space offline and then restores it. If this operation succeeds, ON-Bar completes with an exit code of 177. • Force the creation of nonexistent chunk files. If a chunk file for a storage space being restored no longer exists, ON-Bar recreates it. The newly created chunk file is cooked disk space, not raw disk space. If ON-Bar successfully recreates the missing chunk file, ON-Bar completes with an exit code of 179. • Force a cold restore to proceed if a critical storage space is missing. In a cold restore, ON-Bar checks whether every critical space is being restored. This check occasionally causes false warnings. If the warning was valid, the restore fails. If the warning was false and ON-Bar successfully restores the server, ON-Bar completes with an exit code of 115. Use the **-O** option with a whole-system restore only to recreate missing chunk files. You cannot use the **onbar -r -w -O** command when the database server is online because the root dbspace cannot be taken offline during the whole-system restore. Cannot be combined with the **-rename** option.

Option	Description
-pw *[file name]*	The -pw option is required only when the storage space encryption feature is enabled and no stash file is in use. Supply an optional path to a file containing the keystore password, otherwise onbar will prompt for a password before performing the restore.
-o *new_offset*	Specifies the offset of the renamed chunk. Use with the **-rename** option.
-o *old_offset*	Specifies the offset of the chunk to be renamed. Use with the **-rename** option.
-p	Specifies a physical restore only. You must follow a physical restore with a logical restore before data is accessible unless you use a whole-system restore. This option turns off automatic log salvage before a cold restore. If the LTAPEDEV configuration parameter is set to /dev/null or NUL, you must use the **-p** option during a restore.
-p *old_path*	Specifies the path of the chunk to be renamed. Use with the **-rename** option.
-rename	Renames one or more chunks during a cold restore. The database server must be offline. This option is useful if you need to restore storage spaces to a different disk from the one on which the backup was made. You can rename any type of chunk, including critical chunks and mirror chunks. You can rename chunks that have level-0 backups. Cannot be combined with the **-O** option.
-T *tenant_database*	Restores a tenant database. The database server must be online. No other warm restores or tenant restores can be in progress. If you include the **-t** option, the data is restored to the specified point in time. If you do not include the **-t** option, the data is restored to the current time. Include the **-O** option unless all of the permanent tenant storage spaces are marked as down. Temporary storage spaces are never backed up or restored. For more information, see Restoring a tenant database to a point in time.
-t "*time*"	Specifies the time of the last transaction to be restored from the logical logs in a cold restore or a tenant database point-in-time restore. For a cold restore, the database server must be offline. For a tenant database point-in-time restore, the database server must be online. All storage spaces specified are restored to the same point in time. However, for a cold restore, if you perform a physical restore followed by a logical restore, the logical restore can be to a later point in time. For example you might detect that your current backup is corrupted, and that you need to restore the previous backup. In this case, start your physical restore with the timestamp of your previous backup, and subsequently start the logical recovery to a more recent timestamp. A point-in-time restore is typically used to recover from a mistake. For example, if you accidentally dropped a database, you can restore the server to a point in time just before you dropped the database. To determine the appropriate date and time for the point-in-time restore, use the **onlog** utility. The **onlog** utility output shows the date and time of the committed transactions in the logical log. All data transactions that occurred after the time you specify in the restore command are lost. Use quotation marks around the date and time. The format for the English locale is `yyyy-mm-dd hh:mm:ss`. If the GL_DATETIME environment variable is set, you must specify the date and time according to that variable. Cannot be combined with the **-n** *log* option.
-w	Performs a whole-system restore of all storage spaces and logical logs from the last whole-system backup. The database server must be offline. After the whole-system restore is complete, the server is in quiescent mode. If you specify **onbar -r -w** without a whole-system backup, return code 147 is returned because ON-Bar cannot find any storage spaces backed up as part of a whole-system backup.
-X	Stops continuous logical log restore. Leaves the server in quiescent mode in a logical restore suspend state without restoring additional logs.

Usage

You can restore storage spaces stored in both raw and cooked files. If your system contains primary and mirror storage spaces, ON-Bar writes to both chunks simultaneously during the restore, except for an external restore. You cannot specify to restore temporary spaces. When you restore the critical dbspaces (for example, the root dbspace), the database server recreates the temporary dbspaces, but they are empty.

ON-Bar restores the storage spaces in parallel if the BAR_MAX_BACKUP or BAR_MAX_RESTORE configuration parameter is set to a value greater than 1. To speed up restores, you can add additional CPU virtual processors.

You can restore noncritical storage spaces in a warm restore, when the database server is online, in the following circumstances:

- The storage space is online, but one of its chunks is offline, recovering, or inconsistent.
- The storage space is offline or down.

You cannot perform more than one warm restore simultaneously. If you need to restore multiple storage spaces, specify the set of storage spaces to restore to ON-Bar or allow ON-Bar to restore all down storage spaces by not explicitly specifying any spaces.

Tip: For faster performance in a restore, assign separate storage devices for backing up storage spaces and logical logs. If physical and logical backups are mixed together on the storage media, it takes longer to scan the media during a restore.
In certain situations, you might want to perform a restore in stages. If multiple devices are available for the restore, you can restore multiple storage spaces separately or concurrently, and then perform a single logical restore.

By default, ON-Bar restores the latest backup. If you do not want to restore the latest backup, you can restore from an older backup: for example, when backup verification failed or the backup media was lost. You can perform a point-in-time restore or a point-in-log restore. Alternatively, you can expire a bad backup in the storage manager, run the **onsmsync** command, and then restore from the older backup. If you accidentally drop a storage space, you can use a point-in-time restore or a point-in-log restore to recover it.

You can force a restore of online storage spaces (except critical dbspaces) by using the **-O** option. The database server automatically shuts down each storage space before it starts to restore it. Taking the storage space offline ensures that users do not try to update its tables during the restore process.

You can restore critical files during a cold restore by including the **-cf yes** option.

You can rename chunks by specifying new chunks paths and offsets during a cold restore with ON-Bar. This option is useful if you need to restore storage spaces to a different disk from the one on which the backup was made. You can rename any type of chunk, including critical chunks and mirror chunks.

When storage space encryption is enabled, by default storage spaces retain the same encryption state after a restore as during the back up. You can specify to encrypt or decrypt a storage space during a restore with the **-encrypt** or **-decrypt** options.

Example: Perform a whole-system restore

A whole-system restore is a cold restore and must be performed while the server is offline. The following command restores a whole-system backup:

```
onbar -r -w
```

Example: Restore specific storage spaces

The following example restores two specific storage spaces, **fin_dbspace1** and **fin_dbspace2**:

```
onbar -r fin_dbspace1 fin_dbspace2
```

Example: Perform a warm restore in stages

The following commands perform a physical restore, back up logical logs, and perform a logical restore:

```
onbar -r -p
onbar -b -l
onbar -r -l
```

Example: Point-in-time restore

The following command restores database server data to its state at a specific date and time:

```
onbar -r -t "2011-05-10 11:35:57"
```

In this example, the restore replays transactions that committed on or before the specified time, including any transactions with a commit time of 11:35:57. Transactions in progress but not committed by 11:35:57 are rolled back.

Example: Point-in-time restore with a French locale

The default date and time format for the French locale, `fr_fr.8859-1`, uses the format "`%A %.1d %B %iY %H:%M:%S`."

The following command restores the data to a specific point in time that is formatted for the French locale:

```
onbar -r -t "Lundi 6 Juin 2011 11:20:14"
```

You can set GL_DATETIME to a different date and time format that uses the date, month, two-digit year, hours, minutes, and seconds. For example:

```
%.1d %B %iy %H:%M:%S
```

The following command restores to a specific point in time by specifying a two-digit year for the French locale:

```
onbar -r -t "6 Juin 11 11:20:14"
```

Example: Point-in-time restore in stages

The following commands perform a physical restore and a logical restore to the same point in time:

```
onbar -r -p -t "2011-05-10 11:35:57"
onbar -r -l -t "2011-05-10 11:35:57"
```

Example: Restore a dropped storage space and chunks

Suppose that a transaction dropped a storage space named **dbspace1** and deleted chunks at the time 2011-05-10 12:00:00. The following command restores the storage space and recreates the deleted chunks while the server is offline:

```
onbar -r -t "2011-05-10 11:59:59" -O
```

Example: Restore critical files

The following command restores data and the critical files during a cold restore:

```
onbar -r -cf yes
```

- Avoid salvaging logical logs
  The **onbar -r** command automatically salvages the logical logs. However, avoid salvaging logical logs in some situations.
- Performing a cold restore
  If a critical storage space is damaged because of a disk failure or corrupted data, you must perform a cold restore. If a disk fails, you need to replace it before you can perform a cold restore to recover data.
- Configuring a continuous log restore by using ON-Bar
  Use continuous log restore to keep a second system (hot backup) available to replace the primary system if the primary system fails.
- Restoring data by using a mixed restore
  You can use mixed restore to reduce the time until urgent data becomes online and available when you need to restore the server. Urgent data is data that you deem as critical to your business operation.
- Recreating chunk files during a restore
  If the disk or file system fails, one or more chunk files might be missing from the dbspace. Use the -O option to recreate missing chunk files and any necessary directories during a restore.
- Reinitializing the database server and restoring data
  Reinitializing disk space destroys all existing data managed by the database server. However, you can restore data from a backup that was performed before reinitialization.

Related tasks:
Restoring when a backup is missing data
Replacing disks during a restore
Related reference:
Storage space status and required actions
External restore commands
ON-Bar security
onbar -RESTART syntax: Restarting a failed restore

Avoid salvaging logical logs

The **onbar -r** command automatically salvages the logical logs. However, avoid salvaging logical logs in some situations.

Use the **onbar -r -p** and **onbar -r -l** commands to skip log salvage.

If you set the LTAPEDEV configuration parameter to /dev/null on UNIX or to NUL on Windows, the logical logs are not salvaged in any ON-Bar restore (**onbar -r** or **onbar -r -w**, for example).

Avoid salvaging the logical logs in the following situations:

- When you perform an imported restore
  Salvage the logical logs on the source database server but not on the target database server.
- If you reinitialize the database server (**oninit -i**) before you perform a cold restore
  Reinitialization creates new logical logs that do not contain the data that you want to restore.
- If you install a new disk for the dbspace that contains the logical logs
  Salvage the logs from the old disk, but not from the new disk.

Related tasks:
Restoring to a different computer

Performing a cold restore

If a critical storage space is damaged because of a disk failure or corrupted data, you must perform a cold restore. If a disk fails, you need to replace it before you can perform a cold restore to recover data.

If you try to perform a cold restore without a backup, data in the storage spaces that were not backed up are lost.
To perform a cold restore:

1. Shut down the server by running the **onmode -ky** command.
2. If the disk that contains the logical-log files must be replaced or repaired, use the **onbar -b -l -s** command to salvage logical-log files on the damaged disk.
   Otherwise, ON-Bar automatically salvages the logical logs.
3. If necessary, repair or replace the damaged disk.
4. If the files in INFORMIXDIR are damaged, copy the back ups of administrative files to their original locations.
   Otherwise, you do not need to copy the administrative files.
5. Restore the critical and noncritical storage spaces by running the **onbar -r** command. When the restore is complete, the database server is in quiescent mode.
6. Start the server by running the **onmode -m** command.
7. Synchronize the storage manager by running the **onsmsync** command.

Related reference:
The onsmsync utility
Related information:
onmode -k, -m, -s, -u, -j: Change database server mode

Configuring a continuous log restore by using ON-Bar

Use continuous log restore to keep a second system (hot backup) available to replace the primary system if the primary system fails.

The version of IBM® Informix® must be identical on both the primary and secondary systems.
To configure continuous log restore by using ON-Bar:

1. On the primary system, perform a level-0 backup with the **onbar -b -L 0** command.
2. Import the backup objects that were created to the storage manager of the secondary server.
3. On the secondary system, perform a physical restore with the **onbar -r -p** command. After the physical restore completes on the secondary system, the database server waits in fast recovery mode to restore logical logs.
4. On the primary system, back up logical logs with the **onbar -b -l** command.
5. Transfer the backed up logical logs to the secondary system and restore them with the **onbar -r -l -C** command.
6. Repeat steps 4 and 5 for all logical logs that are available to back up and restore.
7. If you are doing continuous log restore on a secondary system as an emergency standby, run the following commands to complete restoring logical logs and quiesce the server:
   - If logical logs are available to restore, run the **onbar -r -l** command.
   - After all available logical logs are restored, run the **onbar -r -l -X** command.

Related concepts:
Continuous log restore
Related reference:
onbar -b syntax: Backing up

Restoring data by using a mixed restore

You can use mixed restore to reduce the time until urgent data becomes online and available when you need to restore the server. Urgent data is data that you deem as critical to your business operation.

In a mixed restore, you first perform a cold restore of the critical dbspaces (the root dbspace and the dbspaces that contain the physical and logical logs) and the dbspaces containing your urgent data. Because you do not restore all dbspaces, you can bring the server online faster. You then restore the remaining storage spaces in one or more warm restores.

To perform a mixed restore:

1. Shut down the database server by running the **onmode -ky** command.
2. Perform a cold restore of the critical and urgent dbspaces by running the **onbar -r** command with the list of critical and urgent dbspace names. You can specify a point in time to restore from an older backup.
3. Start the server by running the **onmode -m** command.
4. Synchronize the storage manager by running the **onsmsync** command.
5. Perform a warm restore of the remaining storage spaces by running the **onbar -r** command. You can perform multiple warm restores to prioritize certain storage spaces.

Examples

Example 1: Simple mixed restore

A database server has five dbspaces in addition to the root dbspace: **logdbs**, **dbs_1**, **dbs_2**, **dbs_3**, and **dbs_4**. The logical logs are stored in **logdbs** and the physical log is in the root dbspace. The critical dbspaces that must be restored during the initial cold restore are **rootdbs** and **logdbs**. The dbspace that contains urgent data is **dbs_1**. The following commands shut down the database server, perform a cold restore on the critical and urgent dbspaces, and restart the database server:

```
onmode -ky
onbar -r rootdbs logdbs dbs_1
onmode -m
```

After the database server starts, any data stored in **rootdbs**, **logdbs**, and **dbs_1** dbspaces is accessible.

The following commands synchronize the storage manager and perform a warm restore of the remaining dbspaces, **dbs_2**, **dbs_3**, and **dbs_4**:

```
onsmsync
onbar -r
```

Example 2: Point-in-time mixed restore

The following commands perform a cold restore for a subset of the storage spaces (including all critical dbspaces) in the initial cold restore, perform a warm restore for **dbspace_2** and **dbspace_3**, followed by a warm restore of **dbspace_4** and **dbspace_5**, and finally perform a warm restore of all remaining storage spaces:

```
onbar -r -t "2011-05-10 11:35:57" rootdbs logspace_1 dbspace_1
onmode -m
onsmsync
onbar -r dbspace_2 dbspace_3
onbar -r dbspace_4 dbspace_5
onbar -r
```

- Strategies for using a mixed restore
  To implement a mixed-restore strategy, carefully select the set of dbspaces in which you place your databases and database objects when you create them.

Strategies for using a mixed restore

To implement a mixed-restore strategy, carefully select the set of dbspaces in which you place your databases and database objects when you create them.

ON-Bar backs up and restores physical, not logical, entities. Thus, ON-Bar cannot restore a particular database or a particular set of tables. Instead, ON-Bar restores a particular set of storage spaces. It is up to you to track what is stored in those storage spaces.

For example, consider a database with the catalogs in the dbspace **cat_dbs**:

```
create database mydb in cat_dbs with log;
```

A table in this database is fragmented among the dbspaces **tab_dbs_1** and **tab_dbs_2**:

```
create table mytab (i integer, c char(20))
fragment by round robin in tab_dbs_1, tab_dbs_2;
```

An index for the table is stored in the dbspace **idx_dbs**:

```
create index myidx on mytab(i) in idx_dbs;
```

If you need to restore the server, you cannot access all of the data in the example database until you restore the dbspaces containing the database catalogs, table data, and index: in this case, the dbspaces **cat_dbs**, **tab_dbs_1**, **tab_dbs_2**, and **idx_dbs**.

To simplify the management and tracking of your data, divide your set of dbspaces into subsets in which you store data of a particular urgency. When you create your database objects, place them in dbspaces appropriate to their urgency. For example, if you have data with three levels of urgency, you might want to place all the objects (database catalogs, tables, and indexes) associated with your most urgent data in a particular set of dbspaces: for example, **urgent_dbs_1**, **urgent_dbs_2**, ...**urgent_dbs_*n***. You would place all the objects associated with less urgent data in a different set of dbspaces: for example, **less_urgent_dbs_1**, **less_urgent_dbs_2**, ... **less_urgent_dbs_*k***. Lastly, you would place your remaining data in a different set of dbspaces: for example, **non_urgent_dbs_1**, **non_urgent_dbs_2**, ... **non_urgent_dbs_*r***.

If you need to restore the server, you would first perform a cold restore of all critical dbspaces and dbspaces containing urgent data, **urgent_dbs_1** through **urgent_dbs_*n***. For example, assume that logical logs are distributed among two dbspaces, **logdbsp_1** and **logdbsp_2**, and the physical log is in **rootdbs**. The critical **dbspaces** are therefore **rootdbs**, **logdbsp_1**, and **logdbsp_2**.

You would perform the initial cold restore by issuing the following ON-Bar command:

```
onbar -r rootdbs logdbsp_1 logdbsp_2 urgent_dbs_1 ...  urgent_dbs_2
```

You can bring the server online and all business-urgent data is available.

Next, perform a warm restore for the less-urgent data:

```
onsmsync
onbar -r less_urgent_dbs_1 less_urgent_dbs_2 .....  less_urgent_dbs_k
```

Finally, you can perform a warm restore for the rest of the server by issuing the following command.

```
onbar -r
```

In a larger system with dozens of dbspaces, you can divide the warm restore portion of the mixed restore into several warm restores, each restoring only a small subset of the dbspaces remaining to be restored in the system.

Recreating chunk files during a restore

If the disk or file system fails, one or more chunk files might be missing from the dbspace. Use the -O option to recreate missing chunk files and any necessary directories during a restore.

The restore fails if insufficient space exists on the file system. The newly created chunk files are cooked files and are owned by group **informix** on UNIX or group **Informix-Admin** on Windows.

- Restoring when using cooked chunks
  You can recreate missing cooked chunk files during a restore.
- Restoring when using raw chunks
  You can recreate missing raw chunk files during a restore.

Restoring when using cooked chunks

You can recreate missing cooked chunk files during a restore.

Restriction: ON-Bar does not recreate chunk files during a logical restore if the logical logs contain chunk-creation records.

To restore when using cooked chunks:

1. Install the new disk.
2. Based on your system, perform one of the following tasks:
   - On UNIX, mount the device as a file system.
   - On Windows, format the disk.
3. Allocate disk space for the chunk file.
4. Run the **onbar -r -O** *space* command to recreate the chunk files and restore the dbspace.

Related information:
Allocating cooked file spaces on UNIX

Restoring when using raw chunks

You can recreate missing raw chunk files during a restore.

To restore when using raw chunks:

1. Install the new disk.
2. For UNIX, if you use symbolic links to raw devices, create new links for the down chunks that point to the newly installed disk. ON-Bar restores the chunk file to where the symbolic link points.
3. Issue the **onbar -r** *space* command to restore the dbspace.

Related information:
Allocating raw disk space on UNIX
Allocating raw disk space on Windows

Reinitializing the database server and restoring data

Reinitializing disk space destroys all existing data managed by the database server. However, you can restore data from a backup that was performed before reinitialization.

You must have a current, level-0 backup of all storage spaces.
During initialization, ON-Bar saves the emergency boot file elsewhere and starts a new, empty emergency boot file. Therefore, any backups that you performed before reinitializing the database server are not recognized. You must use the copy of the emergency boot file you saved before initialization to restore the previous database server instance.

To reinitialize the database server and restore the old data:

1. Copy the emergency boot file, the oncfg file, and the onconfig file to a different directory.
2. Set the FULL_DISK_INIT configuration parameter to 1 in the onconfig file.
3. Shut down the database server.
4. Reinitialize the database server by running the **oninit -i** command.
5. Move the administrative files into the database server directory. If the administrative files are unavailable, copy them from the last backup into the database server directory.
6. Perform a restore by running the **onbar -r -p -w** command. Do not salvage the logical logs.
7. Verify that you restored the correct instance of the critical and noncritical storage spaces.

Related information:
The oninit utility

Replacing disks during a restore

You can replace disks during a restore by renaming chunks. You rename chunks by specifying new chunks paths and offsets during a cold restore with ON-Bar. This option is useful if you need to restore storage spaces to a different disk from the one on which the backup was made. You can rename any type of chunk, including critical chunks and mirror chunks.

The old chunk must be included in the last level-0 backup.
The following guidelines apply to new chunks:

- The new chunk does not need to exist. You can install the new chunk later and perform a warm restore of a storage space containing it. If you specify a nonexistent chunk, ON-Bar records the rename information in the chunk reserved pages, but does not restore the data. The renamed (but not restored) chunks have a status of offline, designated by an N flag in the output of the **onstat -d** command.
- The new chunk must have the correct permissions.
- The new chunk must be included in the last level-0 backup.
- The new chunk path name and offset cannot overlap existing chunks.

Tip: If you use symbolic links to chunk names, you might not need to rename chunks; you only need to edit the symbolic name definitions.
To rename chunks during a restore:

1. Shut down the database server.
2. Run the **onbar -r** command with the **-rename** option and the chunk information options. If you are renaming the primary root or mirror root chunk, ON-Bar updates the values of the ROOTPATH and ROOTOFFSET, or MIRRORPATH, and MIRROROFFSET configuration parameters. The old version of the onconfig file is saved as $ONCONFIG.*localtime*.
3. Perform a level-0 archive so that you can restore the renamed chunks.

Examples

The following table lists example values for two chunks that are used in the examples in this section.

Element	Value for first chunk	Value for second chunk
Old path	/chunk1	/chunk2
Old offset	0	10000
New path	/chunk1N	/chunk2N
New offset	20000	0

Example 1: Rename chunks by supplying chunk information in the command
The following command renames the chunks **chunk1** to **chunk1N** and **chunk2** to **chunk2N**:

```
onbar -r -rename -p /chunk1 -o 0 -n /chunk1N -o 20000
         -rename -p /chunk2 -o 10000 -n /chunk2N -o 0
```

Example 2: Rename chunks by supplying chunk information in a file

Suppose that you have a file named listfile that has the following contents:

```
/chunk1 0 /chunk1N 20000
/chunk2 10000 /chunk2N 0
```

The following command renames the chunks **chunk1** to **chunk1N** and **chunk2** to **chunk2N**:

```
onbar -r -rename -f listfile
```

- Renaming a chunk to a nonexistent device
  To rename a chunk to a nonexistent device, specify the new path name, but restore the storage spaces after you install the physical device. This option is useful if you need to rename a chunk and it is convenient to perform a cold restore before you install the new device. When the new chunk device is ready, you can perform a warm restore of a storage space onto it.

Related reference:
onbar -r syntax: Restoring data
onbar -b syntax: Backing up

Renaming a chunk to a nonexistent device

To rename a chunk to a nonexistent device, specify the new path name, but restore the storage spaces after you install the physical device. This option is useful if you need to rename a chunk and it is convenient to perform a cold restore before you install the new device. When the new chunk device is ready, you can perform a warm restore of a storage space onto it.

You can combine renaming chunks with existing devices and renaming chunks with nonexistent devices in the same rename operation. This example shows how to rename a single chunk to a nonexistent device name.

The following table lists example values for the chunks used in this example.

Storage space	Old chunk path	Old offset	New chunk path	New offset
sbspace1	/chunk3	0	/chunk3N	0

To rename a chunk to a nonexistent device:

1. Rename the chunk with the following command: **onbar -r -rename -p /chunk3 -o 0 -n /chunk3N -o 0**
2. When you see the following prompt, enter `y` to continue:

   ```
   The chunk /chunk3N does not exist. If you continue, the
   restore may fail later for the dbspace which contains this chunk.
   Continue without creating this chunk? (y/n)
   ```

   The chunk /chunk3 is renamed to /chunk3N, but the data is not yet restored to /chunk3N.

3. Perform a level-0 archive.
4. Add the physical device for /chunk3N.
5. Perform a warm restore of **sbspace1** with the **onbar -r sbspace1** command.
6. Perform a level-0 archive.

Restoring to a different computer

You can back up data on one computer and restore the data on a different computer. Importing a restore is useful for disaster recovery or upgrading a database server. After you back up your data and move over the storage-manager objects, you can perform an imported restore. An imported restore involves copying files from the source to the target computer and performing the restore in one of several ways.

Prerequisites:

- Your storage manager must support imported restores.
- A whole-system backup must include all storage spaces; logical logs are optional.
  The level-0 backup must include all storage spaces and logical logs.

- Both the source and target computers must be on the same LAN or WAN and must have the following attributes:
  - Identical hardware and operating systems
  - Identical database server versions and editions
  - The same configuration and ROOTPATH information, although the server names and numbers can differ.
  - Identical storage-manager versions
  - Compatible XBSA libraries

Important: Every chunk (including mirrors) must match exactly in size, location, and offset on the source and target computers for the imported restore to complete.
To perform the imported restore:

1. Install the database server and the storage manager on the target computer.
2. Set up the storage manager on the target database server instance.
   a. Set the necessary environment variables.
   b. Define the same type of storage devices as on the source instance.
   c. Label the storage media with the correct pool names.
   d. Mount the storage devices.
   e. Update the sm_versions file on the target computer with the storage-manager version.

3. Be sure that the target computer has the devices and links in place for the chunks that match the devices and links on the source computer
4. Perform a level-0 backup (**onbar -b** or **onbar -b -w**) of all storage spaces on the source database server.
   Restriction: Do not perform an incremental backup.
5. Mount the transferred storage volumes.
   - If the backup files are on disk, copy them from the source computer to the target computer.
   - If the backup is on tapes, mount the transferred volumes on the storage devices that are attached to the target computer. Both the source and target computers must use the same type of storage devices such as 8-mm tape or disk.
   - If the backup is on the backup server, retrieve the backup from that backup server.
6. Use storage-manager commands to add the source host name as a client on the target computer.
7. Copy the following files from the source computer to the target computer.

Table 1. Administrative files to copy

File	Action
Emergency boot file	Rename the emergency boot file with the target database server number. For example, rename ixbar.51 to ixbar.52. The emergency boot file needs only the entries from the level-0 backup on the source computer. The file name is ixbar.*servernum*.
The oncfg files: oncfg_*servername.servernum*	ON-Bar needs the oncfg file to know what dbspaces to retrieve. Rename the oncfg file with the target database server name and number. For example, rename oncfg_bostonserver.51 to oncfg_chicagoserver.52. The file name must match the DBSERVERNAME and SERVERNUM on the target computer.
The onconfig file	In the onconfig file, update the DBSERVERNAME and SERVERNUM parameters with the target database server name and number.
Storage-manager configuration files, if any	The storage-manager configuration files might need updating.

8. Restore the data in one of the following ways:

Table 2. Restore data options

Option	Action
If you did not start the instance on the target server	Use the **onbar -r** command to restore the data.
If you are importing a whole-system backup	Use the **onbar -r -w -p** command to restore the data.
If you started the instance on the target server.	Restore the data in stages: a. Use the **onbar -r -p** command to restore the physical data. b. Use the **onbar -r -l** command to restore the logical logs. This process avoids salvaging the logs and any potential corruption of the instance.

9. Before you expire objects on the target computer and the storage manager with the **onsmsync** utility, perform one of the following tasks. Otherwise, **onsmsync** expires the incorrect objects.
   - Manually edit the emergency boot file viz ixbar.*servernum* in the $INFORMIXDIR/etc directory on the target computer. Replace the IBM® Informix® server name that is used on the source computer with the IBM Informix server name of the target computer
   - Run the **onsmsync -b** command as user **informix** on the target computer to regenerate the emergency boot file from the **sysutils** database only. The regenerated emergency boot file reflects the server name of the target computer.

Related reference:
Avoid salvaging logical logs

onbar -RESTART syntax: Restarting a failed restore

If a failure occurs with the database server, media, storage manager, or ON-Bar during a restore, you can restart the restore from the place that it failed. To restart a failed restore, the RESTARTABLE_RESTORE configuration parameter must be set to ON in the onconfig file when the restore fails.

```
Restart a restore

>>-onbar-- -RESTART-------------------------------------------><
```

Table 1. onbar -RESTART command

Option	Description
-RESTART	Restarts a restore after a database server, storage manager, or ON-Bar failure. The RESTARTABLE_RESTORE configuration parameter must be set to ON when the restore failure occurs. You can restart the following types of restores: • Whole system • Point in time • Storage spaces • Logical part of a cold restore Do not use the **-RESTART** option if a failure occurs during a warm logical restore.

Usage

When you enable restartable restore, the logical restore is slower if many logical logs are restored. However, you save time if the restore fails and you restart the restore. Whether a restore is restartable does not affect the speed of the physical restore.

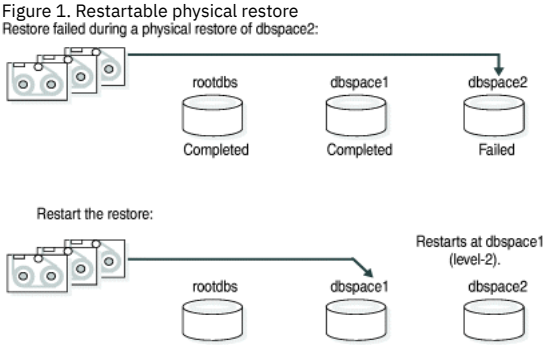
The physical restore restarts at the storage space and level where the failure occurred. If the restore failed while some, but not all, chunks of a storage space were restored, all chunks of that storage space are restored. If storage spaces and incremental backups are restored successfully before the failure, they are not restored again.

If the BAR_RETRY configuration parameter is set to 2, ON-Bar automatically tries to restore any failed storage spaces and logical logs again. If the restore is successful, you do not need to restart the restore.

If the BAR_RETRY configuration parameter is set to 0 or 1, ON-Bar does not try to restore any failed storage spaces and logical logs again. If the database server is still running, ON-Bar skips the failed storage space and attempts to restore the remaining storage spaces. To complete the restore, run the **onbar -RESTART** command.

The following figure shows how a restartable restore works when the restore failed during a physical restore of **dbspace2**. The level-0, level-1, and level-2 backups of **rootdbs**, and the level-0 and level-1 backups of **dbspace1** and **dbspace2** are successfully restored. The database server fails while restoring the level-1 backup of **dbspace2**. When you restart the restore, ON-Bar restores the level-2 backup of **dbspace 1**, the level-1 and level-2 backups of **dbspace2**, and the logical logs.

Figure 1. Restartable physical restore

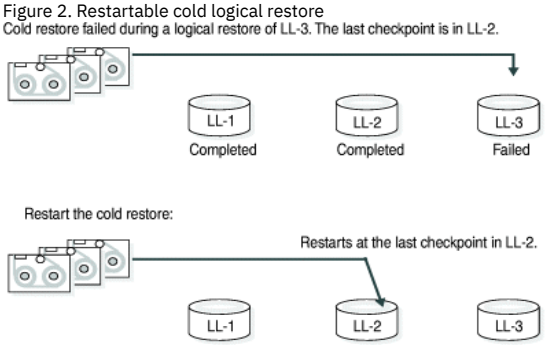


If a restore fails during the logical phase and you restart the restore, ON-Bar verifies that the storage spaces are restored, skips the physical restore, and restarts the logical restore. The following figure shows a cold restore that failed while restoring logical log LL-3. When you restart the cold logical restore, log replay starts from the last restored checkpoint. In this example, the last checkpoint is in logical log LL-2.

If a failure occurs during a cold logical restore, ON-Bar restarts the restore at the place of failure.

Important: If a failure occurs during a warm logical restore, restart the restore from the beginning. If the database server is still running, run the **onbar -r -l** command to complete the restore.

Figure 2. Restartable cold logical restore



Related reference:
onbar -r syntax: Restoring data
BAR_RETRY configuration parameter
RESTARTABLE_RESTORE configuration parameter
ON-Bar security

Resolve a failed restore

How you resolve a failed restore depends on the cause of the failure.

You can save some failed restores even if restartable restore is turned off. For example, if the restore fails because of a storage-manager or storage-device error, you can fix the tape drive or storage-manager problem, remount a tape, and then continue the restore.

The following table shows what results to expect when physical restore fails and the value of the BAR_RETRY configuration parameter is > 1.

Table 1. Failed physical restore scenarios

Type of error	RESTARTABLE_ RESTORE setting	What to do when the physical restore fails?
Database server, ON-Bar, or storage-manager error (database server is still running)	ON or OFF	ON-Bar tries each failed restore again. If the storage manager failed, fix the storage-manager error. If the tried restore fails, issue **onbar -r** *spaces* where *spaces* is the list of storage spaces not yet restored. Use **onstat -d** to obtain the list of storage spaces that need to be restored. ON-Bar restores the level-0 backup of each storage space, then the level-1 and level-2 backups, if any.
ON-Bar or storage-manager error (database server is still running)	ON	Issue the **onbar -RESTART** command. If the storage manager failed, fix the storage-manager error. The restore restarts at the storage space and backup level where the first restore failed. If the level-0 backup of a storage space was successfully restored, the restarted restore skips the level-0 backup and restores the level-1 and level-2 backups, if any.

Type of error	RESTARTABLE_ RESTORE setting	What to do when the physical restore fails?
Database server failure	ON or OFF	Because the database server is down, perform a cold restore. Use **onbar -r** to restore the critical dbspaces and any noncritical spaces that were not restored the first time.
Database server failure	ON	Issue the **onbar -RESTART** command. The restore restarts at the storage space and backup level where the first restore failed. If the level-0 backup of a storage space was successfully restored, the restarted restore skips the level-0 backup and restores the level-1 and level-2 backups, if any.

The following table shows what results to expect when logical restore fails.

Table 2. Failed logical restore scenarios

Type of error	RESTARTABLE_ RESTORE setting	What to do when a logical restore fails?
Database server or ON-Bar error in a cold restore (database server is still running)	ON	Issue the **onbar -RESTART** command. The logical restore restarts at the last checkpoint. If this restore fails, shut down and restart the database server to initiate fast recovery of the logical logs. All logical logs not restored are lost.
Database server or ON-Bar error (database server is still running)	ON or OFF	Issue the **onbar -r -l** command. The restore restarts at the failed logical log. If **onbar -r -l** still fails, shut down and restart the database server. The database server completes a fast recovery. All logical logs that were not restored are lost. If fast recovery does not work, you must do a cold restore.
Database server error	ON	If the cold logical restore failed, issue **onbar -RESTART**. If the warm logical restore failed, issue the **onbar -r -l** command. If that fails, restart the entire restore from the beginning.
Storage-manager error	ON or OFF	ON-Bar tries each failed logical restore again. If the tried restore fails, the logical restore is suspended. Fix the storage-manager error. Then issue the **onbar -r -l** command. The restore restarts at the failed logical log.

Related reference:
BAR_RETRY configuration parameter
RESTARTABLE_RESTORE configuration parameter

External backup and restore

These topics discuss recovering data by using external backup and restore.

- External backup and restore overview
  An external backup and restore eliminates the downtime of systems because the backup and restore operations are performed external to the IBM® Informix® system.
- RS secondary server external backup
  You can perform an external backup of an RS secondary server. Performing a backup of an RS secondary server blocks that RS secondary server, but does not block the primary server.
- Data restored in an external restore
- Performing an external restore
- Initializing HDR with an external backup and restore

External backup and restore overview

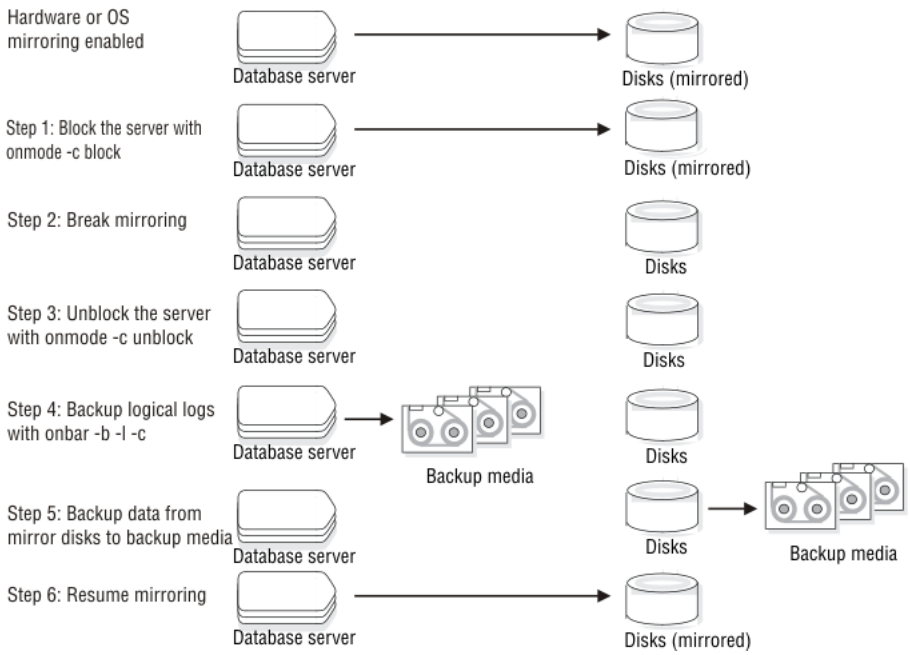
An external backup and restore eliminates the downtime of systems because the backup and restore operations are performed external to the IBM® Informix® system.

ON-Bar does not move the data during the backup or physical restore. An external backup allows you to copy disks that contain storage-space chunks without using ON-Bar. When disks fail, replace them and use vendor software to restore the data, then use ON-Bar for the logical restore. For more information, see Data restored in an external restore.

The following are typical scenarios for external backup and restore:

- Availability with disk mirroring
  If you use hardware disk mirroring, you can get your system online faster with external backup and restore than with conventional ON-Bar commands.
- Cloning
  You can use external backup and restore to clone an existing production system for testing or migration without disturbing the production system.

The following figure shows how to perform a backup with mirroring.
Figure 1. Perform a backup with mirroring



In this configuration, the database server is running continuously, except for the short time when the database server is blocked to break the mirror. The mirrored disks contain a copy of the database server storage spaces. To create a backup, block the database server to stop transactions and disable mirroring. The mirrored disks now contain a copy of the consistent data at a specific point in time. After disabling mirroring, unblock the database server to allow transactions to resume and then backup the logical logs. Copy the data from the offline mirrored disks to back up media with external commands. Now you can resume mirroring.

- Block before backing up
  Before you begin an external backup, block the database server. Blocking forces a checkpoint, flushes buffers to disk, and blocks user transactions that involve temporary tables.
- Rules for an external backup
- Prepare for an external backup
- Performing an external backup when chunks are not mirrored

Block before backing up

Before you begin an external backup, block the database server. Blocking forces a checkpoint, flushes buffers to disk, and blocks user transactions that involve temporary tables.

During the blocking operation, users can access that database server in read-only mode. Then you can physically back up or copy the data to another set of disks or storage media by using operating-system or third-party tools. When you complete the external backup, unblock the database server so that transactions can resume. You should include all the chunk files in each storage space, administrative files, such as onconfig, and the emergency boot file, in an external backup.
Important: To make tracking backups easier, you should back up all storage spaces in each external backup.
ON-Bar treats an external backup as equivalent to a level-0 backup. You cannot perform an external backup and then use ON-Bar to perform a level-1 backup, or vice versa because ON-Bar does not have any record of the external backup. For more information, see Performing an external backup when chunks are not mirrored.

Rules for an external backup

Before you begin an external backup, review the rules for performing an external backup.

The rules that you must follow are:

- The database server must be online or in quiescent mode during an external backup.
- Use ON-Bar to back up all logical logs including the current log so that you can restore the logical logs at the end of the external restore.
- Suspend continuous logical-log backups before you block the database server for an external backup. After the external backup is complete, resume the continuous logical-log backup.
  To stop continuous logical-log backup, use the **CTRL-C** command. To resume continuous logical-log backup, use the **onbar -b -l -C** command.
- Wait until all ON-Bar backup sessions have completed before you block the database server. If any backup sessions are active, the block command displays an error message.
- Any OLTP work or queries are suspended while the database server is blocked. They resume after the database server is unblocked.
- All critical dbspaces of the database server instance must be backed up together simultaneously within the same command bracket of **onmode -c block ... onmode -c unblock**. Backups of different critical dbspaces done at different times cannot be restored to a consistent system.
- On AIX® operating systems, if the server is running with concurrent I/O because the DIRECT_IO configuration parameter is set to enable concurrent I/O, an online external backup program must also use concurrent I/O.

Important: Because the external backup is outside the control of ON-Bar, you must track these backups manually. For more information, see Track an external backup.

Prepare for an external backup

These topics describe the commands used to prepare for an external backup. For the procedure, see Performing an external backup when chunks are not mirrored.

- Block and unblock database server
- Track an external backup

Block and unblock database server

This topic shows the syntax of the block and unblock commands on IBM® Informix®.

```
>>-onmode -c--+---------+--------------------------------------><
              +-block---+
              '-unblock-'
```

Element	Purpose	Key considerations
onmode -c	Performs a checkpoint and blocks or unblocks the database server	None.
block	Blocks the database server from any transactions	Sets up the database server for an external backup. While the database server is blocked, users can access it in read-only mode. Sample command: **onmode -c block**
unblock	Unblocks the database server, allowing data transactions and normal database server operations to resume	Do not unblock until the external backup is finished. Sample command: **onmode -c unblock**

Track an external backup

The database server and ON-Bar do not track external backups. To track the external backup data, use a third-party storage manager or track the data manually.

The following table shows which items we recommend that you track in an external backup. ON-Bar keeps a limited history of external restores.

Table 1. Items to track when you use external backup and restore

Items to track	Examples
Full path names of each chunk file for each backed up storage space	/work/dbspaces/rootdbs (UNIX) c:\work\dbspaces\rootdbs (Windows)
Object type	Critical dbspaces, noncritical storage spaces
ins_copyid_hi and **ins_copyid_lo**	Copy ID that the storage manager assigns to each backup object
Backup date and time	The times that the database server was blocked and unblocked
Backup media	Tape volume number or disk path name
Database server version	The database server version from which the backup was taken.

Performing an external backup when chunks are not mirrored

The database server must be online or in quiescent mode during an external backup.

To perform an external backup when chunks are not mirrored:

1. To obtain an external backup, block the database server with the **onmode -c block** command. The system takes a checkpoint and suspends all update transactions. Users can access the database server in read-only mode.
2. To back up the storage spaces and administrative files, use a copy command, such as **cp**, **dd**, or **tar** on UNIX or **copy** on Windows, or a file-backup program. You must back up all chunks in the storage spaces.
3. To allow normal operations to resume, unblock the database server with the **onmode -c unblock** command.
4. Back up all the logical logs including the current log so that checkpoint information is available for the external restore.
   Important: Because external backup is not done through ON-Bar, you must ensure that you have a backup of the current logical log from the time when you execute the **onmode -c block** command. Without a backup of this logical-log file, the external backup is not restorable.
5. After you perform an external backup, back up the current log with the **onbar -b -l -c** command.

If you lose a disk, or the whole system, you are now ready to perform an external restore.

RS secondary server external backup

You can perform an external backup of an RS secondary server. Performing a backup of an RS secondary server blocks that RS secondary server, but does not block the primary server.

You can perform a logical restore from the logs backed up from the primary instance. The backup obtained from the secondary server cannot be restored with level-1 or level-2 backups.

Important: The external backup is not completed if the database instance contains any of the following:

- Nonlogging smart large objects
- Regular blobspaces
- Nonlogging databases
- Raw tables

If an external backup is performed on an instance that contains any of the previously mentioned items, then the backup is incomplete and cannot be used to restore the primary server.
If the backup fails because the checkpoint from the primary has timed out, you can use the BAR_CKPTSEC_TIMEOUT configuration parameter to increase the amount of time, in seconds, that an RS secondary server should wait for a checkpoint to arrive from the primary server while performing an external backup.

- Performing an external backup of an RS secondary server

Performing an external backup of an RS secondary server

To perform an external backup of an RS secondary server, the STOP_APPLY configuration parameter must not be enabled. If STOP_APPLY is enabled, an error is returned. The server switches to STOP_APPLY mode when a backup is performed on an RS secondary. After the archive checkpoint is processed, the RS secondary server stops applying logical logs, but continues receiving logs from the primary server.

To perform an external backup of an RS secondary server that has a DELAY_APPLY configuration parameter value greater than 0, it might be necessary to temporarily decrease the parameter value. Performing the backup requires that the RSS process a checkpoint in the logical log, and if no checkpoint is observed within the amount of time that is specified by the **onmode -c block** *timeout* command in the second step of the following procedure, a backup is not permitted. The DELAY_APPLY configuration parameter can be decreased by the **onmode -wf DELAY_APPLY=***setting* command.

The primary database server must be online or in quiescent mode during an external backup.

To perform an external backup:

1. Ensure that the LOG_STAGING_DIR configuration parameter on the RS secondary server is set to point to a valid staging directory.
2. To obtain an external backup, block the database server with the **onmode -c block** *timeout* command. The *timeout* parameter indicates the number of seconds that the RS secondary server waits to receive a checkpoint. The *timeout* parameter is valid only when the **onmode -c block** command is run on an RS secondary server. You must wait for the **onmode -c block** command to return successfully before you proceed with the external backup.
3. To back up the storage spaces and administrative files, use a copy command, such as **cp**, **dd**, or **tar** on UNIX or **copy** on Windows, or a file-backup program. You must back up all chunks in the storage spaces.
4. To resume normal operations, unblock the database server by using the **onmode -c unblock** command.
5. After you perform the external backup, back up the current log and any new logs with the ON-Bar or **ontape** utilities.
   Important: Logical log backup is only possible on the primary server.
   If the DELAY_APPLY configuration parameter is set, the logs that are required for the restore process are not necessarily those logs that are currently active on the primary server because some logs could already be archived.

After the backup completes, if the DELAY_APPLY setting on the RS secondary server was decreased, it can be set to its original value by the **onmode -wf DELAY_APPLY=***setting* command. After an external backup, you can perform an external restore if a disk or the whole system fails.

Data restored in an external restore

If you lose a disk, or the whole system, you can externally restore data only if it was externally backed up. You must use the same third-party utility for both the external backup and restore. To externally restore the storage spaces, copy the backed-up data to disk. Use the **onbar -r -e** command to mark the storage spaces as physically restored, replay the logical logs, and bring the storage spaces back online. If you do not specify an external restore command, the database server thinks that these storage spaces are still down.

You can perform these types of external restores:

- Warm external restore
  Mark noncritical storage spaces as physically restored, then perform a logical restore of these storage spaces.
- Cold external restore
  Mark storage spaces as physically restored, then perform a logical restore of all storage spaces. Optionally, you can do a point-in-time cold external restore.

Restriction: When you perform a cold external restore, ON-Bar does not first attempt to salvage logical-log files from the database server because the external backup has already copied over the logical-log data.
To salvage logical logs, perform **onbar -l -s** before you copy the external backup and perform the external restore (**onbar -r -e**).

- Rename chunks
- External restore commands
- Rules for an external restore
  External restores have specific rules.

Rename chunks

You can rename chunks in an external cold restore by using the rename options syntax for other restores. Use the following commands to specify new chunk names during restore:

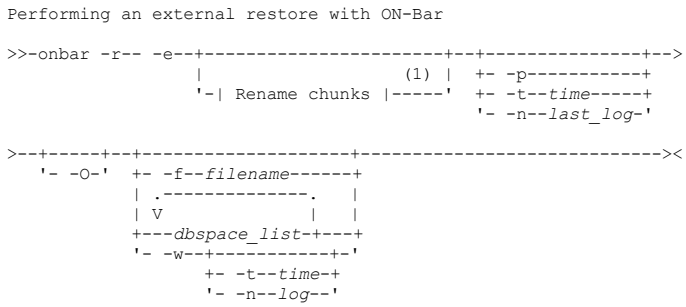
```
onbar -r -e -rename -f filename
```

or

```
onbar -r -e rename -p old_path -o old_offset-n new_path-o new_offset
```

External restore commands

Use the **onbar -r -e** command to perform a warm or cold external restore. This command marks the storage spaces as physically restored and restores the logical logs. The following diagram shows the external restore syntax.



Notes:

1. See onbar -b syntax: Backing up

Element	Purpose	Key considerations
onbar -r	Specifies a restore	In a cold restore, if you do not specify storage space names, all of them are marked as restored.
-e	Specifies an external restore	Must be used with the **-r** option. In a warm external restore, marks the down storage spaces as restored unless the **-O** option is specified.
dbspace_list	Names one or more storage spaces to be marked as restored in a warm restore	If you do not enter *dbspace_list* or **-f *filename*** and the database server is online or quiescent, ON-Bar marks only the down storage spaces as restored. If you enter more than one storage-space name, use a space to separate the names.
-f*filename*	Restores the storage spaces that are listed in the text file whose path name *filename* provides	To avoid entering a long list of storage spaces every time, use this option. The *filename* can be any valid UNIX or Windows file name.
-n *last_log*	Indicates the number of the last log to restore	If any logical logs exist after this one, ON-Bar does not restore them and data is lost. The -n option does not work with the -p option.
-O	Restores online storage spaces	None.
-p	Specifies an external physical restore only	After the physical restore completes, you must perform a logical restore.
-t *time*	Restores the last backup before the specified point in time. If you pick a backup made after the point in time, the restore will fail.	You can use a point-in-time restore in a cold restore only. You must restore all storage spaces. How you enter the time depends on your current GLS locale convention. If the GLS locale is not set, use English-style date format.
-w	Performs a whole-system restore of all storage spaces and logical logs from the last whole-system backup	You must specify the -w option in a cold restore. If you specify **onbar -r -w** without a whole-system backup, return code 147 appears because ON-Bar cannot find any storage spaces backed up as part of a whole-system backup.

Related reference:
onbar -r syntax: Restoring data

Rules for an external restore

External restores have specific rules.

The following rules apply to external restores:

- You must externally restore from an external backup. Although the external backup is treated as a level-0 backup, it might actually be an incremental backup not related to IBM® Informix®.
- A warm external restore restores only noncritical storage spaces.
- You cannot externally restore temporary dbspaces.
- You cannot externally restore from regular ON-Bar backups.
- You cannot verify that you are restoring from the correct backup and that the storage media is readable with ON-Bar.
- If the external backups are from different times, the external restore uses the beginning logical log from the oldest backup.
- You cannot perform a mixed restore. If critical dbspaces must be restored, you must perform a full cold restore.

The following rules apply to external cold restores:

- Salvage the logical logs (**onbar -b -l -s**) before you switch the disks that contain the critical storage spaces.
- If you are restoring critical dbspaces, the database server must be offline.
- Point-in-time external restores must be cold and restore all storage spaces.
- The external backups of all critical dbspaces of the database server instance must be simultaneous. All critical dbspaces must have to be backed up within the same set of **onmode -c block ... onmode -c unblock** commands.

Performing an external restore

This section describes procedures for performing cold and warm external restores.

- Performing a cold external restore
- Performing a warm external restore
- Examples of external restore commands

Performing a cold external restore

If you specify the **onbar -r -e** command in a cold restore, you must restore all storage spaces. Use the **onbar -r -e -p** command to restore all or specific storage spaces.

To perform a cold external restore:

1. Shut down the database server with the **onmode -ky** command.
2. Salvage the logical logs with the **onbar -b -l -s** command.
3. To restore the storage spaces from an external backup, use a copy command, such as **cp**, **dd**, or **tar** on UNIX or a file-backup program.
   You must restore the storage spaces to the same path as the original data and include all the chunk files.
4. To perform an external restore of all storage spaces and logical logs, use the **onbar -r -e** command.
5. To perform a point-in-time external restore of all storage spaces, use the **onbar -r -e -t** *datetime* command.
   This step brings the database server to fast-recovery mode.

   ON-Bar and the database server roll forward the logical logs and bring the storage spaces online.

Performing a warm external restore

The database server is online during a warm external restore. A warm external restore involves only noncritical storage spaces.

To perform a warm external restore:

1. To restore the storage spaces from an external backup, use a copy command, such as **cp**, **dd**, or **tar** on UNIX or a file-backup program.
   You must restore the storage spaces to the same path as the original data and include all the chunk files for each restored storage space.
2. Perform a warm external restore of the noncritical storage spaces to bring them online.
   - To restore selected storage spaces and all logical logs, use the **onbar -r -e** *dbspace_list* command.
   - To restore the down noncritical storage space named dbsp1 and logical logs in separate steps, use the following commands:
     - **onbar -r -e -p dbsp1**
     - **onbar -r -l dbsp1**
   - To restore all the noncritical storage spaces and logical logs, use the **onbar -r -e -O** command.

Examples of external restore commands

The following table contains examples of external restore commands.

External restore command	Action	Comments
onbar -r -e	Complete external restore	In a cold restore, restores everything. In a warm restore, restores all down noncritical storage spaces.
onbar -r -e -p **onbar -r -l**	Physical external restore and separate logical restore	If the external backups come from different times, you must perform a logical restore. The system restores the logical logs from the oldest external backup.
onbar -r -e *dbspace_list*	External restore of selected storage spaces and logical logs	Use this command in a warm external restore only.
onbar -r -e -p *dbspace_list* **onbar -r -l**	External restore of selected storage spaces and separate logical restore	Use this command in a warm external restore only.
onbar -r -e -t *datetime*	External point-in-time (cold) restore	Be sure to select a collection of backups from before the specified time.
onbar -r -e rename -p ***old_path*** **-o** ***old_offset*****-n** ***new_path*** **-o** ***new_offset***	External (cold) restore with renamed chunks	Use this command to rename chunks in cold external restore only.
onbar -r -e -w **onbar -r -e -p -w**	Whole-system external restore	When you use **onbar -r -e -w -p**, back up all storage spaces in one block and unblock session. That way, all storage spaces have the same checkpoint.

Initializing HDR with an external backup and restore

You can use external backups to initialize High-Availability Data Replication (HDR).

To initialize HDR with an external backup and restore:

1. Block the source database server with the **onmode -c block** command.
2. Externally back up all chunks on the source database server.
3. When the backup completes, unblock the source database server with the **onmode -c unblock** command.
4. Make the source database server the primary server with the following command: **onmode -d primary** *secondary_servername*
5. On the target database server, restore the data from the external backup with a copy or file-backup program.
6. On the target database server, restore the external backup of all chunks with the **onbar -r -e -p** command. On HDR, secondary server can restore only level-0 archives.
7. Make the target database server the secondary server with the following command: **onmode -d secondary** *primary_servername*
8. If the logical-log records written to the primary database server since step 1 still reside on the primary database server disk, the secondary database server reads these records to perform the logical recovery. Otherwise, perform the logical recovery with the **onbar -r -l** command.
   The `database server operational` messages appear in the message log on the primary and secondary servers.

Customize and maintain ON-Bar

These topics discuss the following:

- Customizing ON-Bar and storage-manager commands with the **onbar** script
- Starting **onbar-worker** processes manually
- Expiring and synchronizing the backup history

- Customizing ON-Bar and storage-manager commands
  You can edit the script that is installed with ON-Bar to customize backup and restore commands, and storage manager commands.
- Expire and synchronize the backup catalogs
- Monitor the performance of ON-Bar and the storage managers

Customizing ON-Bar and storage-manager commands

You can edit the script that is installed with ON-Bar to customize backup and restore commands, and storage manager commands.

On UNIX operating systems, the **onbar** shell script is in the $INFORMIXDIR/bin directory. On Windows operating systems, the **onbar.bat** batch script is in the %INFORMIXDIR%\bin directory.

Edit the script and backup a copy of the original file in case you need to revert to it.

Important: Edit the script with caution and test your changes. Do not change the cleanup code at the bottom of the script. Doing so might result in unexpected behavior, for example, leftover temporary files during backup verification.
The script contains the following sections:

- Add startup processing here
  Use this section to initialize the storage manager, if necessary, and set environment variables.

- End startup processing here
  This section starts the **onbar_d** driver and checks the return code. Use this section for **onbar_d** and storage-manager commands.

- Add cleanup processing here
  This section removes the **archecker** temporary files.

- End cleanup processing here
  Use this section to return **onbar_d** error codes.

- Updating the ON-Bar script during reinstallation
  After you reinstall the database server, you might need to update the script that is installed with ON-Bar. Existing customized scripts were backed up by the installation process so that you can reuse the content.
- Print the backup boot files
- Migrate backed-up logical logs to tape

Related reference:
Back up with ON-Bar

Updating the ON-Bar script during reinstallation

After you reinstall the database server, you might need to update the script that is installed with ON-Bar. Existing customized scripts were backed up by the installation process so that you can reuse the content.

The installation program installs the default **onbar** shell script on UNIX and the default **onbar.bat** batch script on Windows. If the default script differs from the local script, the installation program backs up the local script and issues a message to inform you that the local script was renamed. The naming convention of the renamed file is onbar.*date*, where *date* is the date when the file was renamed. For example, the file onbar.2012.05.15 was renamed on May 15, 2012.

You can update the default script by adding information to it from the renamed script.

Print the backup boot files

Use the following examples of what to add to the **onbar** script to print the emergency boot file if the backup is successful. Each time that you issue the **onbar -b** command, the emergency boot file is printed.

The following example is for UNIX:

```
onbar_d "$@"    # receives onbar arguments from command line return_code = $?
# check return code

# if backup (onbar -b) is successful, prints emergency boot file
if [$return_code -eq 0 -a "$1" = "-b"]; then
   servernum=`awk '/^SERVERNUM/ {print $2}' $INFORMIXDIR/etc/$ONCONFIG '
   lpr \$INFORMIXDIR/etc/ixbar.$servernum
fi
exit $return_code
```

The following example is for Windows:

```
@echo off
%INFORMIXDIR%\bin\onbar_d %*
set onbar_d_return=%errorlevel%

if "%onbar_d_return%" == "0" goto backupcom
goto skip

REM Check if this is a backup command

:backupcom
if "%1" == "-b" goto printboot
goto skip

REM Print the onbar boot file

:printboot
print %INFORMIXDIR%\etc\ixbar.???

REM Set the return code from onbar_d (this must be on the last line of the script)

:skip
%INFORMIXDIR%\bin\set_error %onbar_d_return%
:end
```

Migrate backed-up logical logs to tape

You can set up your storage manager to back up logical logs to disk and then write a script to automatically migrate the logical logs from disk to tape for off-site storage. Edit the **onbar** script to call this migration script after the **onbar_d** process completes. The following example shows a script that calls the migration script:

The following example is for UNIX:

```
onbar_d "$@" # starts the backup or restore
EXIT_CODE=$?         # any errors?

PHYS_ONLY=false      #if it's physical-only, do nothing
for OPTION in $*; do
    if [$OPTION = -p]; then
        PHYS_ONLY = true

    fi
done
if ! PHYS_ONLY; then   # if logs were backed up, call another
     migrate_logs      # program to move them to tape
fi
```

This example for Windows invokes the migration script:

```
%INFORMIXDIR%\bin\onbar_d %*
set onbar_d_return=%errorlevel%

if "%onbar_d_return%" == "0" goto backupcom
goto skip

REM Check if the command is a backup command

:backupcom
if "%1" == "-b" goto m_log
if "%1" == "-l" goto m_log
goto skip

REM Invoke the user-defined program to migrate the logs

:m_log
migrate_log

REM Set the return code from onbar_d (this must be on the last line of the script)

:skip
%INFORMIXDIR%\bin\set_error %onbar_d_return%

:end
```

Expire and synchronize the backup catalogs

ON-Bar maintains a history of backup and restore operations in the **sysutils** database and an extra copy of the backup history in the emergency boot file. ON-Bar uses the **sysutils** database in a warm restore when only a portion of the data is lost. ON-Bar uses the emergency boot file in a cold restore because the **sysutils** database cannot be accessed. You can use the **onsmsync** utility to regenerate the emergency boot file and expire old backups.

Depending on the command options you supply, the **onsmsync** utility can remove the following items from the **sysutils** database and the emergency boot file:

- Backups that the storage manager has expired
- Old backups based on the age of backup
- Old backups based on the number of times they have been backed up

Use **onsmsync** with the database server online or in quiescent mode to synchronize both the **sysutils** database and the emergency boot file.

To synchronize the **sysutils** database:

1. Bring the database server online or to quiescent mode.
2. Run the **onsmsync** utility without any options.

The **onsmsync** utility synchronizes the **sysutils** database, the storage manager, and the emergency boot file as follows:

- Adds backup history to **sysutils** that is in the emergency boot file but is missing from the **sysutils** database.
- Removes the records of restores, whole-system restores, fake backups, successful and failed backups from the **sysutils** database.
- Expires old logical logs that are no longer needed.
- Regenerates the emergency boot file from the **sysutils** database.

- Choose an expiration policy
- The onsmsync utility
  Use the **onsmsync** utility to synchronize the **sysutils** database and emergency boot file with the storage manager catalog.

Choose an expiration policy

You can choose from the following three expiration policies:

Retention date (-t)
: Deletes all backups before a particular date and time.

Retention interval (-i)
: Deletes all backups older than a specified period.

Retention generation (-g)
: Keeps a certain number of versions of each backup.

ON-Bar always retains the latest level-0 backup for each storage space. It expires all level-0 backups older than the specified time unless they are required to restore from the oldest retained level-1 backup.

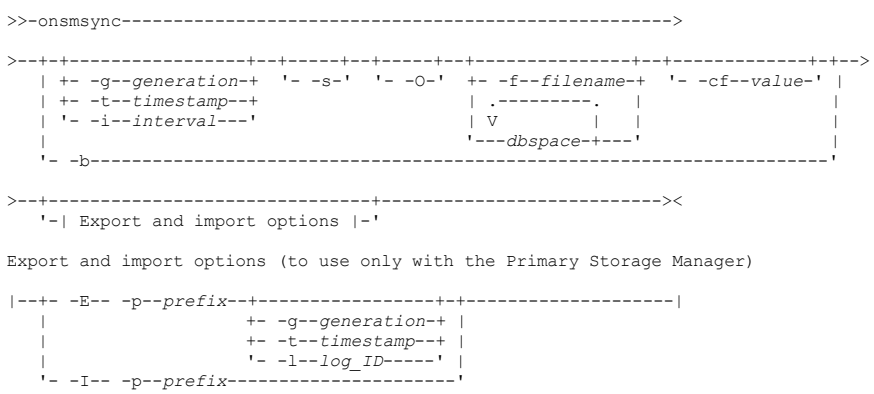
ON-Bar expires all level-1 backups older than the specified time unless they are required to restore from the oldest retained level-2 backup.

ON-Bar retains a whole-system backup that starts before the specified retention time and ends after the specified retention time.

The onsmsync utility

Use the **onsmsync** utility to synchronize the **sysutils** database and emergency boot file with the storage manager catalog.

If your storage manager is the Informix® Primary Storage Manager, you can also use the export and import options of the **onsmsync** utility to export Informix Primary Storage Manager backup objects to an external device and import objects from an external device to a device that is managed by the Informix Primary Storage Manager. You cannot use the export and import options with other storage managers.



The following table lists all **onsmsync** command elements, except the elements you use to import and export backup generations. The command elements that you use for importing and exporting are listed in Table 2.

Table 1. Elements for **onsmsync** commands

Element	Purpose	Key considerations
-b	Regenerates the emergency boot file (ixbar.*servernum*) and the **sysutils** database from each other.	If the ixbar file is empty or does not exist, **onsmsync -b** re-creates the ixbar file and populates it from the **sysutils** tables. If the ixbar file is not empty and contains object data, **onsmsync -b** updates the **sysutils** database and the ixbar file so that they are in sync. If the ixbar file has entries and the **sysutils** database was rebuilt, but is empty because it does not contain data, **onsmsync -b** recreates the **sysutils** data from the ixbar file. Do not use the **-b** element with the other **onsmsync** options. The **-b** element does not synchronize with the storage manager.
dbspace	Specifies the storage space or storage spaces to expire	If you enter the name of more than one storage space, use a blank space to separate the names.
-f *filename*	Specifies the path name of a file that contains a list of storage spaces to expire	Use this option to avoid entering a long list of storage spaces. The file name can be any valid UNIX or Windows file name.
-g *generation*	Specifies the number of versions of each level-0 backup to retain	The latest generation of backups is retained and all earlier ones are expired.
-i *interval*	Specifies the time interval for retaining backups.	The utility: • Retains backups that were created after this interval. • Expires backups that were created before this interval and removes the backups if expired objects are also removed. Backups older than interval are not expired if they are needed to restore from other backups after that interval. Use the ANSI or GLS format for the *interval*: YYYY-MM or DD HH:MM:SS
-O	Overrides internal error checks and enforces expiration policy	If used with the **-t**, **-g**, or **-i** option, expires all levels of a backup, even if some of them are needed to restore from a backup that occurred after the expiration date. The **-O** option does not affect logical-log expiration. See Expire all backups.
-s	Skips synchronizing expired backups	The object expiration is based on other arguments if the **-s** option is provided.
-t *timestamp*	Expires all backups before a particular date and time	Retains backups that completed after the specified timestamp. Backups that occurred before the timestamp are not expired if they are needed to restore from other backups that occurred after the timestamp. Use the ANSI or GLS_DATETIME format for the timestamp.
-cf	Specifies whether to expire the critical files. When used with the **-g**, **-i**, or **-t**, deletes critical file backups from the Informix Primary Storage Manager	The critical files are the onconfig file, the sqlhosts file (on UNIX), the oncfg_*servername.servernum* file, and the ixbar.*servernum* file. Valid values are: • **yes** = Deletes backups of the critical files. • **no** = Does not delete backups of the critical files. • **only** = Deletes only the backups of critical files.

Table 2. Elements for **onsmsync** export and import commands

Element	Purpose	Key considerations
-E	Exports a single backup generation to the Informix Primary Storage Manager external pool	Use this option only if you set up a Informix Primary Storage Manager external pool. If you export a backup generation, you must specify a prefix, which identifies the exported backup. The **onsmsync** utility creates a subdirectory that includes the prefix in the external pool, and deposits the exported objects in that directory.
-g *generation*	Specifies the backup generation to export.	The default value is the current backup.
-I	Imports a single backup generation from the external Informix Primary Storage Manager pool.	If you import a backup generation from an external pool, you must specify the prefix for exported backup. The prefix identifies the backup generation that you want to import.
-l *log_ID*	Exports the backup generation that includes a logical log ID.	
-p *prefix*	Specifies the prefix that you are assigning to a backup generation that you are exporting or are using to identify the backup that you want to import.	When the **onsmsync** utility exports a backup generation, it uses the prefix as the name of the subdirectory in which it places the backup.
-t *timestamp*	Specifies a backup generation that includes a particular date and time (used only for exporting).	Use the ANSI or GLS_DATETIME format. You might want to include the date and time when you roll out a new version of an application.

Usage

If no options are specified, the **onsmsync** command synchronizes the **sysutils** database and emergency boot file with the storage-manager catalog. The **onsmsync** utility compares the backups in the **sysutils** database and emergency boot file with the backups in the storage-manager catalog and then removes all backups that are not in the storage manager catalog from the **sysutils** database and emergency boot file.

Tip: To control whether the **sysutils** database maintains a history for expired backups and restores, use the BAR_HISTORY configuration parameter. For information, see BAR_HISTORY configuration parameter.
The order of the commands does not matter except:

- The storage-space names or file name must come last.
- When exporting or importing, the **-E** or **-I** option must be first. For example, specify **onsmsync -E -g 2**, not **onsmsync -g 2 -E**.

Prerequisites for importing and exporting backup generations on different computers:

- You must have the same version of Informix on the source and target computers and the computers must use the same operating system.
- You must set up the Informix Primary Storage Manager and create an external pool on the source and target computers.

When you use the **-E** or **-I** options to export or import a backup generation, you must specify the prefix that identifies the subdirectory where the backup generation is placed.

If you use the **-E** or **-I** options to export or import a backup generation, you cannot use any **onsmsync** command options that are not related to the export or import operation. For example, you cannot export a backup generation and regenerate the emergency boot file at the same time.

The **onsmsync -I** command renames your current ixbar file and creates a new file hat contains only the information that is necessary to restore the imported backup

You can use the **-cf** option with the **-g**, **-i**, or **-t** option to delete critical file backups from the storage manager.

If you apply the **-g** option and the **onsmsync** utility list of objects contains only logical logs and no space backups, the log backups are not expired. In this situation, use the **-t** or **-i** option to expire logical log backups.

Examples

The following example expires backups that started before November 30, 2012:

```
onsmsync -t "2012-11-30 00:00:00""
```

The following command exports the last backup generation to the directory call `gen` in the external pool:

```
onsmsync -E -p gen -g 1
```

The following command exports the fourth most recent backup generation to the directory called `gen` in the external pool:

```
onsmsync -E -p gen -g 4
```

The following command exports the current backup generation to the directory called `gen` in the external pool:

```
onsmsync -E -p gen
```

The following command exports all backup objects in generation 2 to the directory called `gen` in the external pool:

```
onsmsync -E -p gen -g 2
```

The following command exports all backup objects that have a timestamp of `2012-12-31 12:00:00` to the directory called `gen` in the external pool:

```
onsmsync -E -p gen -t "2012-12-31 12:00:00"
```

The following command imports all objects in the subdirectory that is identified by the prefix `gen`:

```
onsmsync -I -p gen
```

The following command imports all backup objects that were exporting using the prefix `gen` and the timestamp of `2012-12-31 12:00:00`. Because the prefix identifies the backup generation, you do not specify a timestamp.

```
onsmsync -I -p gen
```

The following command deletes all but the last two generations of critical file backups:

```
onsmsync -g 2 -cf yes
```

- Regenerate the emergency boot file
- Regenerate the sysutils database
- Delete a bad backup
- Expire backups based on the retention date
- Expire a generation of backups
- Expire backups based on the retention interval
- Expire backups with multiple point-in-time restores
- Expire all backups

Related concepts:
Managing storage devices
Informix Primary Storage Manager file-naming conventions
Related tasks:
Examples: Manage storage devices with Informix Primary Storage Manager
Restoring when a backup is missing data
Performing a cold restore
Related reference:
ON-Bar script

Regenerate the emergency boot file

To regenerate the emergency boot file only, use the **onsmsync -b** command.

The **onsmsync -b** command saves the old emergency boot file as ixbar.*server_number.system_time* and regenerates it as ixbar.*server_number*.

Regenerate the sysutils database

If you lose the **sysutils** database, use the **bldutil** utility in $INFORMIXDIR/etc on UNIX or %INFORMIXDIR%\etc on Windows to recreate the **sysutils** database with empty tables.

Then use the **onsmsync** utility to recreate the backup and restore data in **sysutils**.
Restriction: If both the **sysutils** database and emergency boot file are missing, you cannot regenerate them with **onsmsync**. Be sure to back up the emergency boot file with your other operating-system files.

Delete a bad backup

The **onsmsync** utility cannot tell which backups failed verification. If the latest backup failed verification but an earlier one was successful, you must manually delete the failed backup records from the storage manager and then run **onsmsync** with no options to synchronize ON-Bar. For more information, see onbar -v syntax: Verifying backups.

Expire backups based on the retention date

The following example expires backups that started before November 24, 2006, and all fake backups, failed backups, and restores:

```
onsmsync -t "2006-11-24 00:00:00"
```

Expire a generation of backups

The following example retains the latest three sets of level-0 backups and the associated incremental backups, and expires all earlier backups and all restores, fake backups, and failed backups: **onsmsync -g 3**

Expire backups based on the retention interval

The following example expires all backups that are older than three days and all fake backups, failed backups, and restores:

```
onsmsync -i "3 00:00:00"
```

The following example expires all backups older than 18 months (written as 1 year + 6 months):

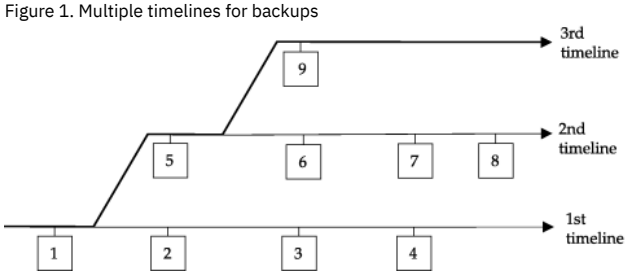
```
onsmsync -i "1-6"
```

Expire backups with multiple point-in-time restores

If you perform more than one point-in-time restores, multiple timelines for backups exist.

The following figure shows three timelines with their backups.

Figure 1. Multiple timelines for backups



In this example, the second timeline begins with a point-in-time restore to backup 1. The second timeline consists of backups 1, 5, 6, 7, and 8. The third timeline (in bold) consists of backups 1, 5, and 9. The third timeline is considered the current timeline because it contains the latest backup.

When you run the **onsmsync** utility to expire old backups, **onsmsync** removes the old backups from the current timeline, and make sure that the current timeline is restorable from the backup objects that are retained. All other backups that are not in the current timeline are also expired but **onsmsync** does not make sure that the other timelines are restorable from the objects retained.

The **onsmsync** utility applies expiration policies in the following order to make sure that objects from current timeline are expired according to the specified expiration policy and that the current timeline is restorable:

- Apply the expiration policy on all sets of backup objects.
- Unexpire backup objects that belong to the current timeline.
- Apply the expiration policy on the current timeline to ensure that the current timeline is restorable.

At the same time, the expiration policy is applied to backups in other timelines.

For example, if you execute the **onsmsync -g 2** command on the example in the previous figure, backup 1 from the current timeline is expired, and backups 2, 3, 4, 6, and 7 from the first and second timelines are expired. Backups 1, 5, and 9 from the current timeline are retained. Backup 8 is retained from other timelines.

Expire all backups

The **onsmsync** utility retains the latest level-0 backup unless you use the -O option. If you use the -O and -t options, all backups from before the specified time are removed even if they are needed for restore. If you use the -O and -i options, all backups from before the specified interval are removed even if they are needed for restore.

For example, to expire all backups, specify the following:

```
onsmsync -O -g 0
```

Important: If you use the -O option with the -t, -i, or -g options, you might accidentally delete critical backups, making restores impossible.

Monitor the performance of ON-Bar and the storage managers

You can monitor the performance of ON-Bar and your storage manager. You can specify the level of performance monitoring and have the statistics print to the ON-Bar activity log. The BAR_PERFORMANCE configuration parameter specifies whether to gather statistics. The following statistics are gathered:

- Total time spent in XBSA calls.
- Total time spent in Archive API calls.
- Time spent by ON-Bar in transferring data to and from XBSA (storage manager calls).
- Time spent by ON-Bar in transferring data between ON-Bar to IBM® Informix®.
- Amount of data transferred to or from the XBSA API.
- Amount of data transferred to or from the Archive API.

- Set ON-Bar performance statistics levels
  To specify the level of performance statistics that are printed to the ON-Bar activity log, set the BAR_PERFORMANCE configuration parameter in the onconfig file.
- View ON-Bar backup and restore performance statistics
  To view ON-Bar performance results, open the ON-Bar activity log file, bar_act.log.

Set ON-Bar performance statistics levels

To specify the level of performance statistics that are printed to the ON-Bar activity log, set the BAR_PERFORMANCE configuration parameter in the onconfig file.

For example, the `BAR_PERFORMANCE 1` setting displays the time spent transferring data between the IBM® Informix® instance and the storage manager.

See BAR_PERFORMANCE configuration parameter for information about the options for this parameter.

View ON-Bar backup and restore performance statistics

To view ON-Bar performance results, open the ON-Bar activity log file, bar_act.log.

The location of the bar_act.log file is set by the BAR_ACT_LOG configuration parameter.

When the BAR_PERFORMANCE configuration parameter is set to 1 or 3, the activity report shows a transfer rate report.
Figure 1. Sample transfer rate performance in the ON-Bar activity log

```
 2013-06-03 15:38:02 8597  8595 Begin restore logical log 310 (Storage Manager
                                copy ID: 28206 0).
 2013-06-03 15:38:03 8597  8595 Completed restore logical log 310.
 2013-06-03 15:38:08 8597  8595 Completed logical restore.
 2013-06-03 15:38:19 8597  8595 PERFORMANCE INFORMATION

                                                                TRANSFER RATES
-------------------------------------------------------------------------------------------------------------------------------
|       OBJECT      |                   XBSA API                          |                   SERVER API                          |
|        NAME       | xfer-kbytes   xfer-time RATIO(kb/s)    API-TIME | xfer-kbytes   xfer-time RATIO(kb/s)    API-TIME |
-------------------------------------------------------------------------------------------------------------------------------
| 309               |          62       0.479         129       1.078 |          62       0.019        3310       0.310 |
| 310               |          62       0.407         152       1.098 |          62       0.025        2522       0.025 |
| rootdbs           |        5828       0.618        9436       1.864 |        5828       8.922         653       8.931 |
| datadbs01         |          62       0.488         127       1.768 |          62       0.004       17174       0.004 |
| datadbs02         |          62       0.306         203       1.568 |          62       0.008        8106       0.008 |
| datadbs03         |          62       0.304         204       1.574 |          62       0.007        8843       0.007 |
| datadbs04         |          62       0.306         202       1.563 |          62       0.007        8664       0.007 |
| datadbs05         |          62       0.315         197       1.585 |          62       0.007        8513       0.007 |
| datadbs06         |          62       0.310         200       1.583 |          62       0.002       25348       0.002 |
----------------- ---------------------------------- ... -------------------------------------------------------------------
```

```
| PID =       8597 |       14722      26.758         550     107.476 |       14756      10.678        1382      15.829 |
-------------------------------------------------------------------------------------------------------------------------
 2013-06-03 15:38:19 8597  8595 PERFORMANCE INFORMATION

                                                    PERFORMANCE CLOCKS
-------------------------------------------------------------------------------------------------------------------------
|                        ITEM DESCIRPTION                                 |                  TIME SPENT                    |
-------------------------------------------------------------------------------------------------------------------------
| Time to Analyze ixbar file                                             |                                          0.000 |
-------------------------------- ----------------------------------------------------------------------------------------
```

When the BAR_PERFORMANCE configuration parameter is set to 2 or 3, the activity report has microsecond timestamps.
Figure 2. Sample processing rates, in microseconds, in the ON-Bar activity log

```
2013-06-03 16:34:04 15272  15270 /usr/informix/bin/onbar_d complete,
                                 returning 0 (0x00)
2013-06-03 16:45:11.608424 17085  17083 /usr/informix/bin/onbar_d -r -w
2013-06-03 16:46:07.926097 17085  17083 Successfully connected to Storage Manager.
2013-06-03 16:46:08.590675 17085  17083 Begin salvage for log 311.
2013-06-03 16:48:07.817487 17085  17083 Completed salvage of logical log 311.
2013-06-03 16:48:08.790782 17085  17083 Begin salvage for log 312.
2013-06-03 16:48:10.129534 17085  17083 Completed salvage of logical log 312.
2013-06-03 17:06:00.836390 17085  17083 Successfully connected to Storage Manager.
                                   ...
2013-06-03 17:07:26.357521 17085  17083 Completed cold level 0 restore datadbs07.
2013-06-03 17:07:28.268562 17085  17083 Begin cold level 0 restore datadbs08
  (Storage Manager copy ID: 28122 0).
2013-06-03 17:07:29.378405 17085  17083 Completed cold level 0 restore datadbs08.
```

Related concepts:
bar_act.log file: ON-Bar activity log
Related reference:
BAR_ACT_LOG configuration parameter
BAR_PERFORMANCE configuration parameter

ON-Bar catalog tables

These topics describe the ON-Bar tables that are stored in the **sysutils** database.

ON-Bar uses these tables for tracking backups and performing restores.

You can query these tables for backup and restore data to evaluate performance or identify object instances for a restore.

- The bar_action table
  The **bar_action** table lists all backup and restore actions that are attempted against an object, except during certain types of cold restores. Use the information in this table to track backup and restore history.
- The bar_instance table
  The **bar_instance** table contains descriptions of each object that is backed up.
- The bar_ixbar table
  The **bar_ixbar** table, which stores a history of all unexpired successful backups in all timelines, is maintained and used by the **onsmsync** utility only.
- The bar_object table
  The **bar_object** table contains descriptions of each backup object. This table is a list of all storage spaces and logical logs from each database server for which at least one backup attempt was made.
- The bar_server table
  The **bar_server** table lists the database servers in an installation. This table is used to ensure that backup objects are returned to their proper places during a restore.
- The bar_syncdeltab table
  The **bar_syncdeltab** table is maintained and used by the **onsmsync** utility only. This table is empty except when **onsmsync** is running.
- ON-Bar catalog map
  This topic contains an example mapping between ON-Bar tables.

The bar_action table

The **bar_action** table lists all backup and restore actions that are attempted against an object, except during certain types of cold restores. Use the information in this table to track backup and restore history.

Table 1. **bar_action** table columns

Column name	Type	Explanation
act_aid	SERIAL	Action identifier. A unique number within the table. Can be used with act_oid column to join with the **bar_instance** table.
act_oid	INTEGER	Object identifier. Identifies the backup object against which a backup or restore attempt is made. Can be used with act_aid to join with **bar_instance**. The act_oid column of the **bar_action** table equals the obj_oid column of the **bar_object** table.
act_type	SMALLINT	Identifies the attempted action: 1 for backup, 2 for restore, 3 for a foreign or imported restore, 4 for a fake backup, 5 for a whole-system backup, 6 for a whole-system restore, 7 for expired or deleted objects, 8 for an external restore.

Column name	Type	Explanation
act_status	INTEGER	Identifies the result of the action: 0 if successful, otherwise an ON-Bar-specific error code. For more information, see ON-Bar messages and return codes.
act_start	DATETIME YEAR TO SECONDS	The date and time when the action began.
act_end	DATETIME YEAR TO SECONDS	The date and time when the action finished.

The bar_instance table

The **bar_instance** table contains descriptions of each object that is backed up.

ON-Bar writes a record to the **bar_instance** table for each successful backup. ON-Bar might later use the information for a restore operation. For example, if you specify a level-2 backup, ON-Bar uses this table to ensure that a level-1 backup was done previously.

Table 1. **bar_instance** table columns

Column name	Type	Explanation
ins_aid	INTEGER	Action identifier. Identifies the successful action that created this instance of the backup object. Combined with **ins_oid**, can be used to join with the **bar_action** table.
ins_oid	INTEGER	Object identifier. Identifies the affected object. Can be used to join with the **bar_object** table. Combined with **ins_aid**, can be used to join with the **bar_action** table.
ins_time	INTEGER	Timestamp (real clock time). The database server uses this value when it creates the next-level backup. Value represents the number of seconds since midnight, January 1, 1970, Greenwich mean time. The **ins_time** value is 0.
rsam_time	INTEGER	The backup checkpoint time stamp. Not a clock time. The database server uses this value when it creates the next level backup.
seal_time	INTEGER	The time that the log file was sealed after a backup completed.
prev_seal_time	INTEGER	The time that the previous log file was sealed.
ins_level	SMALLINT	Level of the backup action: 0 for a complete backup, 1 for a backup of any changes to this object since its last level-0 backup, 2 for a backup of any changes since the last level-1 backup. This value is always 0 for logical-log backups.
ins_copyid_hi	INTEGER	The high bits of the instance copy identifier. Combined with **ins_copyid_lo**, it is a unique value that the storage manager assigns to link the ON-Bar object identifier with the storage-manager object identifier.
ins_copyid_lo	INTEGER	The low bits of the instance copy identifier. Combined with **ins_copyid_hi**, it is a unique value that the storage manager assigns to link the ON-Bar object identifier with the storage-manager object identifier.
ins_req_aid	INTEGER	Stores the required action ID for a backup object. Used in a restore to determine which level-0 backup goes with the level-1 backup, and which level-1 backup goes with the level-2 backup. For a level-0 backup, the value of **ins_req_aid** is the same as **ins_aid** in this table. For example, if this backup is level-1, **ins_req_aid** holds the action ID of the corresponding level-0 backup of this object.
ins_logstream	INTEGER	Not used.
ins_first_log	INTEGER	In a standard backup, identifies the first logical log required to restore from this backup.
ins_chpt_log	INTEGER	The logical log that contains the archive checkpoint in the dbspace backup.
ins_last_log	INTEGER	In a standard backup, identifies the last logical log required to restore from this backup.
ins_partial	INTEGER	Partial flag from salvage.
ins_sm_id	INTEGER	Not used.
ins_sm_name	CHAR(128)	Not used.
ins_verify	INTEGER	Value is 1 if the backup is verified. Value is 0 if the backup is not verified.
ins_verify_date	DATETIME YEAR TO SECOND	The current date is inserted when a backup is verified. If this backup has not been not verified, a dash represents each date and time.
ins_backup_order	INTEGER	The order in which backups occur.

The bar_ixbar table

The **bar_ixbar** table, which stores a history of all unexpired successful backups in all timelines, is maintained and used by the **onsmsync** utility only.

The schema of the **bar_ixbar** table is identical to the schema of the **bar_syncdeltab** table, except for its primary key.

Table 1. **bar_ixbar** table columns

Column name	Type	Explanation
ixb_sm_id	INTEGER	Storage-manager instance ID. Created from BAR_SM in $ONCONFIG or %ONCONFIG%.

Column name	Type	Explanation
ixb_copyid_hi	INTEGER	The high bits of the instance copy identifier. Combined with **ixb_copyid_lo**, it is a unique value that the storage manager assigns to link the ON-Bar object identifier with the storage-manager object identifier.
ixb_copyid_lo	INTEGER	The low bits of the instance copy identifier. Combined with **ixb_copyid_hi**, it is a unique value that the storage manager assigns to link the ON-Bar object identifier with the storage-manager object identifier.
ixb_aid	INTEGER	Action identifier, Identifies the successful action that created this instance of the backup object.
ixb_old	INTEGER	Object identifier. Identifies the affected object.
ixb_time	INTEGER	Time stamp (real clock time). The database server uses this value when it creates the next-level backup. Value represents the number of seconds since midnight, January 1, 1970, Greenwich mean time.
ixb_prevtime	INTEGER	Time stamp (real clock time). This value specifies the time stamp of the previous object. Value represents the number of seconds since midnight, January 1, 1970 Greenwich mean time.
ixb_rsam_time	INTEGER	The backup checkpoint time stamp. Not a clock time. The database server uses this value when it creates the next level backup.
ixb_act_start	datetime year to second	The date and time when the action began.
ixb_act_end	datetime year to second	The date and time when the action finished.
ixb_level	SMALLINT	Level of the backup action: 0 for a complete backup, 1 for a backup of any changes to this object since its last level-0 backup, 2 for a backup of any changes since the last level-1 backup. This value is always 0 for logical-log backups.
ixb_req_aid	INTEGER	Stores the required action ID for a backup object. Used in a restore to determine which level-0 backup Goes with the level-1 backup, and which level-1 backup goes with the level-2 backup. For a level-0 backup, the value of **ixb_req_aid** is the same as **ixb_aid** in this table. For example, if this backup is level-1, **ixb_req_aid** holds the action ID of the corresponding level-0 backup of this object.
ixb_first_log	INTEGER	In a standard backup, identifies the first logical log. Required to restore from this backup.
ixb_chpt_log	INTEGER	The ID of the log that contains the **rsam_time** checkpoint. Used during back up to verify that logs needed for restore are backed up.
ixb_last_log	INTEGER	Log ID of the last log needed during logical restore for this storage space to restore it to the time of the backup.
ixb_lbuflags	INTEGER	Flags describing log backup.
ixb_verify	INTEGER	Value is 1 if the backup is verified. Value is 0 if the backup is not verified.
ixb_verify_date	datetime year to second	The current date is inserted when a backup is verified. If this backup has not been verified, a dash represents each date and time.
ixb_sm_name	VARCHAR(128)	Storage-manager instance name. Created from the BAR_SM_NAME parameter in the onconfig file.
ixb_srv_name	VARCHAR(128)	The database server name. Used to ensure that objects are restored to the correct database server. Used when multiple database servers are on the node to ensure that objects are restored in the database server instance to which the object belongs. The database server name can be up to 128 characters.
ixb_obj_name	VARCHAR(128)	The user name for the object. The name can be up to 128 characters.
ixb_obj_type	CHAR(2)	Backup object type: CD critical dbspace L logical log ND noncritical dbspace or sbspace R rootdbs B blobspace

Related reference:
The bar_syncdeltab table

The bar_object table

The **bar_object** table contains descriptions of each backup object. This table is a list of all storage spaces and logical logs from each database server for which at least one backup attempt was made.

Table 1. **bar_object** table columns

Column name	Type	Explanation

Column name	Type	Explanation
obj_srv_name	VARCHAR(128,0)	The database server name. Used to ensure that objects are restored to the correct database server. Used when multiple database servers are on the node to ensure that objects are restored in the database server instance to which the object belongs. The database server name can be up to 128 characters.
obj_oid	SERIAL	The object identifier. A unique number within the table. Can be used to join with the **bar_action** and **bar_instance** tables.
obj_name	VARCHAR(128,0)	The user name for the object. The name can be up to 128 characters.
obj_type	CHAR(2)	Backup object type: CD critical dbspace L logical log ND noncritical dbspace or sbspace R rootdbs B blobspace

The bar_server table

The **bar_server** table lists the database servers in an installation. This table is used to ensure that backup objects are returned to their proper places during a restore.

Table 1. **bar_server** table columns

Column name	Type	Explanation
srv_name	VARCHAR(128,0)	DBSERVERNAME value specified in the onconfig file. Database server name can be up to 128 characters.
srv_node	CHAR(256)	Host name of the computer where the database server resides. The host name can be up to 256 characters.
srv_synctime	INTEGER	The time **onsmsync** was run.

The bar_syncdeltab table

The **bar_syncdeltab** table is maintained and used by the **onsmsync** utility only. This table is empty except when **onsmsync** is running.

The schema of the **bar_syncdeltab** table is identical to the schema of the **bar_ixbar** table, except for its primary key.

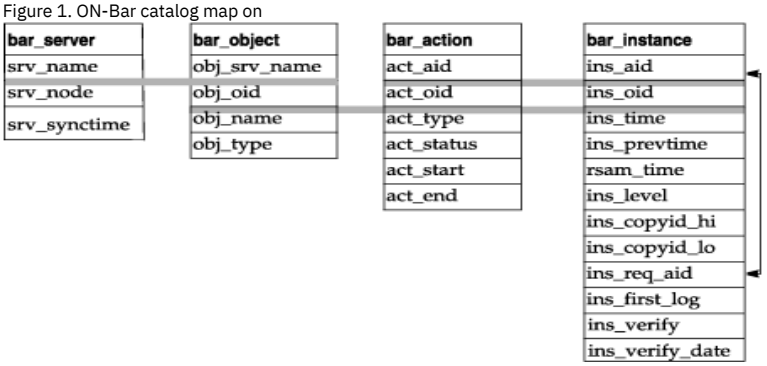
Related reference:
The bar_ixbar table

ON-Bar catalog map

This topic contains an example mapping between ON-Bar tables.

The following figure maps the ON-Bar tables on IBM® Informix®. In this figure, the gray lines show the relations between tables. The arrows show that the **ins_req_aid** value must be a valid **ins_aid** value.

Figure 1. ON-Bar catalog map on



ON-Bar messages and return codes

ON-Bar prints informational, progress, warning, and error messages to the ON-Bar activity log file. ON-Bar return codes indicate the status of the command.

For a description of an error message, use the **finderr** utility or go to Error messages.

- Message format in the ON-Bar message log
  Messages in the ON-Bar activity log file contain timestamps, process IDs, and explanatory text.
- Message numbers
  The ON-Bar message numbers range from -43000 to -43421.
- ON-Bar return codes
  You can troubleshoot problems by viewing activity log messages that are applicable for particular return codes.

Related concepts:
bar_act.log file: ON-Bar activity log
Related reference:
onbar -m syntax: Monitoring recent ON-Bar activity

Message format in the ON-Bar message log

Messages in the ON-Bar activity log file contain timestamps, process IDs, and explanatory text.

A message in the ON-Bar activity log file, bar_act.log, has the following format:

timestamp process_id parent_process_id message

The following table describes each field in the message. No error message numbers appear in the ON-Bar activity log.

Table 1. ON-Bar message format

Message field	Description
timestamp	Date and time when ON-Bar writes the message.
process_id	The number that the operating system uses to identify this instance of ON-Bar.
parent_process_id	The number that the operating system uses to identify the process that executed this instance of ON-Bar.
message	The ON-Bar message text.

The following example illustrates a typical entry in the ON-Bar activity log:

```
1999-08-18 10:09:59 773     772  Completed logical restore.
```

Important: If you receive an XBSA error message, consult the storage-manager logs for more details.

Timestamps when storage managers hang

If a storage manager process hangs, the timestamp for the process in the ON-Bar activity log is inaccurate. An asterisk symbol is appended to the timestamp and the message indicates how long the storage manager process hung. The following example shows that the storage manager process hung for two minutes starting at 10:27. At 10:29, the storage manager completed the backup.

```
2013-02-26 10:27:10* 13410  25695 (-43085) WARNING: BAR_TIMEOUT Storage
                        Manager Progress may be stalled for at least 2 minutes.

2013-02-26 10:29:12 13410  25695 Completed level 0 backup dbspace1 (Storage Manager
                                 copy ID: 1509564809 0).
```

Related reference:
onbar -m syntax: Monitoring recent ON-Bar activity

Message numbers

The ON-Bar message numbers range from -43000 to -43421.

The following table lists the ON-Bar message groups. Because message numbers do not display in the activity log, the best way to find information about ON-Bar messages is to search for the message text in the error messages file, which is in the subdirectory for your locale under the $INFORMIXDIR/msg directory.

ON-Bar message type	Message numbers
ON-Bar usage	-43000 to -43007 and -43357
Options checking	-43010 to -43034
Permission checking	-43035 to -43039
Emergency boot file interface	-43040 to -43059
onconfig file interface	-43060 to -43074
Operating system interface	-43075 to -43099

ON-Bar message type	Message numbers
Database server interface	-43100 to -43229
Back up and restore status	-43230 to -43239
onbar-worker processes	-43240 to -43254
XBSA interface	-43255 to -43301
onsmsync	-43302 to -43319
archecker	-43320 to -43334
ondblog	-43400 to -43424

ON-Bar return codes

You can troubleshoot problems by viewing activity log messages that are applicable for particular return codes.

The following table shows the ON-Bar return codes for all IBM® Informix® database servers. These return codes are accompanied by messages in the ON-Bar activity log. For details about an error, review the activity log before you call Technical Support.

Table 1. Common ON-Bar return codes

Decimal value	ON-Bar return code description
2 through 34	These return codes are produced by XBSA. For more information, consult your storage-manager documentation and log files.
100	ON-Bar cannot find something in **sysutils**, the emergency boot file, or storage-manager catalogs that it needs for processing. Check the ON-Bar activity log for messages that say what could not be found and try to resolve that problem. If the problem recurs, contact Technical Support.
104	Adstar Distributed Storage Manager (ADSM) is in generate-password mode. ON-Bar does not support ADSM running in generate-password mode. For information about changing the ADSM security configuration, refer to your ADSM manual.
115	A critical dbspace is not in the set of dbspaces being cold-restored.
116	The **onsmsync** utility is already running.
117	The information contained in the **sysutils** database and the emergency boot file are inconsistent.
118	An error in tenant restore occurred since another tenant restore was in progress.
119	An error in tenant restore occurred since another restore was in progress.
120	An internal On-Bar error occurred when starting a tenant restore.
121	ON-Bar was unable to determine the list of dbspaces.
122	Deadlock detected. The ON-Bar command is contending with another process. Try the ON-Bar command again.
123	The root dbspace was not in the cold restore. You cannot perform a cold restore without restoring the root dbspace. To resolve the problem, try one of the following procedures: • Bring the database server to quiescent or online mode and restore just the storage spaces that need to be restored. • If the database server is offline, issue the **onbar -r** command to restore all the storage spaces. • Make sure that the root dbspace and other critical dbspaces are listed on the command line or in the **-f *filename***.
124	The buffer had an incomplete page during the backup. For assistance, contact Technical Support.
126	Error processing the emergency boot file. Check the ON-Bar activity log for descriptions of the problem and the emergency boot file for corruption such as non-ASCII characters or lines with varying numbers of columns. If the source of the problem is not obvious, contact Technical Support.
127	Could not write to the emergency boot file. Often, an operating-system error message accompanies this problem. Check the permissions on the following files and directories: • $INFORMIXDIR/etc on UNIX or %INFORMIXDIR%\etc on Windows • The emergency boot file
128	Data is missing in the object description. For assistance, contact Technical Support.
129	ON-Bar received a different object for restore than it had expected. (The backup object did not match.) The requested backup object might have been deleted or expired from the storage manager. Run **onsmsync** to synchronize the **sysutils** database, emergency boot file, and storage-manager catalogs. For assistance, contact Technical Support.
130	Database server is not responding. The database server probably failed during the backup or restore. Run the **onstat -** command to check the database server status and then: • If the operation was a cold restore, restart it. • If the operation was a backup or warm restore, restart the database server and try the backup or warm restore again.

Decimal value	ON-Bar return code description
131	A failure occurred in the interface between ON-Bar and the database server. For assistance, contact Technical Support.
132	Function is not in the XBSA shared library. Verify that you are using the correct XBSA for the storage manager. For information, consult your storage-manager manual.
133	Failed to load the XBSA library functions. Verify that you are using the correct XBSA for the storage manager. Ensure that the BAR_BSALIB_PATH value in the onconfig file points to the correct location of the XBSA shared library. For information, consult your storage-manager manual.
134	User wants to restore a logical-log file that is too early. You probably tried a point-in-log restore (**onbar -r -l -n**) after performing a separate physical restore. The specified logical log is too old to match the backups used in the physical restore. Perform either of the following steps: • Rerun the physical restore from an older set of physical backups. • Specify a later logical log in the -n option when you rerun the point-in-log restore. To find the earliest logical log that you can use, look at the emergency boot file. For assistance, contact Technical Support.
136	ON-Bar cannot warm restore the critical dbspaces. Perform either of the following steps: • Reissue the warm-restore command without listing any critical dbspaces. • Shut down the database server and perform a cold restore.
137	The MAX_DBSPACE_COUNT was exceeded. For assistance, contact Technical Support.
138	An XBSA error occurred. Verify that you are using the correct XBSA for the storage manager. Also check the bar_act.log for XBSA error messages. For information, consult your storage-manager manual.
139	Either the XBSA version is missing from the sm_versions file or the incorrect XBSA version is in the sm_versions file. Insert the correct XBSA version into the sm_versions file. For more information, consult your storage-manager manual.
140	A fake backup failed. Try the fake backup with the **onbar -b -F** command again. Only IBM Informix supports fake backups. If the fake backup fails again, contact Technical Support.
141	ON-Bar received an operating-system signal. Most likely, the user entered the **Ctrl-C** command to stop an ON-Bar process. Fix the cause of the interruption and then try the ON-Bar command again.
142	ON-Bar was unable to open a file. Verify that the named file exists and that the permissions are correct. Check the ON-Bar activity log for an operating-system error message.
143	ON-Bar was unable to create a child process. If BAR_MAX_BACKUP is not 0, ON-Bar could not create child processes to perform the parallel backup or restore. The operating system probably ran out of resources. Either not enough memory is available to start a new process or no empty slot exists in the process table. Check the operating-system logs, the ON-Bar activity log, or the console.
144	The log backup was stopped because one or more blobspaces were down. Attempt to restore the blobspace. If the restore fails, try the log backup with the **onbar -l -O** command again. Executing this command might make the blobspace unrestorable.
145	ON-Bar was unable to acquire more memory space. Wait for system resources to free up and try the ON-Bar command again.
146	ON-Bar was unable to connect to the database server. The network or the database server might be down. For assistance, contact Technical Support.
147	ON-Bar was unable to discover any storage spaces or logical logs to back up or restore. For example, if you specify a point-in-time restore but use a *datetime* value from before the first standard backup, ON-Bar cannot build a list of storage spaces to restore. This return code also displays if you specify a whole-system restore without having performed a whole-system backup. Verify that the database server and the storage spaces are in the correct state for the backup or restore request. Contact Technical Support.
148	An internal SQL error occurred. Provide Technical Support with the information from the ON-Bar activity log.
149	Either you entered the wrong ON-Bar syntax on the command line or entered an invalid or incorrect *datetime* value for your GLS environment. Check the command that you tried against the usage message in the ON-Bar activity log. If that does not help, then try the command with quotes around the *datetime* value again. If your database locale is not English, use the GL_DATE or GL_DATETIME environment variables to set the date and time format.
150	Error collecting data from the onconfig file. Check the permissions, format, and values in the onconfig file. Check that the ONCONFIG environment variable is set correctly.

Decimal value	ON-Bar return code description
151	The database server is in an incorrect state for this backup or restore request, or an error occurred while determining the database server state. Either you attempted an operation that is not compatible with the database server mode or ON-Bar is unable to determine the database server state. For example, you cannot do a physical backup with the database server in recovery mode. Check the error message in the ON-Bar activity log. If an ASF error occurred, the following message displays in the ON-Bar activity log: **`Fatal error initializing ASF; asfcode = code`** To determine the cause of the ASF error, refer to the ASF error code in this message and repeat the backup or restore command. If an ASF error did not occur, change the database server state and repeat the backup or restore command.
152	ON-Bar cannot back up the logical logs. The logical logs are not backed up for either of the following reasons: • If another log backup is currently running. • If you perform a logical-log backup with the LTAPEDEV parameter set to /dev/null (UNIX) or NUL (Windows). You receive this return code when no log backups can be done. To enable log backups, change the LTAPEDEV parameter to a valid value.
153	ON-Bar cannot set the process group ID. If BAR_MAX_BACKUP is set to any value other than 1 and ON-Bar encounters an error setting the process group ID, this value is returned. This message is a warning of a possible operating-system problem.
154	The ON-Bar user does not have the correct permissions. You must be user **root** or **informix** or a member of the **bargroup** group on UNIX or a member of the **Informix-Admin** group on Windows to execute ON-Bar commands.
155	The INFORMIXSERVER environment variable is not set. Set the INFORMIXSERVER environment variable to the correct database server name.
156	Backup or restore was not performed because the LTAPEDEV parameter value is not valid. If LTAPEDEV is not set or /dev/null on UNIX, or if it is NUL on Windows, the logical logs are not backed up, and ON-Bar returns warning 152.
157	Error attempting to set the INFORMIXSHMBASE environment variable to -1. ON-Bar could not set INFORMIXSHMBASE to -1. For assistance, contact either the system administrator or Technical Support.
158	An internal ON-Bar error occurred. Contact Technical Support.
159	An unexpected error occurred. Contact Technical Support.
160	External restore failed. To determine what went wrong with the external restore, look at the bar_act.log and the online.log files. Ensure that you already performed the manual part of the external restore before you try the **onbar-r -e** command again to complete the external restore. If that does not work, try the external restore from a different external backup.
161	Restarted restore failed. Verify that RESTARTABLE_RESTORE is set to ON and try the original restore again. For more information, check the ON-Bar activity log and database server message logs.
162	The ON-Bar log file cannot be a symbolic link. Remove the symbolic link or change the onconfig file so that the ON-Bar parameters BAR_DEBUG_LOG or BAR_ACT_LOG point to non-symbolic linked files.
163	The ON-Bar log file must be owned by user **informix**. Change the ownership of the log file to be owned by user **informix** or change the BAR_ACT_LOG or BAR_DEBUG_LOG values in the onconfig file to point to different log files.
164	Unable to open file. The file or its directory permissions prevent it from being created or opened. Verify the permissions on the file and its directory.
177	An online dbspace was restored. This return code notifies the user that the -O option overrode the internal checks in ON-Bar. You do not need to take any action.
178	The logical log was backed up while one or more blobspaces were down. This return code notifies the user that the -O option overrode the internal checks in ON-Bar. Examine the data in the blobspace to determine which simple large objects you need to recreate. These blobspaces might not be restorable. For assistance, contact Technical Support.
179	ON-Bar created the chunk needed to restore the dbspace. This return code notifies the user that the -O option overrode the internal checks in ON-Bar. You do not need to take any action.
180	ON-Bar could not create the chunk needed to restore the dbspace. Create the chunk file manually. Try the restore without the -O option again.
181	ON-Bar expired an object that was needed for a backup or restore. The **onsmsync** utility expired an object that might be needed for a restore. You probably specified **onsmsync** with the -O option. If you used the -O option by mistake, contact Technical Support to recover the object from the storage manager.
183	ON-Bar could not obtain the logical-log unique ID from the storage manager. The backup of the specified logical log is missing. Query your storage manager to determine if the backup of the specified logical-log file exists and if it is restorable.

Decimal value	ON-Bar return code description
247	On UNIX, look in /tmp/bar_act.log and the file that the BAR_ACT_LOG parameter points to for clues. (The **onbar-merger** writes to /tmp/bar_act.log until it has enough information to read the onconfig file.) Resolve the problems that the bar_act.log describes and try the cold restore again. If the cold restore still fails, contact Technical Support.
252	For assistance, contact Technical Support.

ontape backup and restore system

- Configure ontape
  You configure the **ontape** utility by setting configuration parameters for backups of storage spaces and logical logs.
- Back up with ontape
- Restore with ontape
- Perform an external backup and restore
  These topics discuss performing an external backup and recovering data by restoring it with the **ontape** utility.

Related reference:
LTAPEBLK configuration parameter
LTAPEDEV configuration parameter
LTAPESIZE configuration parameter
TAPEBLK configuration parameter
TAPEDEV configuration parameter
TAPESIZE configuration parameter

Configure ontape

You configure the **ontape** utility by setting configuration parameters for backups of storage spaces and logical logs.

You can also set the IFX_BAR_USE_DEDUP environment variable to optimize the backup images for deduplication devices.

- Set configuration parameters for the ontape utility
  The **ontape** utility uses eight configuration parameters in the onconfig file. Two of the configuration parameters specify filter programs for transforming data during backup and restore; the other six are used to create storage-space and logical-log backups.
- Changing your ontape configuration
  You can change the values of the TAPEDEV, TAPEBLK, and TAPESIZE or LTAPEDEV, LTAPEBLK, and LTAPESIZE configuration parameters after performing a backup and reviewing the current and new parameter values.

Related reference:
IFX_BAR_USE_DEDUP environment variable
Comparison of the ON-Bar and ontape utilities

Set configuration parameters for the ontape utility

The **ontape** utility uses eight configuration parameters in the onconfig file. Two of the configuration parameters specify filter programs for transforming data during backup and restore; the other six are used to create storage-space and logical-log backups.

The onconfig file is located in the $INFORMIXDIR/etc directory. You specify that file in the ONCONFIG environment variable. For a description of the ONCONFIG environment variable and instructions on how to set it, see the *IBM® Informix® Guide to SQL: Reference*.

- Data transformation filter parameters for ontape
  The BACKUP_FILTER and RESTORE_FILTER configuration parameters specify the names of external programs that you can use to transform data before backup and after a restore.
- Tape and tape device parameters for ontape
  The first set of configuration parameters specifies the characteristics of the tape device and tapes for storage-space backups; the second set specifies the characteristics of the tape device and tapes for logical-log backups.
- Set the tape-device parameters
- Specify the tape-block-size
  Use the TAPEBLK and LTAPEBLK configuration parameters to set the largest block size, in kilobytes, that your tape device permits.
- Specify the tape size
  Use the TAPESIZE and LTAPESIZE configuration parameter parameters to specify the maximum amount of data that you can write to a tape.

Data transformation filter parameters for ontape

The BACKUP_FILTER and RESTORE_FILTER configuration parameters specify the names of external programs that you can use to transform data before backup and after a restore.

BACKUP_FILTER
: Specifies the location and name of an external filter program used in data transformation. This filter transforms data before backing it up, such as compressing it. The transformed data is then backed up and stored as a single file. The filter path points to the $INFORMIXDIR/bin directory by default, or an absolute path of the program

RESTORE_FILTER
: Specifies the location and name of an external filter program used in data transformation. This filter transforms data back to its original state before the backup, such as extracting it, before returning the data to the server. The filter path points to the $INFORMIXDIR/bin directory by default, or an absolute path of the program
Prerequisite: The data must have previously been transformed with the BACKUP_FILTER parameter.

See BACKUP_FILTER configuration parameter and RESTORE_FILTER configuration parameter for syntax and usage information, which is the same for ON-Bar and **ontape**.

Tape and tape device parameters for ontape

The first set of configuration parameters specifies the characteristics of the tape device and tapes for storage-space backups; the second set specifies the characteristics of the tape device and tapes for logical-log backups.

The following list shows backup tape devices and their associated tape parameters.

TAPEDEV
: The absolute path name of the tape device or directory file system that is used for storage-space backups. Specify the destination where **ontape** writes storage space data during an archive and the source from which **ontape** reads data during a restore.
To configure **ontape** to use **stdio**, set TAPEDEV to STDIO.

  When backing up to or restoring from a cloud environment, use the following syntax for the TAPEDEV configuration parameter:

  ```
  TAPEDEV 'local_path, keep=option, cloud=cloud_vendor, url=url'
  ```

  - local_path is the complete path name of the directory where storage spaces backup objects are stored temporarily.
  - option can be set to *yes* or *no*. If keep is set to *yes*, the ontape utility retains the backup objects in the local directory. If keep is set to *no*, the backup objects are deleted after they are transferred to or from the cloud storage location.
  - cloud_vendor is the name of the cloud storage vendor.
  - url is the cloud storage location where the storage space backup data is stored persistently.

TAPEBLK
: The block size of the tapes used for storage-space backups, in kilobytes.

TAPESIZE
: The size of the tapes used for storage-space backups, in kilobytes.

The following list shows the logical-log tape devices and their associated tape parameters.

LTAPEDEV
: The logical-log tape device or a directory of a file system.
When backing up to or restoring from a cloud environment, use the following syntax for the LTAPEDEV configuration parameter:

  ```
  LTAPEDEV 'local_path, keep=option, cloud=cloud_vendor, url=url'
  ```

  - local_path is the complete path name of the directory where log backup objects are stored temporarily.
  - option can be set to *yes* or *no*. If keep is set to *yes*, the ontape utility retains the backup objects in the local directory. If keep is set to *no*, the backup objects are deleted after they are transferred to or from the cloud storage location.
  - cloud_vendor is the name of the cloud storage vendor.
  - url is the cloud storage location where the log backup data is stored persistently.

LTAPEBLK
: The block size of tapes used for logical-log backups, in kilobytes.

LTAPESIZE
: The size of tapes used for logical-log backups, in kilobytes.

The following topics contain information about how to set the tape-device, tape-block-size, and tape-size parameters for both storage-space and logical-log backups.

Related tasks:
Back up to Amazon Simple Storage Service

Set the tape-device parameters

Specify values for TAPEDEV and LTAPEDEV in the following ways:

- Use separate tape devices, when possible.
- Use symbolic links.
- Specify a directory of a file system.
- For tape devices, specify /dev/null.
- Rewind tape devices.
- Configure parameters to perform backup to a cloud.

The following sections explain each of these points.

- Specify separate devices for storage-space and logical-log backups
- Specify tape devices as symbolic links
- Specify a file system directory
- Specify a remote device
  You can perform a storage-space or logical-log backup across your network to a remote device attached to another host computer on UNIX and Linux platforms.
- Specify /dev/null for a tape device
  A best practice is to not use /dev/null as the device when backing up. However, if you decide that you do not need to recover transactions from the logical log, you

can specify /dev/null as a tape device for logical-log backups
- Set TAPEDEV to stdio
  To configure the **ontape** utility to read from standard input or write to standard output, set the TAPEDEV configuration parameter to **stdio**.
- Rewind tape devices before opening and on closing

Related tasks:
Back up to Amazon Simple Storage Service

Specify separate devices for storage-space and logical-log backups

When backing up to a tape device, specify different devices for the LTAPEDEV and TAPEDEV parameters in the onconfig file. You can schedule these backups independently of each other. You can create a backup on one device at the same time you continuously back up the logical-log files on the other.

If you specify the same device for the LTAPEDEV and TAPEDEV, the logical log can fill, which causes the database server to stop processing during a backup. When this happens, you have two options.

- Stop the backup to free the tape device and back up the logical-log files.
- Leave normal processing suspended until the backup completes.

Precautions to take when you use one tape device

When only one tape device exists and you want to create backups while the database server is online, take the following precautions:

- Configure the database server with a large amount of logical-log space through a combination of many or large logical-log files. (See your *IBM® Informix® Administrator's Guide*.)
- Store all explicitly created temporary tables in a dedicated dbspace and then drop the dbspace before backing up.
- Create the backup when low database activity occurs.
- Free as many logical-log files as possible before you begin the backup.

The logical log can fill up before the backup completes. The backup synchronizes with a checkpoint. A backup might wait for a checkpoint to synchronize activity, but the checkpoint cannot occur until all virtual processors exit critical sections. When database server processing suspends because of a full logical-log file, the virtual processors cannot exit their critical sections and a deadlock results.

Specify tape devices as symbolic links

You can specify the values of LTAPEDEV and TAPEDEV as symbolic links. Using symbolic links enables you to switch to other tape or tape-compatible devices without changing the path name in the onconfig file. For example, you can specify the following symbolic link for tape device /dev/rst0:

```
ln -s /dev/rst0 /dbfiles/logtape
```

When you set the LTAPEDEV configuration parameter, as the following example shows, you can switch to a different device without changing the LTAPEDEV parameter:

```
LTAPEDEV /dbfiles/logtape
```

You only need to change the symbolic link, as the following example shows:

```
ln -s /usr/backups /dbfiles/logtape
```

A user with one tape device could redirect a logical-log back up to a disk file while using the tape device for a backup.

Specify a file system directory

You can perform a storage-space (level 0, 1, or 2) archive, or a logical-log backup to a directory in the file system by using the **ontape** utility. For each storage-space archive and logical-log backup, **ontape** creates a file in the specified directory.

To specify a file system directory, set the LTAPEDEV and TAPEDEV configuration parameters to the absolute path name for the directory.

When **ontape** repeats an archive operation, it renames the existing files so that old files are not rewritten. A timestamp is added to the file name, which provides a way for related storage space or logical log files to be organized together.

To learn about the file naming schema, see Rename existing files.

Specify a remote device

You can perform a storage-space or logical-log backup across your network to a remote device attached to another host computer on UNIX and Linux platforms.

You should not do a continuous backup to a remote device.

The remote device and the database server computer must have a trusted relationship so that the **rsh** or the **rlogin** utility can connect from the database server computer to the remote device computer without asking for password. You can establish a trusted relationship by configuring the /etc/hosts.equiv file, the ~/.rhosts file, or any equivalent mechanism for your system on the remote device computer. If you want to use a different utility to handle the remote session than the default utility used by your platform, you can set the DBREMOTECMD environment variable to the specific utility that you want to use.

To specify a tape device on another host computer, use the following syntax to set the TAPEDEV or LTAPEDEV configuration parameter:

```
host_machine_name:tape_device_pathname
```

The following example specifies a tape device on the host computer `kyoto`:

```
kyoto:/dev/rmt01
```

For information about the tape size for remote devices, see Tape size for remote devices.

Related information:
DBREMOTECMD environment variable (UNIX)

Specify /dev/null for a tape device

A best practice is to not use /dev/null as the device when backing up. However, if you decide that you do not need to recover transactions from the logical log, you can specify /dev/null as a tape device for logical-log backups

When you specify /dev/null as a backup tape device, you can avoid the overhead of a level-0 backup that is required after some operations, such as changing the logging status of a database. Obviously, you cannot restore storage spaces from a backup to /dev/null.
When you specify the tape device as /dev/null, block size and tape size are ignored. If you set the LTAPEDEV configuration parameter either to or from /dev/null, you must restart the database server for the new setting to take effect.
Important: When you set the LTAPEDEV configuration parameter to /dev/null, the database server marks the logical-log files as backed up as soon as they become full, effectively discarding logical-log information.

Set TAPEDEV to stdio

To configure the **ontape** utility to read from standard input or write to standard output, set the TAPEDEV configuration parameter to **stdio**.

Rewind tape devices before opening and on closing

With **ontape**, you must use *rewindable* tape devices. Before reading from or writing to a tape, the database server performs a series of checks that require the rewind.

Specify the tape-block-size

Use the TAPEBLK and LTAPEBLK configuration parameters to set the largest block size, in kilobytes, that your tape device permits.

When you set the tape parameter to /dev/null, the corresponding block size is ignored.

The **ontape** utility does not check the tape device when you specify the block size. Verify that the tape device can read the block size that you specified. If not, you cannot restore the tape.

Specify the tape size

Use the TAPESIZE and LTAPESIZE configuration parameter parameters to specify the maximum amount of data that you can write to a tape.

To write or read the tape to the end of the device, set TAPESIZE and LTAPESIZE to 0. You cannot use this option for remote devices.

When you specify the tape device as /dev/null, the corresponding tape size is ignored.

- Tape size for remote devices
  You can estimate the amount of continuous logical-log backup data that can be stored on tape on remote devices.

Related reference:
TAPESIZE configuration parameter
LTAPESIZE configuration parameter

Tape size for remote devices

You can estimate the amount of continuous logical-log backup data that can be stored on tape on remote devices.

When you perform a continuous logical-log backup to a remote device, the amount of data written to the tape is the smaller of LTAPESIZE and the following formula:

```
(sum of space occupied by all logical-log files on disk) -
(largest logical-log file)
```

The I/O to the remote device completes and the database server frees the logical-log files before a log-full condition occurs.

Restriction: You cannot set tape size to 0 for remote devices.

Changing your ontape configuration

You can change the values of the TAPEDEV, TAPEBLK, and TAPESIZE or LTAPEDEV, LTAPEBLK, and LTAPESIZE configuration parameters after performing a backup and reviewing the current and new parameter values.

Prerequisites: Before you change the parameters for **ontape**:

- Perform a level-0 backup.
- Examine your configuration file (the file specified in $INFORMIXDIR/etc/$ONCONFIG) by executing **onstat -c** while the database server is running.
- If you plan to change TAPEDEV or LTAPEDEV to a different tape device, verify that the tape device can read the block size that you specify with the TAPEBLK or LTAPEBLK configuration parameter. If not, you cannot restore the tape. (The **ontape** utility does not check the tape device when you specify the block size.)
- Be sure that you are logged in as user **root** or **informix**.

You can change the values of parameters for **ontape** while the database server is online.

To change the value of TAPEDEV, TAPEBLK, and TAPESIZE or LTAPEDEV, LTAPEBLK, and LTAPESIZE:

1. From the command line, use a text editor to edit your onconfig file.
2. Save the file.

The change takes effect immediately. However, if you set either the TAPEDEV parameter or the LTAPEDEV parameter to /dev/null, you must restart the database server.

Related reference:
LTAPEBLK configuration parameter
LTAPEDEV configuration parameter
LTAPESIZE configuration parameter
TAPEBLK configuration parameter
TAPEDEV configuration parameter
TAPESIZE configuration parameter

Back up with ontape

These topics describe how to use the **ontape** utility to back up storage spaces and logical-log files, and how to change the database logging status. The **ontape** utility can back up and restore the largest chunk files that your database server supports. The **ontape** utility cannot back up temporary dbspaces and temporary sbspaces.

- Summary of ontape tasks
- Change database logging status
  You can use the **ontape** utility to change the logging status of a database. Most changes in logging mode require a full level-0 backup.
- Create a backup
- Back up logical-log files with ontape
  You must only use **ontape** to back up logical-log files when you use **ontape** to make your backup tapes.

Summary of ontape tasks

The **ontape** utility lets you complete a wide variety of tasks:

- Change database logging status
- Create a backup
- Starting a continuous logical-log file backup
- ontape utility syntax: Perform a restore
- Use external restore commands

- Start ontape
- Exit codes for ontape

Start ontape

When you need more than one tape during a backup, the **ontape** utility prompts for each additional tape.

If the database server is in maintenance mode, for example, during a conversion, then the **ontape** utility can only be started by one of the following users:

- **root**
- **informix**
- The user who started the database server (if not the user **root** or **informix**)

Restriction: Do not start the **ontape** utility in background mode (that is, with the UNIX **&** operator on the command line). You could also need to provide input from the terminal or window. When you execute **ontape** in background mode, you can miss prompts and delay an operation.
The **ontape** utility does not include default values for user interaction, nor does it support attempts to retry. When **ontape** expects a yes-or-no response, it assumes that any response not recognized as a “yes” is “no”.

Exit codes for ontape

The **ontape** utility has the following two exit codes:

0
: Indicates a normal exit from **ontape**.

1
: Indicates an exception condition.

Change database logging status

You can use the **ontape** utility to change the logging status of a database. Most changes in logging mode require a full level-0 backup.

You cannot change the logging mode of an ANSI-compliant database.

You can change an unbuffered logged or buffered logged database to an unlogged database without making a backup.

You can make the following logging modes changes with a level-0 backup:

- An unbuffered logged or buffered logged database to an ANSI database
- An unbuffered logged database to a buffered logged database
- A buffered logged database to an unbuffered logged database

Examples

The following command changes the logging mode of a database named **stores7** to unbuffered logging:

```
ontape -s -L 0 -U stores7
```

The following command changes the logging mode of a database to ANSI-compliant logging:

```
ontape -s -L 0 -A stores7
```

The following command changes the logging mode of a database to unlogged:

```
ontape -N stores7
```

Related reference:
ontape utility syntax: Perform a backup
Related information:
Database-logging status

Create a backup

These topics explain how to plan for and create backups of your database server data.

- Backup levels that ontape supports
  The **ontape** utility supports level-0, level-1, and level-2 backups.
- Back up after changing the physical schema
  You must perform a level-0 backup to ensure that you can restore the data after change the physical schema.
- Prepare for a backup
- ontape utility syntax: Perform a backup
  Use **ontape** utility command options to back up to tape.
- Back up to Amazon Simple Storage Service
  You can use the **ontape** utility to back up and restore data to or from the Amazon Simple Storage Service (S3). You are responsible for terms and any charges associated with your use of the Amazon Simple Storage Service.
- When the logical-log files fill during a backup
  When the logical log fills during a backup, the console displays a message and the backup suspends normal processing. How you handle the logical-log filling depends on whether you can use one or two tape devices.
- When a backup terminates prematurely
- Monitor backup history by using oncheck
  You can monitor the history of your last full-system backup by using **oncheck**.

Backup levels that ontape supports

The **ontape** utility supports level-0, level-1, and level-2 backups.

For information about scheduling backups, see Plan a recovery strategy.
Tip: Establish a backup schedule that keeps level-1 and level-2 backups small. Schedule frequent level-0 backups to avoid restoring large level-1 and level-2 backups or many logical-log backups.

Level-0 backup

When a fire or flood, for example, completely destroys a computer, you need a level-0 backup to completely restore database server data on the replacement computer. For online backups, the data on the backup tape reflects the contents of the storage spaces at the time the level-0 backup began. (The time the backup started could reflect the last checkpoint before the backup started.)
A level-0 backup can consume lots of time because **ontape** must write all the pages to tape.

Level-1 backup
A level-1 backup usually takes less time than a level-0 backup because you copy only part of the database server data to the backup tape.

Level-2 backup
A level-2 backup after a level-1 backup usually takes less time than another level-1 backup because only the changes made after the last level-1 backup (instead of the last level-0) get copied to the backup tape.

Back up after changing the physical schema

You must perform a level-0 backup to ensure that you can restore the data after change the physical schema.

Perform a level-0 backup after you make the following administrative changes:

- Changing the TAPEDEV or LTAPEDEV configuration parameter from /dev/null
- Adding logging to a database
- Adding a dbspace, blobspace, or sbspace before you can restore it with anything less than a full-system restore
- Starting mirroring for a dbspace that contains logical-log files
- Dropping a logical-log file
- Moving one or more logical-log files
- Changing the size or location of the physical log and after you set up shared memory
- Dropping a chunk before you can reuse the dbspace that contains that chunk
- Renaming a chunk during a cold restore

Consider waiting to make these changes until your next regularly scheduled level-0 backup.

Tip: Although you no longer need to back up immediately after adding a logical-log file, your next backup should be level-0 because the data structures have changed.

Prepare for a backup

When you create a backup, take the following precautions:

- Avoid temp tables during heavy activity.
- Make sure enough logical-log space exists.
- Keep a copy of your configuration file.
- Verify consistency before a level-0 backup.
- Run the database server in the appropriate mode.
- Plan for operator availability.
- Synchronize with other administrative tasks.
- Do not use background mode.
- If necessary, label tapes appropriately.
- If necessary, prepare for writing to standard output.

- Avoid temp tables during heavy activity
- Make sure enough logical-log space exists
- Keep a copy of your configuration file
- Verify consistency before a level-0 backup
- Online and quiescent backups
- Back up to tape
  When you back up to tape, you must ensure that an operator is available and that you have sufficient media.
- Back up to standard output
  A backup to standard output creates an archive in the memory buffer provided by the operating system. If you choose to back up to standard output, you do not need to provide tapes or other storage media.
- Back up to a directory
  If you choose to back up to a directory, you do not need to provide tapes. Instead, you back up the data to a directory of a local file system or a directory that has been mounted on the local system.

Avoid temp tables during heavy activity

When you create a temp table during a backup while using the **ontape** utility, that table is placed in DBSPACETEMP. When heavy activity occurs during the backup process, the temp table can keep growing and can eventually fill up DBSPACETEMP. When this situation occurs, the backup stops and your monitor displays a NO FREE DISK error message.

Make sure enough logical-log space exists

When the total available space in the logical log amounts to less than half a single logical-log file, the database server does not create a backup. You must back up the logical-log files and attempt the backup again.

You cannot add mirroring during a backup.
Important: When you use only one available tape device, make sure you back up all your logical-log files before you start your backup to reduce the likelihood of filling the logical log during the backup.

Keep a copy of your configuration file

Keep a copy of the current onconfig file when you create a level-0 backup. You need this information to restore database server data from the backup tape.

Verify consistency before a level-0 backup

To ensure the integrity of your backups, periodically verify that all database server data and overhead information is consistent before you create a full-system level-0 backup. You do not check this information before every level-0 backup, but we recommend that you keep the necessary tapes from the most recent backup created immediately after the database server was verified as consistent. For information about consistency checking, see your *IBM® Informix® Administrator's Guide*.

Online and quiescent backups

You can create a backup while the database server is *online* or in *quiescent* mode. The terminal you use to initiate the backup command is dedicated to the backup (displaying messages) until the backup completes. Once you start a backup, the database server must remain in the same mode until the backup finishes; changing the mode terminates the backup activity.

Online backup

You can use an online backup when you want your database server accessible while you create the backup.

Some minor inconveniences can occur during online backups. An online backup can slow checkpoint activity, and that can contribute to a loss in performance. However, this decline in performance is far less costly than the time that you lose when users were denied access to the database server during a backup.

During an online backup, allocation of some disk pages in storage spaces can temporarily freeze. Disk-page allocation is blocked for one chunk at a time until you back up the used pages in the chunk.

Quiescent backup

You create a quiescent backup while the database server is quiescent. Use quiescent backups when you want to eliminate partial transactions in a backup.

Do not use quiescent backups when users need continuous access to the databases.

Back up to tape

When you back up to tape, you must ensure that an operator is available and that you have sufficient media.

Keep an operator available during a backup to mount tapes as prompted. A backup could take several reels of tape. When an operator is not available to mount a new tape when one becomes full, the backup waits. During this wait, when the backup is an online backup, the physical log space could fill up, and that causes the database server to stop the backup. Thus, make sure that an operator is available.

After a tape fills, the **ontape** utility rewinds the tape, displays the tape number for labeling, and prompts the operator to mount the next tape when you need another one. Follow the prompts for labeling and mounting new tapes. A message informs you when the backup is complete.

- Label tapes created with ontape

Label tapes created with ontape

When you label tapes created with the **ontape** utility, the label must include the following information:

- Backup level
- Date and time
- Tape number that **ontape** provides

The following example shows what a label can look like:

```
Level 1: Wed Nov 27, 2001 20:45 Tape # 3 of 5
```

Each backup begins with its first tape reel numbered 1. You number each additional tape reel consecutively thereafter. You number a five-tape backup 1 through 5. (Of course, it is possible that you could not know that it is a five-tape backup until it is finished.)

Back up to standard output

A backup to standard output creates an archive in the memory buffer provided by the operating system. If you choose to back up to standard output, you do not need to provide tapes or other storage media.

Backing up to standard output has the following advantages:

- There are no expensive write and read operations to disk or tape.
- You can use operating system utilities to compress or otherwise process the data.
- You can use the archive to create a duplicate of the server by immediately restoring the data onto another database server.

If you back up to standard output, you must also restore from standard input.
When **ontape** performs a backup to standard output, the data is written to an output file. The directory of the output must have enough disk space to hold the backed-up data. You can use operating system utilities to compress the data. In addition, the user executing the backup command must have write permission to the file to which the backup is diverted or permission to create the file.

When you back up to standard output, **ontape** does not prompt for user interaction. Error and information messages are written to stderr instead of being directed to standard output.

The TAPESIZE configuration parameter is not used because the capacity of standard output is assumed to be unlimited. The TAPEBLK configuration parameter, however, is valid because it defines the size of the transport buffer between the backend server and the **ontape** client. You can optimize throughput by setting TAPEBLK to an appropriate value.

You can simultaneously back up and restore a database server to clone it or set up High-Availability Data Replication. For more information, see Simultaneous backup and restore by using standard I/O.

Related reference:
Restore from standard input

Back up to a directory

If you choose to back up to a directory, you do not need to provide tapes. Instead, you back up the data to a directory of a local file system or a directory that has been mounted on the local system.

The person who runs the backup must have write permission to the directory. The directory must have enough disk space to hold the backed-up data. You can use operating system utilities to compress the data after it is backed up.

Backing up to a directory has the following advantages:

- Multiple instances can simultaneously back up to the same directory file system.
- You can use operating system utilities to compress or otherwise process the data.
- You can easily configure your system to automatically back up a log file when the file is full.

- Set the file directory path
- Rename existing files
  When **ontape** repeats an archive operation, it renames the existing files so that old files are not rewritten. A timestamp is added to the file name, which provides a way for related storage space or logical log files to be organized together.
- Override the default name of the archive files

Set the file directory path

Use the TAPEDEV configuration parameter to specify the absolute path name on a directory of a file system to use for the storage-space archive file. This is the destination where **ontape** writes storage space data during an archive and the source from which **ontape** reads data during a restore. You specify the directory where the logical log backup files are written with the LTAPEDEV configuration parameter.

Tip: When you back up to a directory file system, specify the -d option to turn off **ontape** interactive prompts.

Rename existing files

When **ontape** repeats an archive operation, it renames the existing files so that old files are not rewritten. A timestamp is added to the file name, which provides a way for related storage space or logical log files to be organized together.

Renaming conventions:

- Storage-space archive files
  The archive checkpoint time is added, and has the format `servername_YYYYMMDD_hhmmss_archive-level`.

- Logical log backup files
  The backup time is added, and has the format `servername_YYYYMMDD_hhmmss`.

For example, the file My_instance_L0 is renamed to My_instance_20080913_091527_L0

When restoring from a file system directory, **ontape** requires that storage-space archive and logical-log backup files be named as specified by the TAPEDEV and LTAPEDEV parameters. If files have been renamed, including by **ontape** because of repeated archives and backups, files must be manually renamed to their original file names.

Override the default name of the archive files

You can override the default name of the archive files. When TAPEDEV or LTAPEDEV is a directory path, the default permanent file name consists of *hostname_servernum*_L*n* (for levels), and *hostname_servernum*_Log*nnnnnnnnnn* (for log files). You can override the prefix part of the permanent file name, *hostname_servernum*, by setting the environment variable IFX_ONTAPE_FILE_PREFIX.

For example, if you set IFX_ONTAPE_FILE_PREFIX to “My_Instance”, then during archive, the files are named My_Instance_L0, My_Instance_L1, My_Instance_L2, and, My_Instance_Log0000000001, My_Instance_Log0000000002, and so on. During restore, **ontape** searches for files in the TAPEDEV directory with file names like My_Instance_L0, and searches for files in the LTAPEDEV directory with file names like My_Instance_Log0000000001.

ontape utility syntax: Perform a backup

Use **ontape** utility command options to back up to tape.

Pre-requisites: Before you begin a backup, perform the following steps:

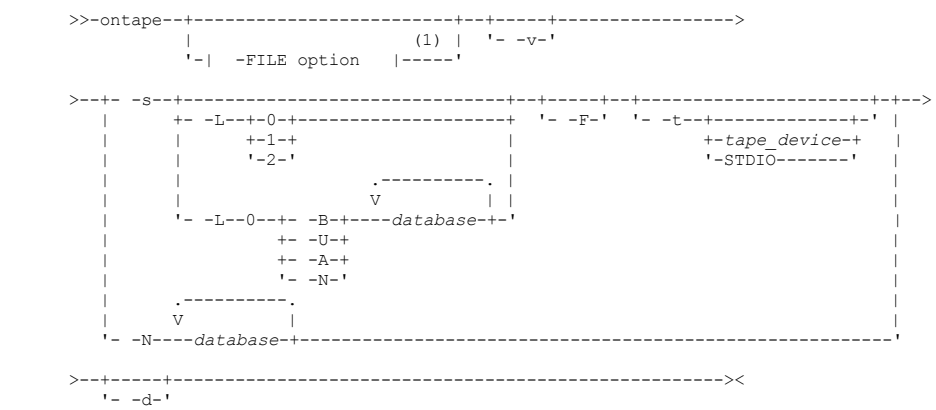
- If necessary, place a write-enabled tape on the tape-drive device that TAPEDEV specifies.
  If you set TAPEDEV to STDIO, ensure that there is enough memory for the backup data.
- If you are using a tape device, put the device online with the appropriate operating-system command.
- Place the database server in online or quiescent mode.
- Log in as the owner of the database server: user **informix** or **root** for a standard installation, or the owner of the non-root installation.

If you are using TAPE devices, do not store more than one backup on the same tape. The tape devices must be of the rewindable type. Begin every backup with a different tape. (Often, a backup spans more than one tape.)

To create a backup, use the **-s** option of the **ontape** command.

Syntax

Create a backup



Notes:

1. See The -FILE option.

Element	Purpose	Key considerations
-A	Directs **ontape** to change the status of the specified database to ANSI-compliant logging.	A database that has ANSI-compliant logging cannot be changed to a different logging mode. A level-0 backup is required to make this logging mode change.
-B	Directs **ontape** to change the status of the specified database to buffered logging.	A level-0 backup is required to make this logging mode change.
-d	Directs **ontape** to proceed without interactive prompts.	You can turn off the prompts if you are backing up to or restoring from a directory of a file system. This option does not apply to tape devices, which must pause the backup while you change tapes.
database	The name of the database to change the logging mode.	The database name cannot include a database server name. More than one database name can be specified in the same command.
-F	Directs **ontape** to perform a fake backup.	A fake backup is only applicable during a backup to standard output. A fake backup is useful for cloning the data in a server. For example, to populate the secondary server in a high-availability cluster. To avoid compromising the normal backup activities, do not keep a record of a fake backup Alternatively, you can use the SQL administration API equivalent: ARCHIVE FAKE. See *IBM® Informix® Administrator's Reference* for more information.

Element	Purpose	Key considerations
-L	Directs **ontape** to create a backup of the level specified.	If you are backing up to tape, use the -L option to specify the backup level as part of the command, you can avoid being prompted for it. If you are backing up to standard output, and do not specify a backup level, **ontape** performs and level-0 backup.
-N	Directs **ontape** to end logging for the specified database.	A backup is optional with this logging mode change.
-s	Directs **ontape** to create a backup.	**ontape** prompts you to supply the backup level (0, 1, or 2) that you want to create if you do not supply a value using the -L option.
-t	Directs **ontape** to use a different tape device for the current backup or restore.	The -t option overrides the value of the TAPEDEV configuration parameter for the current backup or restore. The -t STDIO option directs **ontape** to back up to standard output or restore from standard input.
tape_device	The name of the tape device on which to store the backup.	
-U	Directs **ontape** to change the status of the specified database to unbuffered logging.	A level-0 backup is required to make this logging mode change.
-v	Directs **ontape** to write informational message to stderr during a backup to standard output.	Verbose mode is useful for monitoring the progress of a backup to standard output.

The **ontape** utility backs up the storage spaces in the following order: root dbspaces, blobspaces, sbspaces, and dbspaces.

- Backup examples
- Back up raw tables

Related reference:
Change database logging status

Backup examples

Execute the following command to start a backup to tape without specifying a level: **ontape -s**

You can use the **-L** option to specify the level of the backup as part of the command, as the following example shows: **ontape -s -L 0**

Use the -d option to avoid interactive prompts when you are backing up to or restoring from a directory: **ontape -s -L 0 -d**

When you do not specify the backup level on the command line, **ontape** prompts you to enter it. The following figure illustrates a simple **ontape** backup session.
Figure 1. Example of a simple backup created with ontape

```
ontape -s
Please enter the level of archive to be performed (0, 1, or 2) 0

Please mount tape 1 on /dev/rst0 and press Return to continue ...
16:23:13 Checkpoint Completed: duration was 2 seconds
16:23:13 Level 0 Archive started on rootdbs
16:23:30 Archive on rootdbs Completed.
16:23:31 Checkpoint Completed: duration was 0 seconds

Please label this tape as number 1 in the arc tape sequence.
This tape contains the following logical logs:

 3

Program over.
```

The following example shows how to create a level-0 archive of all storage spaces to standard output, which is diverted to a file named level_0_archive in the directory /home:

```
ontape -s -L 0 >/home/level_0_archive -t STDIO
```

The following example assumes TAPEDEV STDIO in onconfig and creates a level-1 archive to standard output , which is diverted to a pipe:

```
ontape -v -s -L 1|compress -c >/home/compressed/level_1_archive
```

The **compress** system utility reads from the pipe as input, compresses the data, and writes the data to the file level_1_archive in the /home/compressed directory. The **ontape** information messages are sent to stderr.

Back up raw tables

You can use **ontape** to back up a raw table, however, raw tables are not logged. Therefore, when you restore a raw table, any changes that occurred since the last backup cannot be restored. It is recommended that you use raw tables only for initial loading of data and then alter raw tables to standard tables before performing transactions. For more information, see the *IBM® Informix® Administrator's Guide*.

Back up to Amazon Simple Storage Service

You can use the **ontape** utility to back up and restore data to or from the Amazon Simple Storage Service (S3). You are responsible for terms and any charges associated with your use of the Amazon Simple Storage Service.

Prerequisites:

- You must have an Amazon account to perform cloud storage backups. See the Amazon website for instructions about setting up an account.
- Java™ version 1.5 or later is required.
- Backup objects must be 5 GB or smaller.

The following steps show how to back up data to the Amazon Simple Storage Service (S3) System and restore from it by using **ontape** backup and restore utility. In this context, cloud storage refers to an online storage service over the Internet. If you choose to back up to cloud storage, you do not need to provide tapes. Instead, you back up the data to a virtual device, most likely located on the Internet.

1. Configure the online storage device.
   a. Using a web browser, navigate to the Amazon S3 website and log on.
   b. Obtain an access key ID and a secret access key.
   c. Store the access credentials in a file. Set the permissions on the file to allow access only to user executing the **ontape** utility.
      - On UNIX systems, store the values in the file: $INFORMIXDIR/etc/ifxbkpcloud.credentials
      - On Windows systems, store the values in: %INFORMIXDIR%\etc\ifxbkpcloud.credentials

      The file must have the following format:

      ```
      secretKey=secret_access_key
      accessKey=access_key_ID
      ```

   d. Use the **ifxbkpcloud.jar** utility to create and name a storage device in the region where you intend to store data. Amazon uses the term *bucket* to describe the container for backup data. The storage device name you choose has the same restrictions as those for the bucket name in Amazon S3 and must be unique.
      For example, the following command creates a storage device named mytapedevice at a US Standard region on Amazon S3. Run the command from the **$INFORMIXDIR/bin** directory on UNIX systems, or from **%INFORMIXDIR%\bin** on Windows systems.

      ```
      java -jar ifxbkpcloud.jar CREATE_DEVICE amazon mytapedevice US_Standard
      ```

2. Set the TAPEDEV and LTAPEDEV configuration parameters in the onconfig file to point to the cloud storage location. For example:

   ```
   TAPEDEV '/opt/IBM/informix/tapedev_dir, keep = yes, cloud = amazon,
          url = https://mytapedevice.s3.amazonaws.com'
   LTAPEDEV '/opt/IBM/informix/ltapedev_dir, keep = yes, cloud = amazon,
          url = https://mylogdevice.s3.amazonaws.com'
   ```

3. Back up data to the online storage device by using the **ontape** utility.

   ```
   ontape -s -L 0
   ```

   You can restore data from the cloud storage by using the following command:

   ```
   ontape -r
   ```

   You should use https secure data transmission when transferring data to cloud storage. You should encrypt data before transferring data to a cloud image. To encrypt data, use the BACKUP_FILTER and RESTORE_FILTER configuration parameters to call an external encryption program. The archecker utility does not support table-level restore of data from cloud storage.

- The ifxbkpcloud.jar utility
  Use the **ifxbkpcloud.jar** utility to configure an online storage device for the Amazon Simple Storage Service.
- Cloud storage file naming conventions
  Files associated with cloud storage backups have unique file names.

Related reference:
Tape and tape device parameters for ontape
Set the tape-device parameters
Cloud storage file naming conventions
The ifxbkpcloud.jar utility

The ifxbkpcloud.jar utility

Use the **ifxbkpcloud.jar** utility to configure an online storage device for the Amazon Simple Storage Service.

The following options are supported by the **ifxbkpcloud.jar** utility:

- CREATE_DEVICE *provider device [region]*
- DELETE_DEVICE *provider device*
- LIST_DEVICES *provider*
- DELETE_FILE *provider device file*
- LIST_FILES *provider device*

The parameters for the **ifxbkpcloud.jar** commands are defined as follows:

- *provider* is amazon.
- *device* is the name of the storage device.
- *region* is one of the following: US_Standard, US_West, EU_Ireland or AP_Singapore.
- *file* is the name of backup object (key) stored on Amazon S3.

Error messages from **ifxbkpcloud.jar** are written to $INFORMIXDIR/ifxbkpcloud.log on UNIX machines and to %INFORMIXDIR%\ifxbkpcloud.log on Windows machines.

Related tasks:

Back up to Amazon Simple Storage Service

Cloud storage file naming conventions

Files associated with cloud storage backups have unique file names.

Data space backup files are saved by using the following format:

```
hostname_servernum_Larchive_level
```

Log backup file names are saved by using the following format:

```
hostname_servernum_lognnnnnnnnnn
```

If the object exists at the cloud storage location, the file is renamed to avoid overwriting old object. Renaming the file adds a timestamp to the object name.

Data space backup files are saved by using the following format:

```
hostname_servernum_YYYYMMDD_hhmmss_Larchive_level
```

Log backup file names are saved by using the following format:

```
hostname_servernum_lognnnnnnnnnn_YYYYMMDD_hhmmss
```

Related tasks:
Back up to Amazon Simple Storage Service

When the logical-log files fill during a backup

When the logical log fills during a backup, the console displays a message and the backup suspends normal processing. How you handle the logical-log filling depends on whether you can use one or two tape devices.

When you can use two tape devices

When you can use two tape devices with the database server, log in as the owner of the database server: user **informix** or **root** for a standard installation, or the owner of the non-root installation.

Verify that LTAPEDEV and TAPEDEV specify different path names that correspond to separate tape devices. When they do, back up the logical-log files. See Create a backup.

When LTAPEDEV and TAPEDEV are identical, assign a different value to the logical-log tape device (LTAPEDEV) and initiate a logical-log-file backup. Otherwise, your options are to either leave normal database server processing suspended until the backup completes or cancel the backup.

When only one tape device is available

When you create a backup with the only available tape device, you cannot back up any logical-log files until you complete the backup. When the logical-log files fill during the backup, normal database server processing halts. You can either stop the backup (by using **Ctrl-C** only) to free the tape device and back up the logical logs to continue processing, or leave normal processing suspended until the backup completes.

You can take steps to prevent this situation. The section Start an automatic logical-log backup describes these steps.

When a backup terminates prematurely

When you cancel or interrupt a backup, sometimes the backup progresses to the point where you can consider it complete. When listed in the monitoring information, as described in Monitor backup history by using oncheck, you know the backup completed.

Monitor backup history by using oncheck

You can monitor the history of your last full-system backup by using **oncheck**.

Execute the **oncheck -pr** command to display reserved-page information for the root dbspace. The last pair of reserved pages contains the following information for the most recent backup:

- Backup level (0, 1, or 2)
- Effective date and time of the backup
- Time stamp describing when the backup began (expressed as a decimal)
- ID number of the logical log that was current when the backup began
- Physical location in the logical log of the checkpoint record (that was written when the backup began)

The effective date and time of the backup equals the date and time of the checkpoint that this backup took as its starting point. This date and time could differ markedly from the time when the backup process was started.

For example, when no one accessed the database server after Tuesday at 7 P.M., and you create a backup Wednesday morning, the effective date and time for that backup is Tuesday night, the time of the last checkpoint. In other words, when there has been no activity after the last checkpoint, the database server does not perform another checkpoint at the start of the backup.

Back up logical-log files with ontape

You must only use **ontape** to back up logical-log files when you use **ontape** to make your backup tapes.

In addition to backing up logical-log files, you can use **ontape** to switch to the next log file, move logical-log files to other dbspaces, or change the size of the logical log. Instructions for those tasks appear in your *IBM® Informix® Administrator's Guide*.

- Before you back up the logical-log files
- When to back up logical-log files
- Start an automatic logical-log backup
- Starting a continuous logical-log file backup
  When you do not want to monitor the logical-log files and start backups when the logical-log files become full, you can start a continuous backup.
- End a continuous logical-log backup
- Devices that logical-log backups must use

Before you back up the logical-log files

Before you back up the logical-log files, you need to understand the following issues:

- Whether you need to back up the logical-log files
- When you need to back up the logical-log files
- Whether you want to perform an automatic or continuous backup

For more information about these issues, see Logical-log backup.

- Use blobspace TEXT and BYTE data types and logical-log files
- Use /dev/null when you do not need to recover

Use blobspace TEXT and BYTE data types and logical-log files

You must keep in mind the following two points when you use TEXT and BYTE data types in a database that uses transaction logging:

- To ensure timely reuse of blobpages, back up logical-log files. When users delete TEXT or BYTE values in blobspaces, the blobpages do not become freed for reuse until you free the log file that contains the delete records. To free the log file, you must back it up.
- When you must back up an unavailable blobspace, **ontape** skips it and makes it impossible to recover the TEXT or BYTE values when it becomes necessary. (However, blobpages from deleted TEXT or BYTE values do become free when the blobspace becomes available even though the TEXT or BYTE values were not backed up.)

In addition, regardless of whether the database uses transaction logging, when you create a blobspace or add a chunk to a blobspace, the blobspace or new chunk is not available for use until the logical-log file that records the event is not the current logical-log file. For information about switching logical-log files, see your *IBM® Informix® Administrator's Guide*.

Use /dev/null when you do not need to recover

When you decide that you do not need to recover transactions or administrative database activities between backups, you can set the database server configuration parameter LTAPEDEV to /dev/null.
Important: When you set LTAPEDEV to /dev/null, it has the following implications:

- You can only restore the data that your database server manages up to the point of your most recent backup and any previously backed-up logical-log files.
- When you perform a recovery, you must always perform a full-system restore. (See Full-system restore.) You cannot perform partial restores or restore when the database server is online.

When you set LTAPEDEV to /dev/null, the database server marks a logical-log file as backed up (status B) as soon as it becomes full. The database server can then reuse that logical-log file without waiting for you to back it up. As a result, the database server does not preserve any logical-log records.

Fast recovery and rolling back transactions are not impaired when you use /dev/null as your log-file backup device. For a description of fast recovery, see your *IBM® Informix® Administrator's Guide*. For information about rolling back transactions, see the ROLLBACK WORK statement in the *IBM Informix Guide to SQL: Syntax*.

When to back up logical-log files

You must attempt to back up each logical-log file as soon as it fills. You can tell when you can back up a logical-log file because it has a *used* status. For more information about monitoring the status of logical-log files, see your *IBM® Informix® Administrator's Guide*.

Start an automatic logical-log backup

The database server can operate online when you back up logical-log files. To back up all full logical-log files, use the **-a** option of the **ontape** command.

```
Request a logical-log backup

>>-ontape -a-------------------------------------------------><
```

The **-a** option backs up all full logical-log files and prompts you with an option to switch the logical-log files and back up the formerly current log.

When the tape mounted on LTAPEDEV becomes full before the end of the logical-log file, **ontape** prompts you to mount a new tape.

When you press the Interrupt key while a backup occurs, the database server finishes the backup and then returns control to you. Any other full logical-log files receive a used status.

To back up all full logical-log files, execute the **ontape -a** command.

Starting a continuous logical-log file backup

When you do not want to monitor the logical-log files and start backups when the logical-log files become full, you can start a continuous backup.

When you start a continuous backup, the database server automatically backs up each logical-log file as it becomes full. When you perform continuous logical-log file backups, the database server protects you against ever losing more than a partial logical-log file, even in the worst case media failure when a chunk that contains logical-log files fails.

To start a continuous backup of the logical-log files, use the **ontape -c** command. The **-c** option initiates continuous backup of logical-log files. The database server backs up each logical-log file as it becomes full. Continuous backup does not back up the current logical-log file. The database server can operate in online mode when you start continuous backups.

Whether the logical-log files are backed up to tapes or a directory depends on the setting of the LTAPEDEV configuration parameter:

- If the LTAPEDEV configuration parameter is set to a tape device, someone must always make media available for the backup process. When the specified mounted tape becomes full before the end of the logical-log file, the database server prompts the operator for a new tape. Also, you must dedicate the backup device to the backup process.
- If the LTAPEDEV configuration parameter is set to a directory, logical-log files can be backed up unattended. Logical logs are backed up as they fill and a new file is created in the directory for each logical log. Backup is limited by space available for new files.

To back up to a directory, as an alternative to using the **ontape -c** command, you can call the **ontape -a -d** automatic logical log backup command from a script specified by the ALARMPROGRAM configuration parameter. You can use either the alarmprogram or script or the log_full script, both of which are found in the $INFORMIXDIR/etc directory.

To use the alarmprogram script to back up logical logs to a directory:

1. Set the LTAPEDEV parameter to an existing directory. Make sure that this directory is owned by **informix** and group **informix**.
2. Edit the ALARMPROGRAM script ($INFORMIXDIR/etc/alarmprogram.sh on UNIX or Linux or %INFORMIXDIR%\etc\alarmprogram.bat on Windows), as follows:
   a. Set the BACKUPLOGS parameter within the file to `Y`.
   b. Change the backup program from `onbar -b -l` to `ontape -a -d`.
3. Restart the database server.

End a continuous logical-log backup

To end continuous logical-log backup, press the Interrupt key (**CTRL-C**).

When you press the Interrupt key while the database server backs up a logical-log file to a local device, all logs that were backed up before the interrupt are captured on the tape and are marked as backed up by the database server.

When you press the Interrupt key while the database server waits for a logical-log file to fill (and thus is not backing up any logical-log files), all logs that were backed up before the interrupt reside on the tape and are marked as backed up by the database server.

When you press the Interrupt key while the database server performs a continuous backup to a remote device, any logical-log files that were backed up during this operation can or cannot reside on the tape, and are not marked as backed up by the database server (a good reason why you should not do continuous remote backups).

After you stop continuous logging, you must start a new tape for subsequent log backup operations.

You must explicitly request logical-log backups (by using **ontape -a**) until you restart continuous logging.

Devices that logical-log backups must use

The **ontape** utility uses parameters defined in the onconfig file to define the tape device for logical-log backups. However, consider the following issues when you choose a logical-log backup device:

- When the logical-log device differs from the backup device, you can plan your backups without considering the competing needs of the backup schedule.

- When you specify /dev/null as the logical-log backup device in the configuration parameter LTAPEDEV, you avoid having to mount and maintain backup tapes. However, you can only recover data up to the point of your most recent backup tape. You cannot restore work done after the backup. See the warning about setting LTAPEDEV to /dev/null in Use /dev/null when you do not need to recover.
  If the log backup device on any server node in a high-availability cluster is set to /dev/null (on Linux or UNIX) or NUL (on Windows), then the backup device for all of the other servers within the cluster (including the primary server and any HDR, RSS or SDS secondary servers) must also be set to /dev/null (or NUL).
- When your tape device runs slow, the logical log could fill up faster than you can copy it to tape. In this case, you could consider performing the backup to disk and then copying the disk backup to tape.

Restore with ontape

These topics provide instructions for restoring data with the **ontape** utility for the following procedures:

- A whole-system restore
- A restore of selected dbspaces, blobspaces, and sbspaces

Before you start restoring data, you must understand the concepts in Restore systems. As explained in that section, a complete recovery of database server data generally consists of a physical restore and a logical restore.

- Types of physical restore
- Cold, warm, or mixed restores
- ontape utility syntax: Perform a restore
  Use the **-r** option to perform a full physical and logical restore of the database server data with **ontape**. Use the **-D** option to restore selected storage spaces. Use the **-rename** option to rename chunks during the restore.
- Restore the whole system
  This section outlines the prerequisites and steps you need to complete to restore your entire database server with **ontape**.
- Configuring continuous log restore with ontape
- Rename chunks during a restore
- Restore from standard input
  You can perform a restore from standard input, you must first have performed a backup to standard output.
- Restore data to a remote server
  You can restore data to a remote server with the **ontape** utility.
- Simultaneous backup and restore by using standard I/O

Types of physical restore

If a failure causes the database server to go offline, you must restore all the database server data. This type of restore is a *full-system* restore. You can only restore data to the same version of IBM® Informix®. When the failure did not cause the database server to go offline, you can restore only the storage spaces that failed. For illustrations of the restore types, see Warm, cold, and mixed restores.

- Full-system restore
- Restores of dbspaces, blobspaces, and sbspaces

Full-system restore

When your database server goes offline because of a disk failure or corrupted data, it means that a critical dbspace was damaged. The following list shows critical dbspaces:

- The root dbspace
- The dbspace that contains the physical log
- A dbspace that contains logical-log files

When you need to restore any critical dbspace, you must perform a full system restore to restore all the data that your database server manages. You must start a full-system restore with a cold restore. See Cold, warm, or mixed restores.

Restores of dbspaces, blobspaces, and sbspaces

When your database server does not go offline because of a disk failure or corrupted data, the damage occurred to a noncritical dbspace, blobspace, or sbspace.

When you do not need to restore a critical dbspace, you can restore only those storage spaces that contain a damaged chunk or chunks. When a media failure occurs in one chunk of a storage space that spans multiple chunks, all active transactions for that storage space must terminate before the database server can restore it. You can start a restore operation before the database server finishes the transactions, but the restore becomes delayed until the database server verifies that you finished all transactions that were active at the time of the failure.

Cold, warm, or mixed restores

When you restore the database server data, you must decide whether you can do it while the database server is offline or online. This decision depends in part on the data that you intend to restore.

Cold restores

Perform a *cold restore* while the database server is offline. It consists of both a physical restore and a logical restore. You must perform a cold restore to restore any critical dbspaces.

The database server is offline when you begin a cold restore but it goes into recovery mode after it restores the reserved pages. From that point on it stays in recovery mode until either a logical restore finishes (after which it works in quiescent mode) or you use the **onmode** utility to shift it to another mode.

You can rename chunks by specifying new chunks paths and offsets during a cold restore. This option is useful if you need to restore storage spaces to a different disk from the one on which the backup was made. You can rename any type of chunk, including critical chunks and mirror chunks. For more information, see Rename chunks during a restore. You can also rename chunks for an external cold restore; see Rename chunks for more information.

A cold restore can be performed after a dbspace has been renamed and a level-0 backup or a backup of the rootdbs and renamed dbspace is performed.

Warm restores

A *warm restore* restores noncritical storage spaces while the database server is in online or quiescent mode. It consists of one or more physical restore operations (when you restore multiple storage spaces concurrently), a logical-log backup, and a logical restore.

During a warm restore, the database server replays backed-up logical-log files for the storage spaces that you restore. To avoid overwriting the current logical log, the database server writes the logical-log files that you designate for replay to temporary space. Therefore, a warm restore requires enough temporary space to hold the logical log or the number of log files being replayed, whichever is smaller. For information about how the database server looks for temporary space, see the discussion of DBSPACETEMP in the *IBM® Informix® Administrator's Guide*.

Important: Make sure that enough temporary space exists for the logical-log portion of the warm restore; the maximum amount of temporary space that the database server needs equals the size of all the logical-log files.
A warm restore can be performed after a dbspace has been renamed and a level-0 archive of the rootdbs and renamed dbspace is taken.

Mixed restores

A *mixed restore* is a cold restore followed by a warm restore. A mixed restore restores some storage spaces during a cold restore (the database server is offline) and some storage spaces during a warm restore (the database server is online). You could do a mixed restore when you perform a full-system restore, but you need to provide access to a particular table or set of tables as soon as possible. In this case, perform a cold restore to restore the critical dbspaces and the dbspaces that contain the important tables.

A cold restore takes less total time to restore all your data than a mixed restore, even though the database server is online during part of a mixed restore because a mixed restore requires two logical restores (one for the cold restore and one for the warm restore). A mixed restore, however, requires the database server to go offline for less time than a cold restore.

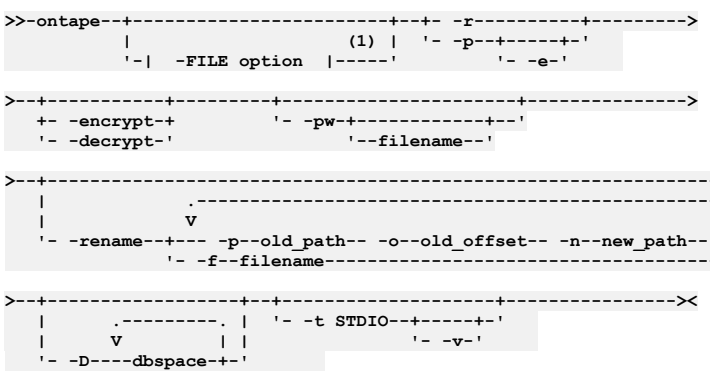
The dbspaces not restored during the cold restore do not become available until after the database server restores them during a warm restore, even though a critical dbspace possibly did not damage them.

ontape utility syntax: Perform a restore

Use the **-r** option to perform a full physical and logical restore of the database server data with **ontape**. Use the **-D** option to restore selected storage spaces. Use the **-rename** option to rename chunks during the restore.

You must run the **ontape** command as the owner of the database server: user **informix** or **root** for a standard installation, or the owner of the non-root installation. The owner of the database server for the restore must be the same as the owner of the database server for the backup.

Run a full or physical restore



```
Run a logical restore

>>-ontape--+- -l--+- -C-+-+-------------------------------------><
           |      '- -X-' |
           '- -S----------'
```

Element	Purpose	Key considerations
-C	Restores logs from the current logical log tape without sending prompts to mount the tape.	The server is placed in suspend log restore state, and the command exits after the last applicable log is restored. The server sends a prompt if a log spans tapes.
-D	Directs **ontape** to restore only the storage spaces you specify.	The database server must go into online or quiescent mode to do a warm restore. When you use the **-D** option, you can restore selected storage spaces. When you do not specify the **-D** option, **ontape** performs a full-system restore. The database server must go offline to do a full-system restore. For more information, see Restore selected storage spaces.
dbspace	Is the name of a storage space to restore.	You can specify multiple storage spaces, but you must include the root dbspace.
-decrypt	Specifies to decrypt any encrypted storage spaces during the physical restore of the spaces.	For more information about storage space encryption, see Changing storage space encryption during a restore.
-e	Directs **ontape** to perform an external restore	For more information, see Perform an external backup and restore. This option is compatible with renaming chunks for external cold restores.
-encrypt	Specifies to encrypt storage spaces during the physical restore of the spaces.	Storage space encryption must be enabled by the DISK_ENCRYPTION configuration parameter. Otherwise, storage spaces are not encrypted during the restore. For more information about storage space encryption, see Changing storage space encryption during a restore.
-f *filename*	Specifies a file containing the names and offsets of chunks to be renamed and their new locations. Use to rename many chunks at one time.	The file name can be any valid UNIX or Windows file name, including simple (listfile_1), relative (../backup_lists/listfile_2 or ..\backup_lists\listfile2), and absolute (/usr/informix/backup_lists/listfile3 or c:\informix\backup_lists\listfile3) file names. In the file, list the old chunk path name and offset and the new chunk path name and offset, with a blank space or a tab between each item. Put information for each chunk on a separate line. Blank lines are ignored. Begin comment lines with a # symbol.
-l	Directs **ontape** to perform a logical restore.	The **-l** option restores data from the logical-log backup tapes you created after (and including) your last level-0 backup.
-p	Directs **ontape** to perform a physical data restore.	The **-p** option restores data from the backup tape you created after (and including) your last level-0 backup. During the restore, the database server is in single-user mode.
-p *old_path* **-o** *old_offset***-n** *new_path* **-o** *new_offset*	Specifies the chunk to be renamed and its new location. Use to rename one or more chunks at one time.	The variables for this element are: *old_path* The current path and file name of the chunk. *old_offset* The current offset of the chunk, in kilobytes. *new_path* The new path and file name of the chunk. *new_offset* The new offset of the chunk.
-r	Directs **ontape** to perform a data restore (both physical and logical).	The **-r** option restores data from the backup tape and the logical-log backup tapes you created after (and including) your last level-0 backup.
-rename	Directs **ontape** to rename the specified chunks.	For more information about renaming chunks during a restore, see Rename chunks during a restore.
-S	Directs **ontape** to perform a logical log salvage.	If you want to salvage logical logs, you must use the **-S** option before performing a restore from standard input. The LTAPEDEV configuration parameter must be set to the logical log tape device.
-t STDIO	Directs **ontape** to restore from standard input.	The **-t** option overrides the value of the TAPEDEV configuration parameter for the current restore.
-v	Directs **ontape** to write informational message to stderr during a restore from standard input.	Verbose mode is useful for monitoring the progress of a restore from standard input.
-X	Quiesces a server in logical restore suspend state without restoring additional logs.	Include this option with **-r -l** to end continuous log restore of logical logs.

Note: The -pw option is required only when the Storage space encryption feature is enabled and no stash file is in use. Supply an optional path to a file containing the keystore password, otherwise ontape will prompt for a password before performing the restore.

Restore the whole system

This section outlines the prerequisites and steps you need to complete to restore your entire database server with **ontape**.

The following list summarizes the main steps in a full-system restore:

1. Gather the appropriate backup and logical log tapes.

2. Decide on a complete cold or a mixed restore.
3. Verify your database server configuration.
4. Perform a cold restore.

Familiarize yourself with these instructions before you attempt a full-system restore.

- Gather backup and logical-log tapes before restoring
  Before you restore an entire database system, you must gather backup and logical log tapes. If you changed the names of backup and logical log files, you must also manually rename the files to their original file names.
- Decide on a complete cold or a mixed restore
- Verify your database server configuration
- Perform a cold restore
  To perform a cold restore with **ontape**, the database server must be offline.
- Restore selected storage spaces
- Restore raw tables

Gather backup and logical-log tapes before restoring

Before you restore an entire database system, you must gather backup and logical log tapes. If you changed the names of backup and logical log files, you must also manually rename the files to their original file names.

Backup tapes
: Gather all the tapes from your latest level-0 backup that contain the storage spaces you are restoring and any subsequent level-1 or level-2 backups.
Identify the tape that has the latest level-0 backup of the root dbspace on it; you must use this tape first.

Logical-log tapes
: If at the time of the archive checkpoint, an open transaction started, gather all logical-log tapes before you perform the level-0 backup.
Gather all the logical-log tapes from the backup after the latest level-0 backup of the storage spaces you are restoring.

When using **ontape** to create an archive backup of the system, a snapshot of the logical logs is included with the archive. At the end of the archive, the system displays a message that indicates what logical logs are included in the archive. The snapshot is included in the archive so that if any open transactions exist at the time of the backup, those transactions can be reconciled when the archive is restored. Then:

- If you decide not to replay any logical logs, the system can be brought to a consistent state.
- If you decide to replay logical logs, the logs contained within the archive backup are discarded and you must replay transactions from the logical log backups.
  The starting log file is the oldest log file containing an open transaction at the time of the last restored archive. You can identify that log file from the message that was displayed when the last archive was restored.

Example:

- The **ontape -s -L 0** command performs a level-0 backup of the system and displays a message that states that the archive contains logs 2-4.
- The **ontape -s -L 1** command performs a level 1 incremental backup of the system and displays a message that states that the archive contains logs 8-9.

If you restore only the level-0 archive and want to replay logs, you need log backups starting with log 2. If you restore the level-0 and level-1 archives and want to replay logs, you need log backups starting with log 8.
The restore of the logical log files uses an archive format, not a log file format. However, the logs contained within the restored archive are in log file format, not an archive format.

File names when restoring from directory

When restoring from a file system directory, **ontape** requires that storage-space archive and logical-log backup files be named as specified by the TAPEDEV and LTAPEDEV configuration parameters. If files were renamed, including by **ontape** because of repeated archives and backups, you must manually rename the files to their original file names. To learn about the naming conventions for storage-space archive files and logical-log backup files, see Back up to a directory for the naming conventions of these files.

Decide on a complete cold or a mixed restore

As mentioned in Cold, warm, or mixed restores, when you restore your entire database server, you can restore the critical dbspaces (and any other storage spaces you want to come online quickly) during a cold restore, and then restore the remaining storage spaces during a warm restore. Decide before you start the restore if you want a cold restore or a mixed restore.

Verify your database server configuration

During a cold restore, you cannot set up shared memory, add chunks, or change tape devices. Thus, when you begin the restore, the current database server configuration must remain compatible with, and accommodate, all parameter values assigned after the time of the most recent backup.

For guidance, use the copies of the configuration file that you create at the time of each backup. However, do not set all current parameters to the same values as were recorded at the last backup. Pay attention to the following three groups of parameters:

- Shared-memory parameters
- Mirroring parameters
- Device parameters

- Set shared-memory parameters to maximum assigned value

- Set mirroring configuration to level-0 backup state
- Verify that the raw devices or files are available

Set shared-memory parameters to maximum assigned value

Make sure that you set your current shared-memory parameters to the maximum value assigned after the level-0 backup. For example, if you decrease the value of USERTHREADS from 45 to 30 sometime after the level-0 backup, you must begin the restore with USERTHREADS set at 45, and not at 30, even though the configuration file copy for the last backup could register the value of USERTHREADS set at 30. (When you do not possess a record of the maximum value of USERTHREADS after the level-0 backup, set the value as high as you think necessary. You could reassign values to BUFFERPOOL, LOCKS, and TBLSPACES as well because the minimum values for these three parameters are based on the value of USERTHREADS.)

Set mirroring configuration to level-0 backup state

Verify that your current mirroring configuration matches the configuration that was in effect at the time of the last level-0 backup. Because it is recommended that you create a level-0 backup after each change in your mirroring configuration, this creates no problems. The most critical parameters are the mirroring parameters that appear in the configuration file, MIRRORPATH and MIRROROFFSET.

Verify that the raw devices or files are available

Verify that the raw devices or files that you used for storage (of the storage spaces being restored) after the level-0 backup are available.

For example, if you drop a dbspace or mirroring for a dbspace after your level-0 backup, you must make the dbspace or mirror chunk device available to the database server when you begin the restore. When the database server attempts to write to the chunk and cannot find it, the restore does not complete. Similarly, if you add a chunk after your last backup, you must make the chunk device available to the database server when it begins to roll forward the logical logs.

Perform a cold restore

To perform a cold restore with **ontape**, the database server must be offline.

You must run the **ontape** command as the owner of the database server: user **informix** or **root** for a standard installation, or the owner of the non-root installation. The owner of the database server for the restore must be the same as the owner of the database server for the backup.

Run the following **ontape** command to restore all the storage spaces: **ontape -r**

When you perform a mixed restore, you restore only some of the storage spaces during the cold restore. You must restore at least all the critical dbspaces, as the following example shows:

```
ontape -r -D rootdbs llogdbs plogdbs
```

- Salvage logical-log files
- Mount tapes during the restore
- Restore logical log files
- Bring the database server online when the restore is over

Salvage logical-log files

Before the cold restore starts, the console prompts you to salvage the logical-log files on disk. To salvage the logical-log files, use a new tape. It saves log records that you did not back up and enables you to recover your database server data up to the point of the failure.

The following example shows a log salvage:

```
...
Continue restore? (y/n) y
Do you want to back up the logs? (y/n) y

Please mount tape 1 on /dev/ltapedev and press Return to continue.
Would you like to back up any of logs 31 - 32? (y/n) y
Logical logs 31 - 32 may be backed up.
Enter the id of the oldest log that you would like to backup? 31

Please label this tape as number 1 in the log tape sequence.

This tape contains the following logical logs:
    31-32
Log salvage is complete, continuing restore of archive.
Restore a level 1 archive (y/N) y
Ready for level 1 tape
...
```

Mount tapes during the restore

During the cold restore, **ontape** prompts you to mount tapes with the appropriate backup files.

When restoring from a directory, the prompt specifies the absolute path name of the directory. Before responding to the prompt, you can copy or rename the file in the directory.

You can avoid the prompt by using the **ontape -d** option. When using this option, ensure that storage-space archive and logical-log backup files exist in the directory, as specified by the TAPEDEV and LTAPEDEV parameters. The **ontape** utility scans the directories for the files and uses them for the restore. After restoring the newest applicable logical-log backup file, **ontape** automatically commits the restore and brings the IBM® Informix® instance into quiescent mode.

Restore logical log files

When you perform a mixed restore, you must restore all the logical-log files backed up after the last level-0 backup.

When you perform a full restore, you can choose not to restore logical-log files. When you do not back up your logical-log files or choose not to restore them, you can restore your data only up to the state it was in at the time of your last backup. For more information, see Back up logical-log files with ontape.

To restore the logical logs, use the **ontape -l** command.

Bring the database server online when the restore is over

At the end of the cold restore, the database server is in quiescent mode. You can bring the database server online and continue processing as usual.

When you restore only some of your storage spaces during the cold restore, you can start a warm restore of the remaining storage spaces after you bring the database server online.

Restore selected storage spaces

These topics outline the steps that you must perform during a restore of selected storage spaces with **ontape** while the database server is in online or quiescent mode (a warm restore). During a warm restore, you do not need to worry about shared-memory parameters as you do for cold restores.

Before you attempt a restore, familiarize yourself with these instructions.

The following list describes the main steps in a warm restore:

1. Gather the appropriate tapes
2. Ensure that needed device are available
3. Back up logical-log files
4. Perform a warm restore
   To perform a warm restore with **ontape**, the database server must operate in online or quiescent mode.

Gather the appropriate tapes

Gather the appropriate backup and logical-log tapes.

Backup tapes

Before you start your restore, gather together all the tapes from your latest level-0 backup that contain the storage spaces you are restoring and any subsequent level-1 or level-2 backups.

Logical-log tapes

Gather together all the logical-log tapes from the logical-log backup after the latest level-0 backup of the storage spaces you are restoring.

Next topic: Ensure that needed device are available

Ensure that needed device are available

Verify that storage devices and files are available before you begin a restore. For example, when you drop a dbspace or mirroring for a dbspace after your level-0 backup, you must ensure that the dbspace or mirror chunk device is available to the database server when you begin the restore. If the storage device is not available, the database server cannot write to the chunk and the restore fails.

When you add a chunk after your last backup, you must ensure that the chunk device is available to the database server when it rolls forward the logical logs.

Previous topic: Gather the appropriate tapes

Next topic: Back up logical-log files

Back up logical-log files

Before you start a warm restore (even when you perform the warm restore as part of a mixed restore), you must back up your logical-log files. See Back up logical-log files with ontape.

After the warm restore, you must roll forward your logical-log files to bring the dbspaces that you are restoring to a state of consistency with the other dbspaces in the system. Failure to roll forward the logical log after restoring a selected dbspace results in the following message from **ontape**:

```
Partial system restore is incomplete.
```

Previous topic: Ensure that needed device are available
Next topic: Perform a warm restore

Perform a warm restore

To perform a warm restore with **ontape**, the database server must operate in online or quiescent mode.

You must run the **ontape** command as the owner of the database server: user **informix** or **root** for a standard installation, or the owner of the non-root installation. The owner of the database server for the restore must be the same as the owner of the database server for the backup.

To restore selected storage spaces, run the **ontape** command, with the options that the following example shows:

```
ontape -r -D dbspace1 dbspace2
```

You cannot restore critical dbspaces during a warm restore; you must restore them as part of a cold restore, described in Restore the whole system.

During the restore, **ontape** prompts you to mount tapes with the appropriate backup files.

At the end of the warm restore, the storage spaces that were down go online.

Previous topic: Back up logical-log files

Restore raw tables

When you use **ontape** to restore a raw table, it contains only data that existed on disk at the time of the backup. Because raw tables are not logged, any changes that occurred since the last backup cannot be restored. For more information, see Back up raw tables and the *IBM® Informix® Administrator's Guide*.

Configuring continuous log restore with ontape

Ensure that the version of IBM® Informix® is identical on both the primary and secondary systems.
Use continuous log restore to restart a log restore with newly available logs after all currently available logs have been restored. For more information, see Continuous log restore.

To configure continuous log restore with **ontape**:

1. On the primary system, perform a level-0 archive with the **ontape -s -L 0** command.
2. On the secondary system, copy the files or mount the tape (as assigned by LTAPEDEV) and perform a physical restore with the **ontape -p** command.
3. Respond to the following prompts:

   ```
   Continue restore? Y
   Do you want to back up the logs? N
   Restore a level 1 archive? N
   ```

   After the physical restore completes, the database instance waits in fast recovery mode to restore logical logs.

4. On the primary system, back up logical logs with the **ontape -a** command.
5. On the secondary system, copy the files or mount the tape that contains the backed up logical logs from the primary system. Perform a logical log restore with the **ontape -l -C** command.
6. Repeat steps 4 and 5 for all logical logs that are available to back up and restore.
7. If you are doing continuous log restore on a secondary system as an emergency standby, run the following commands to complete restoring logical logs and quiesce the server
   - If logical logs are available to restore, use the **ontape -l** command.
   - After all available logical logs are restored, use the **ontape -l -X** command.

Related concepts:
Continuous log restore

Rename chunks during a restore

You can rename chunks during a cold restore with **ontape**. This option is useful if you need to restore storage spaces to a different disk from the one on which the backup was made. You can rename any type of chunk, including critical chunks and mirror chunks.

The **ontape** rename chunk restore only works for cold restores.

The critical dbspaces (for example, the rootdbs) must be restored during a cold restore. If you do not specify the list of dbspaces to be restored, then the server restores the critical dbspaces and all the other dbspaces. But if you specify the list of dbspaces to be restored, then the critical dbspaces must be included in the list.

For the syntax of renaming chunks with **ontape**, see ontape utility syntax: Perform a restore.

Tip: If you use symbolic links to chunk names, you might not need to rename chunks; you need only edit the symbolic name definitions. For more information, see the *IBM® Informix® Administrator's Guide*.
You can rename chunks during an external cold restore. See Rename chunks for more information.

Validation sequence for renaming chunks

During a cold restore, **ontape** performs the following validations to rename chunks:

- It validates that the old chunk path names and offsets exist in the archive reserved pages.
- It validates that the new chunk path names and offsets do not overlap each other or existing chunks.
- If renaming the primary root or mirror root chunk, it updates the onconfig file parameters ROOTPATH and ROOTOFFSET, or MIRRORPATH, and MIRROROFFSET. The old version of the onconfig file is saved as $ONCONFIG.*localtime*.
- It restores the data from the old chunks to the new chunks (if the new chunks exist).
- It writes the rename information for each chunk to the online log.

If either of the validation steps fails, the renaming process stops and **ontape** writes an error message to the **ontape** activity log.
Important:

- Perform a level-0 archive after you rename chunks; otherwise your next restore fails.
- If you add a chunk after performing a level-0 archive, that chunk cannot be renamed during a restore. Also, you cannot safely specify that chunk as a new path in the mapping list.
- Renaming chunks for database servers participating in HDR involves a significant amount of offline time for both database servers. For more information, see the *IBM® Informix® Administrator's Guide*.

New chunk requirements

To rename a chunk, follow these guidelines for new chunks:

- The new chunk does not need to exist
  You can install the new chunk later and perform a warm restore of a storage space containing it. If you specify a nonexistent chunk, **ontape** records the rename information in the chunk reserved pages, but does not restore the data. The renamed (but not restored) chunks have a status of offline, designated by `D`, in the **onstat -d** chunk status command output.

- New chunks must have the proper permissions.
  Rename operations fail unless the chunks have the proper permissions. For more information, see the *IBM® Informix® Administrator's Guide*.

Rename chunks with command-line options

To rename the chunks by supplying information about the command line, use this command:

```
ontape -r -rename -p /chunk1 -o 0 -n /chunk1N -o 20000
          -rename -p /chunk2 -o 10000 -n /chunk2N -o 0
```

Perform a level-0 archive after the rename and restore operation is complete.

Rename chunks with a file

To rename the chunks by supplying a file named listfile, use the following command: **ontape -r -rename -f listfile**

The contents of the listfile file are:

```
/chunk1 0 /chunk1N 20000
/chunk2 10000 /chunk2N 0
```

Perform a level-0 archive after the rename and restore operation is complete.

Rename chunks while specifying other options

To rename the chunks with command-line options while performing a restore of **dbspace1** and **dbspace2** where the **rootdbs** is the rootdbs, use the following command:

```
ontape -r -rename -p /chunk1 -o 0 -n /chunk1N -o 20000
          -rename -p /chunk2 -o 10000 -n /chunk2N -o 0
          -D rootdbs dbspace1 dbspace2
```

Alternatively, to rename the chunks by using file while performing a restore of **dbspace1** and **dbspace2**, use the following command:

```
ontape -r -rename -f listfile -D rootdbs dbspace1 dbspace2
```

Perform a level-0 archive after the rename and restore operation is complete.

Rename a chunk to a nonexistent device

To rename a chunk to a device that does not yet exist, you specify the new path name, but you do not restore its storage spaces until after you install the physical device. This option is useful if you need to rename a chunk and it is convenient to perform a cold restore before you install the new device. When the new chunk device is ready, you can perform a warm restore of a storage space onto it.

You can combine renaming chunks with existing devices and renaming chunks with nonexistent devices in the same rename operation. This example shows how to rename a single chunk to a nonexistent device name.

The following table lists example values for the chunks used in this example.

Storage space	Old chunk path	Old offset	New chunk path	New offset
sbspace1	/chunk3	0	/chunk3N	0

- Renaming a chunk to a nonexistent device

Renaming a chunk to a nonexistent device

To rename a chunk to a nonexistent device:

1. Rename the chunk: using the following command: **ontape -r -rename -p /chunk3 -o 0 -n /chunk3N -o 0**
2. When the following prompt appears, enter `y` to continue:

   ```
   The chunk /chunk3N does not exist. If you continue, the restore
   may fail later for the dbspace which contains this chunk.
   Continue without creating this chunk? (y/n)
   ```

   The chunk `/chunk3` is renamed to `/chunk3N`, but the data has not yet been restored to `/chunk3N`.
3. Perform a level-0 archive.
4. Add the physical device for `/chunk3N`.
5. Perform a warm restore of **sbspace1** with the **ontape -r -D sbspace1** command.
6. Perform a level-0 archive.

Restore from standard input

You can perform a restore from standard input, you must first have performed a backup to standard output.

When you perform a restore from standard input, **ontape** does not prompt you for options or information. If **ontape** cannot perform the operation with the information you provided in the restore command, **ontape** exits with an appropriate error. Restoring from standard input differs from restoring from tapes in the following ways:

- No logical restore or logical log salvage occurs.
  To perform a logical restore, use the **ontape -l** command after the physical restore.

  To salvage logical logs, use the **ontape -S** command before the physical restore.
- You are not prompted to confirm the restore. Informational messages about the archive are sent to stderr.
  If you detect a problem, you can interrupt the restore during the 10 second delay between the completion of the archive information and starting the database server.

Examples

In the following example, **ontape** performs a physical restore from the file level_0_archive, which contains the archive previously performed to standard output:

```
cat /home/level_0_archive | ontape -p
```

In the following example, **ontape** performs a restore of a level-0 archive, followed by a restore of a level-1 archive:

```
cat /home/level_0_archive /home/level_1_archive | ontape -r
```

In the following example, **ontape** performs a restore of sbspace1:

```
cat/home/level_0_archive | ontape -r -D spspace1 -t STDIO
```

When these restores are completed, the database server is left in single-user mode.

Related reference:
Back up to standard output

Restore data to a remote server

You can restore data to a remote server with the **ontape** utility.

The remote server must have the following characteristics:

- Identical hardware and operating systems
- Identical database server versions and editions
- The same configuration and ROOTPATH information, although the server names and numbers can differ.

You can restore data to a remote server with the following command:

```
ontape -s -L 0 -F | rsh remote_server "ontape -p"
```

However, the process might hang after completing successfully. You have three primary options:

- Terminate the remote shell process
- Execute the remote shell from the remote server with the following command:

  ```
  rsh local_server "ontape -s -L 0 -F" | ontape -p
  ```

- Redirect the standard output (stdout) and standard error (stderr) on the remote server with the following command from the sh or bash shell:

  ```
  ontape -p >/dev/null 2>&1
  ```

- You can simplify this redirection by placing it in a shell script, ontape.sh, on the remote server. You can issue the following command from the local server:

  ```
  ontape -s -L 0 -F | rsh remote_server /my/path/ontape.sh
  ```

- The shell script ontape.sh contains the following text:

  ```
  #!/bin/sh
  #define some environment variables, such as

  INFORMIXDIR=/... ; export INFORMIXDIR
  INFORMIXSQLHOSTS=/...; export
  INFORMIXSQLHOSTS ONCONFIG=/...; export ONCONFIG
  INFORMIXSERVER=/...; export INFORMIXSERVER
  PATH=/...; export PATH
  # invoke ontape with stdout/stderr redirection

  ontape -p >/dev/null 2>&1
  ```

Simultaneous backup and restore by using standard I/O

To clone a database server or quickly set up High-Availability Data Replication (HDR), you can perform a simultaneous backup to standard output and restore from standard input. If you perform the backup and restore solely to duplicate a database server, use the -F option to prevent the archive from being saved.

On HDR, the secondary server can restore only level-0 archives.

To use standard I/O to perform the backup and restore, set the TAPEDEV configuration parameter to STDIO, or you can specify **-t STDIO** from the command line.

For example, if the TAPEDEV configuration parameter is set to STDIO, the following command loads data into the secondary server on an HDR pair (named **secondary_host**).

```
ontape -s -L 0 -F | rsh secondary_host "ontape -p"
```

In the next example, assume that the TAPEDEV configuration parameter is not set. The following command loads data into the secondary server of an HDR pair (named **secondary_host**):

```
ontape -s -L 0 -F -t STDIO | rsh secondary_host "ontape -t STDIO -p"
```

The examples perform a fake level-0 archive of the database server on the local computer, pipe the data to the remote computer by using the **rsh** system utility, and perform a physical restore on the remote computer by reading the data directly from the pipe.
Important: The previous examples require that the INFORMIXDIR, INFORMIXSERVER, INFORMIXSQLHOSTS, and ONCONFIG environment variables be set in the default environment for the user on the remote computer on which the command is executed. The user must be **informix** or **root**.

Perform an external backup and restore

These topics discuss performing an external backup and recovering data by restoring it with the **ontape** utility.

- Recover data by using an external backup and restore
  You can perform an *external backup and restore*, which eliminates the downtime of systems because the backup and restore operations are performed external to

the IBM® Informix® system.

- Data that is backed up in an external backup
- Prepare for an external backup
- Data that is restored in an external restore

Recover data by using an external backup and restore

You can perform an *external backup and restore*, which eliminates the downtime of systems because the backup and restore operations are performed external to the IBM® Informix® system.

The **ontape** utility does not move the data during the backup or physical restore. An external backup allows you to copy disks that contain storage-space chunks without using **ontape**. When disks fail, replace them and use vendor software to restore the data, then use **ontape** for the logical restore. For more information, see Data that is restored in an external restore.

The following are typical scenarios for external backup and restore:

- Availability with disk mirroring
  If you use hardware disk mirroring, you can get your system online faster with external backup and restore than with conventional **ontape** commands.
- Cloning
  You can use external backup and restore to clone an existing production system for testing or migration without disturbing the production system.

Data that is backed up in an external backup

Before you begin an external backup, block the database server. Blocking forces a checkpoint, flushes buffers to disk, and blocks user transactions that involve temporary tables. During the blocking operation, users can access that database server in read-only mode. Then you can physically back up or copy the data to another set of disks or storage media by using operating-system or third-party tools. When you complete the external backup, unblock the database server so that transactions can resume. You should include all the chunk files in each storage space and administrative files, such as onconfig, in an external backup.
Important: To make tracking backups easier, it is recommended that you back up all storage spaces in each external backup.
The **ontape** utility treats an external backup as equivalent to a level-0 backup. You cannot perform an external backup and then use **ontape** to perform a level-1 backup, or vice versa because **ontape** does not have any record of the external backup. For more information, see Performing a cold external restore.

- Rules for an external backup
- Performing an external backup

Rules for an external backup

Before you begin an external backup, keep in mind the following rules:

- The database server must be online or quiescent during an external backup.
- Use **ontape** to back up all logical logs including the current log so that you can restore the logical logs at the end of the external restore.
- Suspend continuous logical-log backups before you block the database server for an external backup. After the external backup is complete, resume the continuous logical-log backup.
- Wait until all **ontape** backup sessions have completed before you block the database server. If any backup sessions are active, the block command displays an error message.
- Any OLTP work or queries are suspended while the database server is blocked. They resume after the database server is unblocked.
- All critical dbspaces of the database server instance must be backed up together simultaneously within the same command bracket of **onmode -c block ... onmode -c unblock**. Backups of different critical dbspaces done at different times cannot be restored to a consistent system.

Important: Because the external backup is outside the control of **ontape**, you must track these backups manually. For more information, see Track an external backup.

Performing an external backup

The database server must be online or in quiescent mode during an external backup.

To perform an external backup without disk mirroring:

1. To obtain an external backup, block the database server with the **onmode -c block** command. The system takes a checkpoint and suspends all update transactions. Users can access the database server in read-only mode.
2. To back up the storage spaces and administrative files, use a copy command, such as **cp**, **dd**, or **tar** on UNIX or **copy** on Windows, or a file-backup program.
   You must back up all chunks in the storage spaces.
3. To allow normal operations to resume, unblock the database server with the **onmode -c unblock** command.
4. Back up all the logical logs including the current log so that checkpoint information is available for the external restore.
   Important: Because external backup is not done through **ontape**, you must ensure that you have a backup of the current logical log from the time when you execute the **onmode -c block** command. Without a backup of this logical-log file, the external backup is not restorable.
5. After you perform an external backup, back up the current log. using the **ontape -a** command.

If you lose a disk or the whole system, you are now ready to perform an external restore.

Prepare for an external backup

These topics describe the commands used to prepare for an external backup. For the procedure, see Performing an external backup.

- Block and unblock the database server
  This section shows the syntax of the block and unblock commands.
- Track an external backup

Block and unblock the database server

This section shows the syntax of the block and unblock commands.

```
>>-onmode-- -c--+---------+-------------------------------------><
               +-block---+
               '-unblock-'
```

Element	Purpose	Key considerations
-c	Performs a checkpoint and blocks or unblocks the database server	None.
block	Blocks the database server from any transactions	Sets up the database server for an external backup. While the database server is blocked, users can access it in read-only mode. Sample command: **onmode -c block**
unblock	Unblocks the database server, allowing data transactions and normal database server operations to resume	Do not unblock until the external backup is finished. Sample command: **onmode -c unblock**

Track an external backup

The database server and **ontape** do not track external backups. To track the external backup data, use a third-party storage manager or track the data manually. The following table shows the items we recommend that you track in an external backup.

Table 1. Items to track when you use external backup and restore

Items to track	Examples
Full path names of each chunk file for each backed up storage space	UNIX: /work/dbspaces/rootdbs Windows: c:\work\dbspaces\rootdbs
Object type	Critical dbspaces, noncritical storage spaces
ins_copyid_hi and **ins_copyid_lo**	Copy ID that the storage manager assigns to each backup object
Backup date and time	The times that the database server was blocked and unblocked
Backup media	Tape volume number or disk path name
Database server version	Version 14.10

Data that is restored in an external restore

If you lose a disk or the whole system, you can externally restore data only if it was externally backed up. You must use the same third-party utility for both the external backup and restore. To externally restore the storage spaces, copy the backed up data to disk. Use the **ontape -p -e** command to mark the storage spaces as physically restored, replay the logical logs with the **ontape -l** command, and bring the storage spaces back online. If you do not specify an external restore command, the database server cannot update the status of these storage spaces to online.

You can only perform a cold external restore with **ontape**. A cold external restore marks storage spaces as physically restored, then performs a logical restore of all storage spaces.

When you perform a cold external restore, **ontape** does not first attempt to salvage logical-log files from the database server because the external backup has already copied over the logical-log data.

To salvage logical logs, perform **ontape -S** before you copy the external backup and perform the external restore (**ontape -p -e**).

- Use external restore commands
- Rules for an external restore
  Before you begin an external restore, know what you can and cannot restore from an external backup and be aware of the rules for an external restore.
- Rename chunks
  You can rename chunks in an external cold restore by using the rename options syntax for other restores.
- Performing a cold external restore
- Initializing HDR with an external backup and restore

Use external restore commands

Use the **ontape -p -e** command to perform a cold external restore. This command marks the storage spaces as physically restored. The following diagram shows the external physical restore syntax.

```
Perform an external physical restore

>>- -p-- -e-------------------------------------------------><
```

Element	Purpose	Key considerations
-e	Specifies an external restore	Must be used with the -p option.
-p	Specifies a physical restore	In a cold restore, if you do not specify storage space names, all of them are marked as restored. After the physical restore completes, you must perform a logical restore.

Use the **ontape -l** command to perform a logical restore. For more information, see ontape utility syntax: Perform a restore.

Rules for an external restore

Before you begin an external restore, know what you can and cannot restore from an external backup and be aware of the rules for an external restore.

These requirements and rules are:

- You must externally restore from an external backup. Although the external backup is treated as a level-0 backup, it might actually be an incremental backup that was created by another source than Informix®.
- You cannot externally restore temporary dbspaces.
- You cannot externally restore from regular **ontape** backups.
- You cannot verify that you are restoring from the correct backup and that the storage media is readable with **ontape**.
- If the external backups are from different times, the external restore uses the beginning logical log from the oldest backup.
- Salvage the logical logs (**ontape -l**) before you switch the disks that contain the critical storage spaces.
- If you are restoring critical dbspaces, the database server must be offline.
- If you are restoring the rootdbs, disable mirroring during the restore.
- The external backups of all critical dbspaces of the database server instance must have been simultaneous. All critical dbspaces must have been backed up within the same **onmode -c block ... onmode -c unblock** command bracket.

Rename chunks

You can rename chunks in an external cold restore by using the rename options syntax for other restores.

Use the following commands to rename chunks during an external cold restore:

```
ontape -p -e -rename -f filename
```

or

```
ontape -p -e -rename -p old_path -o old_offset-n new_path-o new_offset
```

Performing a cold external restore

If you specify the **ontape -p -e** command in a cold restore, you must restore all storage spaces. Use the **ontape -p -e** command to restore all storage spaces.

To perform a cold external restore:

1. Shut down the database server with the **onmode -ky** command.
2. To restore the storage spaces from an external backup, use a copy command, such as **cp**, **dd**, or **tar** on UNIX or a file-backup program.
   You must restore the storage spaces to the same path as the original data.
3. To perform an external restore of all storage spaces followed by a logical restore, use the following commands:
   - **ontape -p -e**
   - **ontape -l**

- Examples of external restore commands

Examples of external restore commands

The following table contains an example of external restore commands.

External restore command	Action	Comments

External restore command	Action	Comments
ontape -p -e **ontape -l**	Physical external restore and logical restore	The system restores the logical logs from the oldest external backup.
ontape -p -e -rename -f	External cold restore with renamed chunks	

Initializing HDR with an external backup and restore

You can use external backups to initialize High-Availability Data Replication (HDR).

To initialize HDR with an external backup and restore:

1. Block the source database server with the **onmode -c block** command.
2. Externally back up all chunks on the source database server.
3. When the backup completes, unblock the source database server with the **onmode -c unblock** command.
4. Make the source database server the primary server with the following command: **onmode -d primary** *secondary_servername*
5. On the target database server, restore the data from the external backup with a copy or file-backup program.
6. On the target database server, restore the external backup of all chunks with the **ontape -p -e** command.
7. Make the target database server the secondary server with the following command: **onmode -d secondary** *primary_servername*
8. If the logical-log records written to the primary database server since step 1 still reside on the primary database server disk, the secondary database server reads these records to perform the logical recovery. Otherwise, perform the logical recovery with the **ontape -l** command.
   The `database server operational` messages appear in the message log on the primary and secondary servers.

Backup and restore a Remote Secondary Server(RSS)

It is possible to archive a Remote Secondary Server and to back up logical logs on that node, using onbar or ontape. This archive may then be used to rebuild the RSS if necessary, saving time over using an archive that is taken on the primary and copied to the secondary machine.

Although an archive and log backups taken on an RSS node may be used to restore a primary node, they are recommended for this purpose only when no other archive exists. The logical log position on the secondary is often several logs behind that of the primary, which means the last logical logs backed up on the RSS may not be the last log completed on the primary. This is not a problem when the archive is used to recreate the RSS, because once reconnected to the primary the logs will be resynchronized and no transactions will be lost.

Prerequisites for taking archives and log backups on an RSS node are as follows:

1. Enable the feature by setting the BAR_SEC_ALLOW_BACKUP configuration parameter to 1 and restarting the RSS. This parameter may not be tuned dynamically.
2. Ensure that the RSS node has one or more active temporary dbspaces, and that they are listed in the DBSPACETEMP configuration parameter. Once an archive begins, all pages physically logged for a particular space will need to be stored until that space has been archived. The temporary dbspaces listed in DBSPACETEMP are used for this before-image storage. The total amount of temporary space required will vary according to the update load on the RSS during the archive, which will fail to complete if it runs out of temporary space.
3. Set the TAPEDEV, LTAPEDEV, and BAR-related configuration parameters to appropriate values depending on your preferred backup utility.
4. The primary node must not contain any of the following non-logged objects:
   1) BLOB spaces

   2) Non-logged smartblobs

   3) No-log databases

   4) Raw tables

   If any of these unlogged objects are present in the instance, an archive taken on the RSS node will fail because the archive would be incomplete and therefore unusable for restoration on a primary node.

5. In order to back up logical logs on an RSS node the LTAPEDEV configuration parameter must *not* be set to the Null device. If logical logs will not be backed up on the RSS node, LTAPEDEV *must* be set to the Null device. Note that these two rules apply on an RSS node only when BAR_SEC_ALLOW_BACKUP is set to 1. When BAR_SEC_ALLOW_BACKUP is set to 0 the setting of LTAPEDEV on the RSS node must be in sync with that of the primary, and no logical log backups will be allowed or required on the RSS node regardless of the LTAPEDEV setting.

If it becomes necessary to recreate the RSS node from an archive and log backups taken on that node, the same procedures used for restoring archives taken on the primary may be used with the local archive. No new configuration changes or steps are required.

Even when taking archives and log backups on an RSS node, it is recommended that archives and log backups be taken on the primary as well, because restoration of the primary is likely to be faster with a local backup and there is a greater chance that all committed transactions will be salvaged and restored. Again, the risk of lost transactions is not an issue when recreating the RSS from a local backup as long as the primary node can forward all missing logs to the RSS during synchronization.

In an emergency it is possible to perform a cold restore of a primary node using an archive taken on an RSS and log backups taken either on the RSS or the primary. This process is made simpler with the **ontape** utility but is also feasible with **onbar**.

- If using **ontape**, simply restore the archive and roll forward the log backups as usual, regardless of the origin of these backup elements.
- If using **onbar** the storage manager containing all backup objects must be available on the primary, and you must replace the ixbar file on the primary with the ixbar file from the RSS node before performing the restore.

Using an archive taken on an RSS node for a warm restore on the primary node is not recommended.

Related reference:
BAR_SEC_ALLOW_BACKUP configuration parameter

Integrated Backup Encryption

These topics provide information about Integrated Backup Encryption.

Although it is possible to encrypt backups since version 11.10.xC1 using Backup Filters, the process of setting up encryption keys and keeping track of all the elements necessary for the encryption and decryption of backups is neither short or easy, and so, the Backup filter functionally has been mostly relegated to compress/decompress backups, which can be achieved more easily.

Note: Encrypting backups is risky. If you misplace your encryption key or delete a remote master encryption key, you can render any number of backups unusable. If you misplace the encryption key for a backup or lose access to the Remote Master Key, there is no way for anybody, including technical support, to restore those backups, they are lost forever.
Although there is a way to encrypt the backups using a local encryption key provided by the operator, Integrated backup Encryption was designed to work mainly with Remote Key Servers because they offer the flexibility and reliability needed to minimized the likelihood of rendering backups unusable due misplaced/missing encryption keys.

Integrated Backup Encryption does not reuse the encryption keys used for Storage Space Encryption. When a backup is performed, the engine decrypts the pages before sending them to the backup client and the On-Bar/ontape utilities receive a stream of unencrypted pages.

The backup client then generates an encryption key called Backup Encryption Key (Depending on the capabilities of the RKS, the backup encryption key can be generated locally, or at the RKS). The backup encryption key is then used to encrypt the backup data.

The backup client also encrypts the backup encryption key using a Remote Master Encryption Key (RMEK) to generate an Encrypted Backup Encryption Key (EBEK) and stores the identification of the Remote Master Key, the Encrypted Backup Encryption Key, and other relevant information necessary to decrypt the data in a structure called the Encryption Envelope (envelope for simplicity). The envelope structure is stored together with the encrypted backup data and therefore it is impossible to lose or misplace the backup encryption key since it is always stored together with the data that it protects.

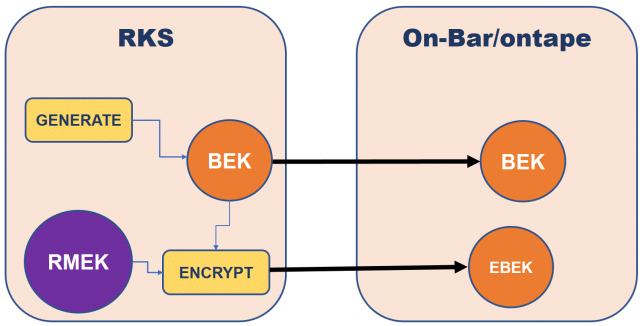
As long as there is access to the RKS and the Remote Master Encryption Key is not deactivated, the backup will be decryptable.

The process of encrypting a backup, as already described above, requires the generation of a backup encryption key for each backup session. All backup objects generated in that session will share the same BEK (For On-Bar, this means that each storage space and log file backed up will share the same BEK. For ontape, it means that every volume generated will be encrypted with the same BEK).

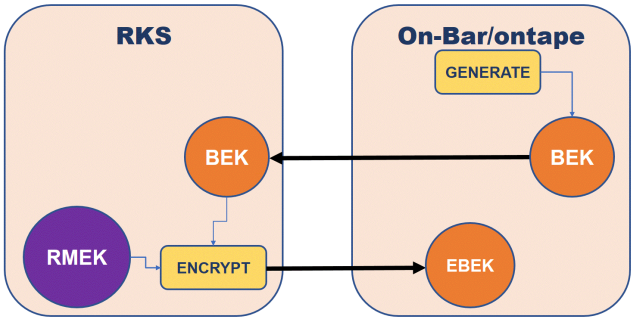
Depending on the capabilities of the RKS, there are two ways in which this BEK can be generated:

- Method 1: The RKS is capable of generating symmetric encryption keys. In this case the RKS will generate the BEK and provide the backup client with both the BEK and the product of encrypting the BEK with the Remote Master Encryption Key (EBEK).

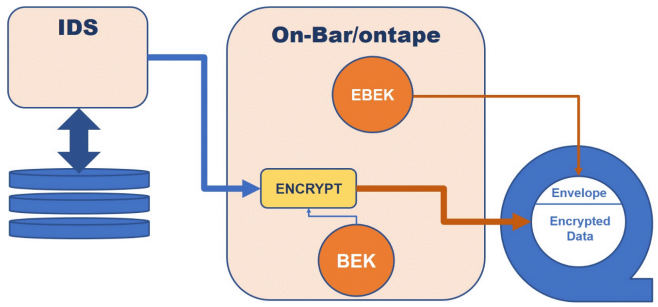
Figure 1. Method 1 to generate BEK



Method 2:If the Remote Key Server does not support the creation of symmetric encryption keys, the BEK is locally generated, the BEK is then transferred to the RKS where it is encrypted using the RMEK, then RKS returns the EBEK to generate the encryption envelope.
Figure 2. Method 2 to generate BEK



Note: In both methods, it is necessary that the RKS has cryptographic capabilities, meaning that the RKS has to be capable of encrypting and decrypting data using the RMEK. If the RKS is not capable of providing cryptographic operations (which is the case of some KMIP-enabled servers) it is not possible to use Integrated Backup Encryption with that server. This is done to minimize the risk of key exposure/leakage (since the RMEK never leaves the RKS, the chances of compromising the key are minimized).
Once the BEK is generated and the backup client has the RMEK Id, the BEK, and the EBEK, it can generate the encryption envelope, encrypt the backup data and send the encrypted data to the backup medium/server.
Figure 3. Backup Encryption



The BAR_ENCRYPTION configuration parameter

In order to use Integrated Backup Encryption, you must setup either a local key file or access to a remote keys server. Then you need to set the BAR_ENCRYPTION configuration parameter to let know the backup client that you want to use Integrated Backup Encryption and which method you want to use.

- Using a Remote Key Server
  These topics provide information about Remote Key Server.
- Using a Local Encryption Key
  These topics provide information about Local Encryption Key

Using a Remote Key Server

These topics provide information about Remote Key Server.

The usage of a remote key server allows you to work with Integrated Backup Encryption.

We currently support:

1. KMIP complaint servers that support the ENCRYPT and DECRYPT cryptographic operations.
2. The Amazon Web Services Key Management Service (AWS-KMS).
3. The Microsoft Azure Key Vault service.

In order to use a remote key server, you must provide the appropriate credentials to connect to it. The credentials to access the server are stored in a keystore generated by the onkstore utility.

The credential types supported by Integrated backup encryption are KMIP (for KMIP servers), “aws-bar” for AWS-KMS and “azure-bar” for Azure KeyVault. Any keystore with other types of credentials (ie AWS-EAR) are not supported and its usage will result in an error.

The parameters required to create each type of credentials vary depending from the provider, you will need to understand the meaning of this parameters and how to request/generate them. For example, what we describe in this document as Remote Master Encryption Key (RMEK) is known as “Azure Key Name” for Azure KeyVault or “AWS CMD Id” (AWS Customer Master Key Id) for AWS KMS.

For more information, see The onkstore Utility

Using a Local Encryption Key

These topics provide information about Local Encryption Key

To use a local encryption key, the operator must manually generate an encryption key of the appropriate size for the cipher you want to use (ie a 192-bit, or 24 byte long encryption key). Then store it in a text file in base64 format. The file must have 600 permissions in UNIX/Linux and must be readable only the DBSA. In Windows the file must be owned by the Administrators group or the Informix user and readable only by the owner.

Once the file has been created the full path to the file must be set in the BAR_ENCRYPTION configuration parameter together with the cipher to use.

Note: It is not recommended to use local encryption keys, however they are necessary in certain scenarios. If you misplace your encryption key, there is no way for anybody, including technical support, to recover that backup.
Example to create the local encryption file for aes192 using the openssl utility:

```
openssl rand -base64 24 > /home/informix/etc/l_key192
```

Example to create the local encryption file for aes128 using the openssl and base64 utilities:

```
openssl rand 16 | base64  > /home/informix/etc/l_key128
```

Example on how the BAR_ENCRYPTION configuration parameter will look for the first example:

```
BAR_ENCRYPTION  keyfile=/home/informix/etc/l_key192,cipher=aes192
```

The keystore used to hold local Master Encryption Keys for Storage Space Encryption is not supported by Integrated Backup Encryption.

Informix Primary Storage Manager

The IBM® Informix® Primary Storage Manager manages storage for ON-Bar backup and restore operations, including parallel backups, that use file devices (disks).

- IBM Informix Primary Storage Manager
  IBM Informix Primary Storage Manager is an application that manages storage devices used for backup and restore requests that are issued by ON-Bar. This storage manager supports both serial and parallel processing for backup and restore requests.

IBM Informix Primary Storage Manager

IBM® Informix® Primary Storage Manager is an application that manages storage devices used for backup and restore requests that are issued by ON-Bar. This storage manager supports both serial and parallel processing for backup and restore requests.

Informix Primary Storage Manager consists of the following components:

onpsm utility
A command-line utility that you can use to perform the following tasks:

- Create, modify, and delete storage devices
- Define and modify the maximum sizes for devices
- Move backup information from one device to another within a device pool
- Determine whether volumes, storage objects, and devices are locked or busy
- Release locked volumes, storage objects, and devices
- Verify volume names and labels

XBSA shared library
A unique version of the X/Open Backup Services API (XBSA) shared library that ON-Bar and the Informix Primary Storage Manager use to communicate with each other. When ON-Bar stores or retrieves data that is stored on storage devices, the storage manager coordinates the request through the XBSA interface at the device level. You specify the location of the XBSA shared library with the BAR_BSALIB_PATH configuration parameter.

Storage catalog tables
A set of flat files that track information about all storage objects, devices, and device pools. These files are required to restore backup objects that are created by Informix Primary Storage Manager. By default, these files are stored in the $INFORMIXDIR/etc/psm directory. You can use the PSM_CATALOG_PATH configuration parameter to specify another location for the storage catalog tables.
Important:

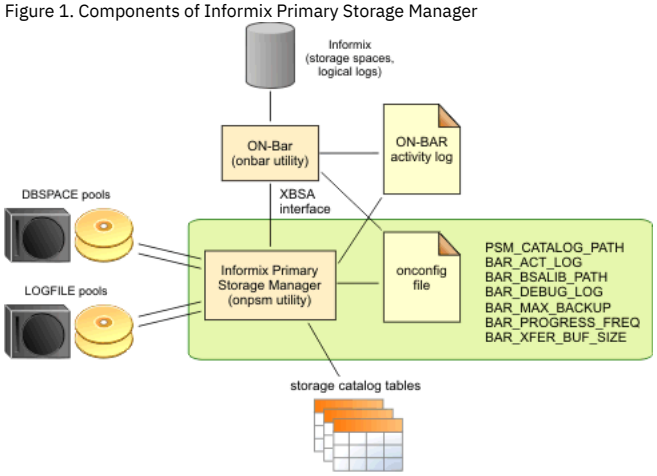
- Back up the storage catalog tables with your operating system tools as part of a disaster recovery strategy. The storage catalog tables are not backed up with the database instance and they are not associated with system catalog tables.
- To prevent the storage catalog tables from getting too large, delete old generations of backups regularly. Use the **onsmsync** utility to manage expiration policies.

The configuration parameters that you use to configure the Informix Primary Storage Manager are in the onconfig file.

You define and maintain storage devices with the **onpsm** command-line utility. You can configure one device at a time or generate a device-configuration file to configure multiple devices. During backups, Informix Primary Storage Manager selects a device from a pool of available devices. If the device becomes full or fails, the storage manager automatically moves to another device in the same pool.

Informix Primary Storage Manager writes informational, warning and error messages to the storage manager activity log. You can use the PSM_ACT_LOG configuration parameter to specify the location of the activity log. If the PSM_ACT_LOG configuration parameter does not contain information, the storage manager puts activity information in the directory specified with the BAR_ACT_LOG configuration parameter.

Figure 1. Components of Informix Primary Storage Manager



Features of Informix Primary Storage Manager

Storage manager feature	Explanation
Storage devices to use with the storage manager	File devices only The storage manager automatically creates a default device when a catalog is created. The default device is $INFORMIXDIR/backups. You can remove the default device.
Buffer transfer size	Unlimited
Encryption and compression	Achieved with BACKUP_FILTER, RESTORE_FILTER FILTERS in ON-Bar (The storage manager does not provide encryption or compression.)

Storage manager feature	Explanation
Expiration policies of the storage manager	No expiration policies. (You manually expire backup objects from the storage manager with the **onsmsync** utility. The **onsmsync** object expiration commands remove objects from the storage manager.)

You can perform an imported restore with ON-Bar and the Informix Primary Storage Manager. In an imported restore, you back up the Informix instance on one machine and restore the instance on a different machine. Use the **onsmsync** export and import options to export the backup objects from the storage manager on the backup machine and import the backup objects into the storage manager on the restore machine.

Backups to Cloud and STDIO devices

Using STDIO devices for backup and restore:

- PSM will write/read a stream of data to an external utility (i.e. sftp or curl)
- Operator to provide parameters to invoke the utility for read/write/drop operations
- Transfer of data to the third party program will happen using STDIO, PSM will write to the standard input of the utility and will read from its standard output
- The Operator do not have direct access to the stream of data
- In order to use this feature the operator must create a PSM device of type "STDIO"
- A device of type STDIO will require you provide the path to the program to be executed as the device name (i.e. /usr/bin/curl)
- Also you must provide the arguments to call the program during backup, restore and drop
- You can optionally provide the maximum size of a file, if the backup is bigger than this size, it will be broken up in pieces of this size

The new command line is:

```
onpsm -D -add /usr/bin/sftp.sh -t STDIO --stdio_warg "BACKUP @obj_name1@.@obj_id@.@obj_part@" --stdio_rarg "RESTORE
@obj_name1@.@obj_id@.@obj_part@"
--stdio_darg "DELETE @obj_name1@.@obj_id@.@obj_part@" --max_part_size <size in KB>
```

- Examples: Manage storage devices with Informix Primary Storage Manager
  Learn how to set up and use Informix Primary Storage Manager to manage storage devices that the onbar utility uses for backing up and restoring instances. Each example shows how you can use the storage manager for a specific backup strategy.
- Setting up Informix Primary Storage Manager
  Setting up involves gathering and specifying information about your storage devices and, if necessary, changing the default configuration of the storage manager.
- Managing storage devices
  Use the **onpsm** utility to add, monitor, and remove storage devices and to manage IBM Informix Primary Storage Manager catalogs, locks, and objects. Use the **onsmsync** utility to export ON-Bar backups to and import them from external pools and to expire backups.
- The onpsm utility for storage management
  Use the **onpsm** utility to manage the IBM Informix Primary Storage Manager catalogs, devices, locks, and objects.
- Device pools
  The IBM Informix Primary Storage Manager pool is a named group of disk devices that you use as a repository for backups.
- Device-configuration file for the Informix Primary Storage Manager
  The **onpsm** utility can generate a device-configuration file, which is a text file that contains information about a storage device. The utility uses this information to re-create the devices.
- Informix Primary Storage Manager file-naming conventions
  When creating files that store your backup data, the IBM Informix Primary Storage Manager uses specific file-naming conventions.
- Message logs for Informix Primary Storage Manager
  The Informix Primary Storage Manager writes messages to a storage manager activity log and debug log.

Related concepts:
Backup Services API (XBSA)
Device pools
Related tasks:
Examples: Manage storage devices with Informix Primary Storage Manager
Related reference:
Configure a storage manager
The onpsm utility for storage management
Informix Primary Storage Manager configuration parameters

Examples: Manage storage devices with Informix Primary Storage Manager

Learn how to set up and use Informix® Primary Storage Manager to manage storage devices that the onbar utility uses for backing up and restoring instances. Each example shows how you can use the storage manager for a specific backup strategy.

Prerequisites:

- 14.10 is installed with the ON-Bar utility.
- Environment variable INFORMIXDIR is set to the path where the database server is installed.
- Environment variable ONCONFIG is set to the file in $INFORMIXDIR/etc that contains the configuration parameters for your database. The name of the file must be unique for each database server instance.
- User **informix** or root privileges.

- Example 1: Storing backups for an instance
- Example 2: Storing backups for two instances
- Example 3: Exporting backups to and restoring them from another directory
- Example 4: Exporting a backup from one server and importing it into another server

In these examples, *storage manager* refers to Informix Primary Storage Manager.

Related concepts:
IBM Informix Primary Storage Manager
Related reference:
Informix Primary Storage Manager configuration parameters
The onpsm utility for storage management
The onsmsync utility

Example 1: Storing backups for an instance

This example shows how to set up and use Informix Primary Storage Manager to back up the data and logical logs for a single database server instance to a directory: $INFORMIXDIR/backups.

In this example, you update the configuration file so that the Informix Primary Storage Manager can communicate with ON-Bar and you specify the directory where you want backups stored. Then you use the onbar utility to perform a standard, level-0 backup of all online storage spaces and used logical logs. You validate the backup by checking the messages that were logged and by using the onpsm utility to confirm that storage objects were created.

1. Set the BAR_BSALIB_PATH configuration parameter to the full path and name of the shared library for the storage manager.
   For example, on Linux, Solaris:

   ```
   BAR_BSALIB_PATH $INFORMIXDIR/lib/libbsapsm.so
   ```

   You must use the version of the XBSA shared library that is provided for Informix Primary Storage Manager. If you do not specify the path with the BAR_BSALIB_PATH configuration parameter, you must ensure that the XBSA library is in the default location on your operating system.

2. If needed, create the directory in which to store the backup objects.
   By default, the storage manger includes the default pools LOGPOOL and DBSPOOL, with the default directory $INFORMIXDIR/backups in each pool.

   - If you want to use the default backup directory, verify that the $INFORMIXDIR/backups directory exists.
   - If you want to use a different backup directory, use the **onpsm -D add** command to add a new backup directory for LOGPOOL and DBSPOOL. For example, run the following commands to add different backup directories for the LOGPOOL and DBSPOOL pools:

     ```
     onpsm -D add /backups/infx/logs -g LOGPOOL -p HIGHEST -t FILE
     onpsm -D add /backups/infx/spaces -g DBSPOOL -p HIGHEST -t FILE
     ```

     Use the HIGHEST priority for the device that should be filled first. Only one device in a pool can have the priority setting of HIGHEST.

3. Run the onbar utility to perform a standard, level-0 backup of all online storage spaces and used logical logs.

   ```
   onbar -b -L 0
   ```

   If the storage catalog tables do not exist, they are created in the $INFORMIXDIR/etc/psm directory.
4. Validate that the storage manager is set up and that the backup objects are created.
   a. Look in the ON-Bar activity log to confirm that the storage manager is ready and that ON-Bar recognizes the storage manager.
      For example, the first message is from the storage manager and the second message is from the backup utility:

      ```
      2012-01-03 15:51:23 11193  2569 Informix PSM is ready.
      2012-01-03 15:51:23 11193  2569 Using Informix PSM version 14.10.FC1
      as the Storage Manager. XBSA API version is 1.0.3.
      ```

      By default, the storage manager posts messages to the ON-Bar activity log. The location of the activity log is set by the BAR_ACT_LOG configuration parameter. If you want the storage manager messages to be logged separately, you must set the PSM_ACT_LOG configuration parameter.
   b. Run the **onpsm -O list** command to list the storage objects that were created:
      The list, as shown in the following example, includes the storage object IDs, the date the storage objects were created, the size of the storage objects, and where the storage objects are in the storage device. The object IDs are also stored in the ixbar file and are used by ON-Bar to locate the objects.

      ```
      ============================
      Object List Report

                                          Logical Path
      Obj ID  Date Created        Size (MB)    Name.Version (omits piece #)
      -------
           1  2012-08-06 12:02:10      12.5  /serv1/rootdbs/0/serv1.1
           2  2012-08-06 12:02:12       0.1  /serv1/logdbs/0/serv1.1
           3  2012-08-06 12:02:12       0.1  /serv1/dbs2/0/serv1.1
           4  2012-08-06 12:02:12       0.1  /serv1/dbs1/0/serv1.1
           5  2012-08-06 12:02:13       0.1  /serv1/physdbs/0/serv1.1
           6  2012-08-06 12:02:14       0.3  /serv1/10/9/serv1.1
           7  2012-08-06 12:02:14       0.0  /serv1/crit_files/ixbar/serv1.1
           8  2012-08-06 12:02:14       0.0  /serv1/crit_files/oncfg/serv1.1
           9  2012-08-06 12:02:14       0.1  /serv1/crit_files/onconfig/serv1.1
          10  2012-08-06 12:02:14       0.0  /serv1/crit_files/sqlhosts/serv1.1
      ============================
      ```

   c. Run the **onpsm -D list** command to display a list that shows that the device was added to the DBSPOOL and LOGPOOL pools. The following example shows output of the command:

      ```
      Type  Prio     Block/Size (MB)  Pool Name   Device Name---
      FILE   HIGHEST       --/--         DBSPOOL     /backups/infx/logs

      FILE   HIGHEST       --/--         LOGPOOL     /backups/infx/spaces
      ```

With a few simple steps, you configured the storage manager and performed a full backup of an instance to a file device. Very little configuration was required because the storage manager uses the default settings for various ON-Bar configuration parameters.

Storage catalog tables are not included in a backup. Be sure to back up the storage catalog tables with your operating system tools as part of a disaster recovery strategy. If the storage catalog tables are lost, the **onbar** utility cannot restore the backup objects that Informix Primary Storage Manager created. The location of the storage

catalog tables is set by the PSM_CATALOG_PATH configuration parameter (default = $INFORMIXDIR/etc/psm).

To restore the instance from the backup objects, use the onbar utility. The storage manager tracks the backup objects and storage devices for you.

Example 2: Storing backups for two instances

This example shows how to configure one instance of Informix Primary Storage Manager to manage the storage devices for two database server instances in a multiple residency environment.

In this example, you set up two independent database server environments on the same computer. Each database server is installed in a separate directory: (/usr/informix/ids1210fc1 and /usr/informix/ids1210fc1b) and has a database server instance. Storage for backup operations on both database server instances is managed by one instance of Informix Primary Storage Manager. Pools of storage devices for physical and logical data are configured for each instance.

1. For *each* instance, edit the **onconfig** file to configure storage management for ON-Bar.

Table 1. Configuration parameters and their associated values

Configuration parameter	Value
BAR_BSALIB_PATH Specify the full path and name of the shared library for the storage manager.	/usr/informix/ids1210fc1b/lib/libbsapsm.so
PSM_CATALOG_PATH Specify the path of the storage catalog tables.	/usr/informix/ids1210fc1b/etc/psm
PSM_DBS_POOL Specify the name for a group of devices for storing online data (dbspace) backups.	FC1: DBSPOOL_FC1 FC1B: DBSPOOL_FC1B
PSM_LOG_POOL Specify the name for a group of devices for storing online logical log backups.	FC1: LOGPOOL_FC1 FC1B: LOGPOOL_FC1B

2. For *each* instance, create a directory in which to store the backup objects.

```
mkdir $INFORMIXDIR/backups/dev_for_1201fc1
mkdir $INFORMIXDIR/backups/dev_for_1201fc1b
```

3. Run the onpsm utility to create device pools for each instance. For example, specify:

```
onpsm -P add DBSPOOL_FC1
onpsm -P add LOGPOOL_FC1
onpsm -P add DBSPOOL_FC1B
onpsm -P add LOGPOOL_FC1B
```

4. Run the **onpsm** utility to add the storage devices.

```
onpsm -D add $INFORMIXDIR/backups/dev_for_1201fc1 -t FILE -g DBSPOOL_FC1
onpsm -D add $INFORMIXDIR/backups/dev_for_1201fc1 -t FILE -g LOGPOOL_FC1
onpsm -D add $INFORMIXDIR/backups/dev_for_1201fc1b -t FILE -g DBSPOOL_FC1B
onpsm -D add $INFORMIXDIR/backups/dev_for_1201fc1b -t FILE -g LOGPOOL_FC1B
```

5. For *each* instance, run the onbar utility to perform a standard, level-0 backup of all online storage spaces and used logical logs.

```
onbar -b -L 0
```

6. Validate that the storage manager is set up and that the backup objects are created.
   a. For *each* instance, look in the ON-Bar activity log to confirm that the storage manager is ready and that ON-Bar recognizes the storage manager. For example, look for this information:

```
2012-01-03 15:51:23 11193  2569 Informix PSM is ready.
2012-01-03 15:51:23 11193  2569 Using Informix PSM version 14.10.FC1
as the Storage Manager. XBSA API version is 1.0.3.
```

   b. Use the onpsm utility to list the storage objects that were created:

```
onpsm -O list
```

   The report includes the storage object IDs, the date the storage objects were created, the size of the storage objects, and the location of the storage objects in the storage device.

Example 3: Exporting backups to and restoring them from another directory

This example shows how to export backups to a new directory and import the backup objects from that directory.

Suppose that you keep five generations of backups. As an added precaution, you also keep copies of the most recent backups in a separate directory. In this example, you use the **onsmsync** utility to export your most recent backup to and import it from the Informix Primary Storage Manager external pool in a separate directory.

The storage manager tracks devices in the external device pool (EXTPOOL) so it can copy objects to and from external devices. (Although the storage manager tracks devices, it does not track files and objects that are inside the EXTPOOL pool in the storage manager catalogs.)

1. Store backups for an instance, following the steps in Example 1: Storing backups for an instance.
2. Run the **onpsm -D list** command to check that there is a device in the EXTPOOL pool.
   a. If there is no device in the EXTPOOL pool, add one using the **onpsm -D add** command.
   The following example shows how to add a device with the path /export/informix/psm_exportdir to the EXTPOOL pool.

```
$ onpsm -D add /export/informix/psm_exportdir -g EXTPOOL -t FILE
```

3. Run the **onsmsync** command to export all backup objects in the generation 1 level-0 backup, using prefix `pw_sept5`, which becomes the name of the subdirectory in which the utility places the backup:

   ```
   onsmsync -E -p pw_sept5 -g 1
   ```

   After you run the **onsmsync -E** command to export the backup objects, you will see a subdirectory in the EXTPOOL directory that includes a directory holding the backup objects and a file called export.bom.

Suppose that something happens to the backup generation stored in your primary backup directory and you want to import the `pw_sept5` backup generation from the second directory. To import the backup generation:

1. Run the **onsmsync** command to import all backup objects in the `pw_sept5` subdirectory:

   ```
   onsmsync -I -p pw_sept5
   ```

Use your own file-transfer methods to move the exported backups, as needed, to other machines.

Example 4: Exporting a backup from one server and importing it into another server

This example shows how to use the **onsmsync** utility to export a backup from a database server that has the name `informix_serv1`. Then the example shows how to use the **onsmsync** utility to import the data into a server that has the name `informix_serv2`.

1. Set up and export files on database server `informix_serv1`:
   a. Set the INFORMIXDIR, INFORMIXSERVER, ONCONFIG, PATH, INFORMIXSQLHOSTS environment variables for `informix_serv1`.
   b. Run the **onpsm -D list** command to check that there is a device in the EXTPOOL pool. If there is no device in the EXTPOOL pool, add one using the **onpsm -D add** command.
   c. Run the **onsmsync** command to export all backup objects in the generation 1 level-0 backup, using prefix `serv1_20120810`, which becomes the name of the subdirectory in which the utility places the backup:

      ```
      $ onsmsync -E -p serv1_20120810 -g 1
      ```

2. Prepare to import files on the second database server, `informix_serv2`, as follows:
   a. Set the INFORMIXDIR, INFORMIXSERVER, ONCONFIG, PATH, INFORMIXSQLHOSTS environment variables for `informix_serv2`.
   b. Run the **onpsm -D list** command to determine if the EXTPOOL has the same device that you viewed or added in step 1b. (This could occur for shared devices). If there is no device in the EXTPOOL pool, add one using the **onpsm -D add** command.
   c. Copy the previously exported backup objects (for example, subdirectory `serv1_20120810`) into the EXTPOOL device from which you will import the backup objects.
   d. Run the following command to import backup objects from EXTPOOL device:

      ```
      $ onsmsync -I -p serv1_20120810
      ```

      After you run the **onsmsync -I** command to import the backup objects, the objects are stored in the new LOGPOOL and DBSPOOL pools.
   e. Run the **onpsm -O list** command to view the imported objects. Notice that the import command also creates a new ixbar file in $INFORMIXDIR/etc/ directory.

      ```
      $ ls -l $INFORMIXDIR/etc/*ixbar*

      -rw-rw-- 1 informix informix     0 Aug 10 19:44
       /usr/informix/etc/ixbar.12.20120810.194441
      -rw-rw-- 1 informix informix  2704 Aug 10 19:44
       /usr/informix/etc/ixbar.12
      ```

      The new ixbar file lists the imported backup objects so that you can perform an ON-BAR cold restore to restore the `informix_serv1` instance from the first database server to the `informix_serv2` instance on the second database server.

Setting up Informix Primary Storage Manager

Setting up involves gathering and specifying information about your storage devices and, if necessary, changing the default configuration of the storage manager.

- Collecting information about file directories and devices
  You must gather information about and configure at least one file directory or device for each of the DBSPOOL and LOGPOOL pools before ON-Bar can use the IBM Informix Primary Storage Manager.
- Configuring Informix Primary Storage Manager
  By default, the IBM Informix Primary Storage Manager is automatically configured with the information specified in the storage manager and with some ON-Bar configuration parameters. It is also automatically configured when you use the **onpsm** utility. You can change the configuration.

Related reference:
Configure a storage manager
The onpsm utility for storage management
Informix Primary Storage Manager configuration parameters

Collecting information about file directories and devices

You must gather information about and configure at least one file directory or device for each of the DBSPOOL and LOGPOOL pools before ON-Bar can use the IBM® Informix® Primary Storage Manager.

Before defining directories or devices, gather the following information:

- The full path names and types of the devices that you plan to use for your backup storage.
- The amount of space that you want to commit to ON-Bar backups.

Configuring Informix Primary Storage Manager

By default, the IBM® Informix® Primary Storage Manager is automatically configured with the information specified in the storage manager and with some ON-Bar configuration parameters. It is also automatically configured when you use the **onpsm** utility. You can change the configuration.

The Informix Primary Storage Manager uses file devices (disks) only, not tapes. You cannot configure the storage manager to use tapes.

To manually configure the Informix Primary Storage Manager:

1. Update the BAR_BSALIB_PATH configuration parameter to point to the storage manager library.
   For example, on Linux or Solaris, specify:

   ```
   BAR_BSALIB_PATH $INFORMIXDIR/lib/libbsapsm.so
   ```

2. Specify the destination and source devices for backup and restore operations by using the **onpsm** utility.
3. Change the default configuration for the storage manager if necessary for your environment:
   a. To override the default values for the location of storage manager log files and catalogs, debugging activity, and pool names, specify new values in the Informix Primary Storage Manager configuration parameters.
   b. To specify a larger transfer buffer with ON-Bar and the Informix Primary Storage Manager, increase the size in the BAR_XFER_BUF_SIZE configuration parameter.
   c. To change the frequency of the progress messages in the ON-Bar activity log, update the value specified in the BAR_PROGRESS_FREQ configuration parameter.
   d. To change the number of processes that ON-Bar runs concurrently, update the value specified in the BAR_MAX_BACKUP configuration parameter.

Related reference:
Informix Primary Storage Manager configuration parameters
ON-Bar and ontape configuration parameters and environment variable
The onpsm utility for storage management

Managing storage devices

Use the **onpsm** utility to add, monitor, and remove storage devices and to manage IBM® Informix® Primary Storage Manager catalogs, locks, and objects. Use the **onsmsync** utility to export ON-Bar backups to and import them from external pools and to expire backups.

Related reference:
The onpsm utility for storage management
The onsmsync utility

The onpsm utility for storage management

Use the **onpsm** utility to manage the IBM® Informix® Primary Storage Manager catalogs, devices, locks, and objects.

Pre-requisite: To run the **onpsm** utility, you must be user **root** or **informix**.

Syntax

```
>>-onpsm--+-+-| Catalog options |-+-+--------------------------><
          | +-| Device options |--+ |
          | +-| Object options |--+ |
          | '-| Pool options |----' |
          +--- -h ------------------+
          '-+----------------+-------'
            +- -V ----------+
            +- -version-----+
            '- -version all-'

Catalog options

|-- -C -+-check--+-----+-----------+---------------------------|
        |        +- -l +           |
        |        '- -n '           |
        +-+---------+--------------+
        | '- detail-'              |
        +-+---------+--+----------+-+
        | '- export-'  +- import + |
        |              '- -y ----' |
        +-+----------+-------------+
        | +- init ---+             |
        | +- init -d +             |
        | '- -y -----'             |
        '-+---------+--------------'
          '- unlock '

Device options

|-- -D -+-add--path-- -p -priority-- -g -pool_name-- -t -type--+----------+-- -b -block_size-+--|
        |                                                      '- -s -size-'                  |
        +-del--path--+- -g -pool_name--+-----+-+-----------------------------------------------+
```

```
        |           |               '- -y ' |                                                  |
        |           '- -d -+-----+-----------'                                                  |
        |                  '- -y '                                                             |
        +-list--+- -u ----------------+-------------------------------------------------------+
        |       '- -l -dev_def_file.txt-'                                                      |
        +-purge--path---------------------------------------------------------------------------+
        +-scan--path----------------------------------------------------------------------------+
        '-update--path--+---------------+--+-----------+-------------------------------------'
                        '- -p -priority-'  '- -s -size-'

Object options

|-- -O -+-del-- -o -object_id--+-----+-+-----------------------|
        |                      '- -y ' |
        +-detail-- -o -object_id-------+
        +-dump-- -o -object_id---------+
        '-list-------------------------'

Pool options

|-- P -+-add--pool_name----------+------------------------------|
       +-del--pool_name--+-----+-+
       |                 '- -y ' |
       '-list--------------------'
```

Table 1. **onpsm** utility catalog options

Element	Purpose	Key Considerations
-C check	Checks storage manager catalog tables, which store metadata about the pools and devices that the storage manager manages	This command identifies files that have problems.
-C check -l	Displays index keys while checking the catalog tables	
-C check -n	Indicates that the storage manager does not fix errors that are found	
-C detail	Shows details about the storage manager catalog tables	
-C export	Exports the Informix Primary Storage Manager catalog tables to a directory called psm_catalog.exp	
-C import	Replaces the current Informix Primary Storage Manager catalog with a catalog that is recreated from files that are in the psm_catalog.exp directory	Only import the catalog if you have a system problem, lost your current catalog, and need to revert to the exported catalog. If you need to import a catalog, run the **onpsm -C init** command before you run the **onpsm -C import** command.
-C init	Deletes storage manager catalog tables	
-C init -d	Deletes storage manager catalog table and the backup objects in file devices	
-C unlock	Unlocks the storage manager catalog	If the storage manager exits abnormally from a backup or restore session because a failure occurred, storage manager catalog tables might remain locked. If catalog tables are locked, you can release the locks.
-y	Specifies to not to ask for confirmation before deleting catalog tables	

Table 2. **onpsm** utility device options

Element	Purpose	Key Considerations
-D add	Adds a device to the pool specified with the **-g** option	Before adding devices, gather information about the device. See Collecting information about file directories and devices.
-D del	Removes a device: • If you use the **-g** option, removes a device from the pool specified with the **-g** option, while retaining the device objects in the Informix Primary Storage Manager catalog. • If you use the **-d** option, removes the device from all pools and removes all backup objects from the file system in that device	If you delete the device using **-g** option, you can restore the objects if necessary. If you remove a device, the storage manager cannot add new objects to the device.
-D list	Displays a list of all devices in the system	
-D purge	Removes missing storage manager objects from the Informix Primary Storage Manager catalog	

Element	Purpose	Key Considerations
-D scan	Scans objects in the device to verify that the objects exist in the Informix Primary Storage Manager catalog so the objects can be restored if necessary If an object is not in the catalog, this command adds the object to the catalog.	If the command cannot add an object to the catalog, the command ignores the missing file. Before a missing object can be added to a catalog, the following conditions must occur: • The object ID must not be assigned to any other object in the storage manager. • Files must not be renamed or relocated to different directories inside the device • The object version must not be assigned to any other object in the storage manager.
-D update	Modifies information about a device	If you want to modify information about more than one device, run a separate command for each device.
path	Full name and path to the device (for TAPE devices) or to a directory (for FILE devices)	The path must be in the format appropriate to the operating system to which the device is attached . The name of the device must be unique within a pool. You can include the same device in multiple pools. If you are deleting, listing, purging, scanning, or updating information, the path must be to an existing device.
-b *block_size*	For tape devices only, the minimum number of bytes of data that need to accrue before the data is written to the device	The block size is required for tape devices.
-d	Deletes the pool from all pools and deletes backup objects	The block size is required for tape devices.
-g *pool_name*	The pool in which to add the device, either DBSPOOL, LOG POOL, or EXTPOOL	Information about pools is stored in the Informix Primary Storage Manager catalog. If you do not provide a pool name, the command fails. Specify: • DBSPOOL for backups of dbspaces, blobspaces, and sbspaces • LOGPOOL for backups of logical logs • EXTPOOL that serves as a staging area from which you can move specific backups or backup generations to permanent storage or onto a different computer.
-l *dev_def_file.txt*	Loads information about the device from a device-definition file	
-p *priority*	Priority of the device, either HIGHEST, HIGH, LOW, or READ-ONLY	The storage manager fills high-priority devices in a pool before placing data into low-priority devices in that pool. If the high-priority devices are busy at the moment when the storage manager is ready to fill a pool, the storage manager uses low priority devices. Only one device in a pool can have the priority of HIGHEST. If multiple devices have the same priority in the same pool, the storage manager determines which device to use first. When a device becomes full, the storage manager changes the priority to READ-ONLY. You can change the priority after you add more space to the device.
-s *size*	For tape devices only, the maximum storage capacity of the device in kilobytes	The size is optional for tape devices. If a size is not specified, or if you specify `0`, the storage manager interprets the size as unlimited. When the size is unlimited, the device is not considered full until it returns an error that specifies that the device is full. To specify the size enter the numeric value of the size followed by the suffix B, K, M, G, T, or P (for bytes, kilobytes, megabytes, gigabytes, terabytes, or petabytes). The suffix can be upper or lowercase.
- t *type*	Type of device, either FILE or TAPE	Information about devices is stored in the Informix Primary Storage Manager catalog,
-u	Unloads information about the device to a device-definition file	The device definition file is a text file with a specific format. The storage manager uses the file to recreate the information when you run an **onpsm** command with the load option.
-y	Specifies not to ask for confirmation to complete the requested action	

Table 3. **onpsm** object options

Element	Purpose	Key Considerations
-O del	Deletes physical objects from a pool	
-O detail	Displays details about the specified object. Details include the location of the object.	
-O dump	Extracts the object data to a file in the current directory	

Element	Purpose	Key Considerations
- o *object_id*	Identifies the particular object	You can delete or dump one or more objects with a single command, as shown in Usage.
-O list	Displays all objects in a pool	For each object, the list includes the date and time the object was created, the size of the object, and the path name of the object.
-y	Specifies not to ask for confirmation to complete the requested action.	

Table 4. **onpsm** pool options

Element	Purpose	Key Considerations
-P add *pool_name*	Adds a new pool	
-P del *pool_name*	Deletes the specified pool	
-P list	Lists all pools in the system	
-y	Specifies not to ask for confirmation to complete the requested action.	

Table 5. **onpsm** utility general options

Element	Purpose	Key Considerations
-h	Displays help information	
-V	Displays the software version number and the serial number	For more details about the standard **-V** and **-version** options, see Obtaining utility version information.
-version	Displays the software version number, serial number, and additional information such as the host, operating system, build date, and the Global Language Support (GLS) version	For more details about the standard **-V** and **-version** options, see Obtaining utility version information.
-version all	Displays **onpsm** version information and information about the PSM shared library	

Usage

When you run an **onpsm** command to define a device, the storage manager automatically creates storage manager catalogs if they do not exist.

The default device for the storage manager is $INFORMIXDIR/backups. The device, which is low priority, is automatically created when the catalog is created. You can remove the default device.

When you create a device, the storage manager automatically creates the directory for the device if the directory does not exist. The storage manager uses the directory path that you specify in the **onpsm -D add** command.

You can delete one or more objects with a single command, for example, by running a command that has this format:

```
onpsm _O del -o obj_1 -o obj_2
```

You can also dump one or more objects with a single command, for example, by running a command that has this format:

```
onpsm _O dump -o obj_1 -o obj_2
```

If the data is not needed, run the **onsmsync** utility to delete backup objects from the Informix Primary Storage Manager.

Some third-party storage managers do not allow the **onsmsync** utility to delete backup objects from the storage manager. If you have a third-party storage manager, you might need to manually delete backup objects that you no longer need.

Examples

The following command adds a file device with the path name `$INFORMIXDIR/backups` in the DBSPOOL pool:

```
onpsm -D add $INFORMIXDIR/backups -g DBSPOOL -t FILE  -p HIGH
```

The following command checks Informix Primary Storage Manager catalog tables and indicates that the storage manager does not fix any errors found during the check:

```
onpsm -C check -n
```

The following command lists objects in pools, including the date and time the object was created, the size of the object, and the path name of the object.

```
onpsm -O list
```

- onpsm -C detail output
  Use the **onpsm -C detail** command to view details about the storage manager catalog tables.
- onpsm -D list output
  The **onpsm -D list** command displays information about all of the devices in each IBM Informix Primary Storage Manager pool. You can use this list to determine whether you need to change information about your devices.
- onpsm -O list output
  The **onpsm -O list** command displays all of the objects stored in a pool.

Related concepts:
IBM Informix Primary Storage Manager
Setting up Informix Primary Storage Manager
Managing storage devices

Device pools
Related tasks:
Configuring Informix Primary Storage Manager
Examples: Manage storage devices with Informix Primary Storage Manager

onpsm -C detail output

Use the **onpsm -C detail** command to view details about the storage manager catalog tables.

Sample onpsm -C detail command output

```
D:\IFMXDATA\gacpsm>onpsm -C detail

Informix Primary Storage Manager State:

    PSM Unique ID     : 1358735848
    Catalog Location  : D:\database\1210\etc\psm\
    Catalog State     : Locked
    Catalog Owner     : 2
    Catalog Lock Mode: Regular

Sessions:
    Session ID        Process ID
    2                 1576
    3                 4556
    4                 5555

Informix PSM Locked Objects

Session  Object Id Date       Time     Server         Object Name

     3         116 2012-12-09 20:54:34 /gacpsm_tcp    /gacpsm_tcp/168/242
     4         117 2012-12-09 21:07:44 /gacpsm_tcp    /gacpsm_tcp/168/242
```

The output contains the following sections:

Informix® Primary Storage Manager State
: Shows general information about all of the sessions that are active in the system and whether the catalog is locked.

  PSM Unique ID
  : ID for the catalog
  Catalog Location
  : Path to the catalog
  Catalog State
  : Indicates whether the catalog is locked
  Catalog Owner
  : Informix Primary Storage Manager session ID
  Catalog Lock Mode
  : Lock category
  For example, `Regular` means that the lock is a user lock.

Sessions
: Lists all of the sessions that are active in the system and the process IDs that match the sessions.

  Session ID
  : ID of the session. This is the same ID that appears in the `Catalog Owner` field.
  Process ID
  : Internal ID for the ON-Bar, **archecker**, or storage manager process that is locking the catalog.

Informix PSM Locked Objects
: Shows the locks in devices or objects that are held by storage manager sessions. For each lock, the output shows the session number, the object ID, the date and time that the lock was placed, the server, and the object name.

onpsm -D list output

The **onpsm -D list** command displays information about all of the devices in each IBM® Informix® Primary Storage Manager pool. You can use this list to determine whether you need to change information about your devices.

Sample onpsm -D list command output

```
Type   Prio     Block/Size (MB)  Pool Name     Device Name
-----  -------  ---------------  ------------  ----------------------------
FILE    LOW          --/--        DBSPOOL      /informix/backups

FILE    LOW          --/--        LOGPOOL      /informix/backups
```

Type
: Type of device, either FILE or TAPE (Currently only the FILE type is supported.)

Prio
: Priority of the device, either HIGH, HIGHEST, LOW, or READ-ONLY
HIGH is the default priority if a priority is not specified. Only one device in a pool can have a priority of HIGHEST.

Block Size
: Size of the device (only applies to devices of the TAPE type)

Pool Name
: The name of the pool (DBSPOOL, LOGPOOL, or EXTPOOL)
In the example output above, there are no EXTPOOL devices.

Device Name
: Complete path name for the device

onpsm -O list output

The **onpsm -O list** command displays all of the objects stored in a pool.

Sample onpsm -O list command output

```
Object List Report
                                        Logical Path
Obj ID  Date Created         Size (MB)    Name.Version (omits piece #)
------  -------------------  ---------  ----------------------------------
     1  2012-07-06 14:39:47       12.0  /gacpsm_tcp/rootdbs/0/gacpsm_tcp.1
     2  2012-07-06 14:41:18       12.0  /gacpsm_tcp/rootdbs/0/gacpsm_tcp.2
     3  2012-07-11 13:42:10        3.9  /gacpsm_tcp/160/14/gacpsm_tcp.1
     4  2012-07-11 13:42:13        3.9  /gacpsm_tcp/160/15/gacpsm_tcp.1
     5  2012-07-11 13:42:15        3.9  /gacpsm_tcp/160/16/gacpsm_tcp.1
     6  2012-07-11 13:42:15        3.9  /gacpsm_tcp/160/17/gacpsm_tcp.1
     7  2012-07-11 13:42:15        3.9  /gacpsm_tcp/160/18/gacpsm_tcp.1
     8  2012-07-11 13:42:15        3.9  /gacpsm_tcp/160/19/gacpsm_tcp.1
     9  2012-07-11 13:42:16        3.9  /gacpsm_tcp/160/20/gacpsm_tcp.1
    10  2012-07-11 13:42:16        3.9  /gacpsm_tcp/160/21/gacpsm_tcp.1
```

Obj ID
: The ID of the stored object

Date Created
: The date and time when the object was created

Size
: The size of the object in megabytes

Name.Version
: The path name of the object followed by `.` and the version of the object (for example, `.2` to indication version 2)

Device pools

The IBM® Informix® Primary Storage Manager pool is a named group of disk devices that you use as a repository for backups.

When storing backup objects, the Informix Primary Storage Manager selects particular devices from the pool and moves automatically from one device to another when devices are full or when they fail. You maintain pools by using the **onpsm** utility to add, modify, view, and drop devices in the pool.

Three pools are available:

DBSPOOL
: Holds online backups of dbspaces, blobspaces, and sbspaces

LOGPOOL
: Holds online backups of logical logs

EXTPOOL
: Serves as a staging area for exporting a backup set of objects to a single large, external logical object or for importing the backed up objects. You can move specific backups or backups generations in this pool to permanent storage or onto a different computer. Files in the EXTERNAL pool are offline. The files are not visible to ON-Bar, and the Informix Primary Storage Manager does not track them.

Related concepts:
IBM Informix Primary Storage Manager
Informix Primary Storage Manager file-naming conventions
Related reference:
PSM_DBS_POOL configuration parameter
PSM_LOG_POOL configuration parameter
The onpsm utility for storage management

Device-configuration file for the Informix Primary Storage Manager

The **onpsm** utility can generate a device-configuration file, which is a text file that contains information about a storage device. The utility uses this information to re-create the devices.

The configuration file contains the following information about each device:

DEVICE
: The complete path to the device

TYPE
: Type of device

  - FILE = file directory on a disk device

POOL
: The pool that contains the device, either DBSPOOL, LOGPOOL, or EXTPOOL

BLOCKSIZE
: Not applicable for disk devices.

SIZE
: Not applicable for disk devices.

PRIORITY
: The priority of the device

For example, the device-configuration file might contain the following information

```
DEVICE=/vobs/tristarm/sqldist/psm_backup/dbspace
TYPE=FILE
POOL=DBSPOOL
BLOCKSIZE=0
SIZE=0
PRIORITY=HIGH
```

Informix Primary Storage Manager file-naming conventions

When creating files that store your backup data, the IBM® Informix® Primary Storage Manager uses specific file-naming conventions.

For the DBSPOOL and LOGPOOL device pools, the path name of the storage file consists of:

1. Category information:
   - For space backups, the category consists of the device name, the database server name, and the space name
   - For log backups, the category consists of the device name, the database server name, server number, and a log file number
2. For space backups, the backup level, either: 0, 1, or 2
3. A version number, which is an integer that starts at 1 for an object of that category and is incremented for each subsequent backup of an object of that category.
4. An ID that identifies a piece of the backed up object

For space backups, file names have this format:

/device/DBSERVERNAME/dbspace/backup_level/DBSERVERNAME.version.piece

For example, the name of the file for the third piece of a second level 0 backup of the dbspace `rootdbs` for the server named `SERVER1` is:

```
/my_device/SERVER1/rootdbs/0/SERVER1.2.3
```

For log backups, file names have this format:

/device/DBSERVERNAME/SERVERNUM/LOG_UNIQUE_ID/DBSERVERNAME.version.piece

If you use the **onsmsync** utility with the -E option to export a backup generation in the Informix Primary Storage Manager, the **onsmsync** utility creates and places the backup files in a subdirectory of the storage manager EXTPOOL device. You must provide a *prefix* (the name of a subdirectory) when you use the **onsmsync** utility with the -E or -I options. ON-Bar uses the specified path as a key to communication with the storage manager to store and retrieve objects.

Related concepts:
Device pools
Related reference:
The onsmsync utility

Message logs for Informix Primary Storage Manager

The Informix® Primary Storage Manager writes messages to a storage manager activity log and debug log.

The message logs are stored in the directories specified in the BAR_DEBUG_LOG or BAR_ACT_LOG configuration parameters. You can use the PSM_ACT_LOG and PSM_DEBUG_LOG configuration parameters to specify another directory for each of these logs.

The PSM_DEBUG configuration parameter specifies the level of debugging activity that is captured in the debug log.

Related reference:
PSM_ACT_LOG configuration parameter
PSM_DEBUG_LOG configuration parameter
PSM_DEBUG configuration parameter

archecker table level restore utility

- archecker table level restore utility
  You can use the **archecker** utility to perform point-in-time table-level restores that extract tables or portion of tables from archives and logical logs.

archecker table level restore utility

You can use the **archecker** utility to perform point-in-time table-level restores that extract tables or portion of tables from archives and logical logs.

The **archecker** utility restores tables by specifying the source table to be extracted, the destination table where the data is placed, and an INSERT statement that links the two tables.

For information about using the **archecker** utility to verify backups, see onbar -v syntax: Verifying backups.

- Overview of the archecker utility
  The **archecker** utility is useful where portions of a database, a table, a portion of a table, or a set of tables need to be recovered. It is also useful in situations where tables need to be moved across server versions or platforms.
- Data restore with archecker
  Use the **archecker** utility to perform to types of restore operations.
- Syntax for archecker utility commands
  The **archecker** utility provides a command-line interface for restoring data from an archive. To use **archecker**, you must specify both a configuration file and a schema command file.
- The archecker schema reference
  The topics in this section describe the SQL-like statements used by the **archecker** schema command file. This file provides information that the **archecker** utility uses to perform data recovery.

Related reference:
The archecker utility configuration parameters and environment variable

Overview of the archecker utility

The **archecker** utility is useful where portions of a database, a table, a portion of a table, or a set of tables need to be recovered. It is also useful in situations where tables need to be moved across server versions or platforms.

Use **archecker** utility in the following situations:

- Restore data
  You can use the **archecker** utility to restore a specific table or set of tables that have previously been backed up with ON-Bar or **ontape**. These tables can be restored to a specific point in time. This is useful, for example, to restore a table that has accidentally been dropped.

  The following restrictions, however, limit the functionality of the **archecker** utility in some table-level restore operations:
  - The **archecker** utility is not JSON compatible. If you try to use the utility with target tables that contain columns of JSON or BSON (binary JSON) data types, the utility will abort and return an error message. (Besides data-restore contexts, this limitation affects all **archecker** operations on tables with JSON or BSON columns.)
  - You cannot logically restore a table when restoring smart large objects. Only a physical restore of BLOB or CLOB objects is supported for table-level restore operations using the **archecker** utility.
  - You cannot restore data from a remote device.
  - You cannot use a shared memory connection when performing a table-level restore.
- Copy data
  The **archecker** utility can also be used as a method of copying data. For example, you can move a table from the production system to another system.

  The **archecker** utility is more efficient than other mechanisms for copying data. Because **archecker** extracts data as text, it can copy data between platforms or server versions.

- Migrate data
  You can also use the **archecker** utility as a migration tool to move a table to other IBM® Informix® servers.

The **archecker** utility is designed to recover specific tables or sets of tables. Other situations require that you use different utilities. For example, use ON-Bar or **ontape** in the following data recovery scenarios:

- Full system restore
- Recovery from disk failure

To configure the behavior of the **archecker** utility, use the **archecker** configuration file. To define the schema of the data that **archecker** recovers, use the **archecker** schema command file. These files are described in the following sections.

- The archecker configuration file
  The **archecker** utility uses a configuration file to set certain parameters.
- Schema command file
- Table-level restore and locales

Related reference:
ON-Bar script

The archecker configuration file

The **archecker** utility uses a configuration file to set certain parameters.

Set the AC_CONFIG environment variable to the full path name of the **archecker** configuration file. By default, the AC_CONFIG environment variable is set to $INFORMIXDIR/etc/ac_config.std. If you set AC_CONFIG to a user-defined file, you must specify the entire path including the file name.

For information about the configuration parameters used in this file, see The archecker utility configuration parameters and environment variable.

Schema command file

The **archecker** utility uses a schema command file to specify the following:

- Source tables
- Destination tables
- Table schemas
- Databases
- External tables
- Point in time the table is restored to
- Other options

This file uses an SQL-like language to provide information **archecker** uses to perform data recovery. For complete information about the supported statements and syntax, see The archecker schema reference.

There are two methods to set the schema command file:

- Set the AC_SCHEMA configuration parameter in the **archecker** configuration file. For more information, see AC_SCHEMA configuration parameter.
- Use the -f *cmdname* command-line option. For more information, see Syntax for archecker utility commands.

If both methods are specified, the -f command-line option takes precedence.

Table-level restore and locales

For table-level restore, if the table being restored (table on the archive) has a locale code set different from the default locale (en_US.8859-1) the DB_LOCALE environment variable must be set to have the same code set as the locale of the archived table being restored.

No code set conversion is performed during a table-level restore; the locale code set of the database or table being restored must match the locale code set of the database or table that the data is being restored to. In addition, the same DB_LOCALE information is used for all of the tables being restored by using the same table-level restore command schema file.

Data restore with archecker

Use the **archecker** utility to perform to types of restore operations.

The two types of restores that the **archecker** utility performs are:

- A physical restore that is based on a level-0 archive.
- A physical restore followed by a logical restore, which uses both a level-0 archive and logical logs to restore data to a specific point in time.

When reading the command file, **archecker** determines whether to perform a physical restore only or a physical restore followed by a logical restore. By default, **archecker** performs a physical and logical restore. If you use the WITH NO LOG clause, **archecker** does not perform a logical restore.

The procedures and resources that **archecker** uses differ between a physical-only restore and a physical and logical restore. These procedures are outlined in the following sections.

- Physical restore
  When the **archecker** utility performs a physical restore, the utility extracts data from a level-0 archive.
- Logical restore
  After a physical restore, logical recovery can further restore tables to a user-specified point in time. To do this, the **archecker** utility reads backed-up logical logs, converts them to SQL statements, and then replays these statements to restore data.

Physical restore

When the **archecker** utility performs a physical restore, the utility extracts data from a level-0 archive.

When performing a physical restore, **archecker** performs the following tasks:

- Disables all constraints (including foreign constraints that reference the target table), indexes, and triggers until the data is restored. Restore performance is better if the table has no constraints, indexes, or triggers.
- Reads the schema command file to determine the following:
  - The source tables
  - The destination tables
  - The schema of all tables
  - The dbspace names of where tables are located
  - The specific archive to extract data from
- Scans the archive for pages belonging to the tables being restored

- Processes each row from the data page and determines if the row is complete or partial.
  If the row is a partial row, then **archecker** determines if the remaining portion of the row has been staged, and if not, it stages the row for later processing.
- For a physical-only restore, applies filters to the row and rejects rows that are not required.
- Inserts the row into the destination table.

To restore a table with the original schema, the source schema must be specified. To restore a table with a different schema, the table name in the target schema must be different from the table name in the source schema. After restoring by using a different schema, the table can be renamed with the **rename table** statement.

Logical restore

After a physical restore, logical recovery can further restore tables to a user-specified point in time. To do this, the **archecker** utility reads backed-up logical logs, converts them to SQL statements, and then replays these statements to restore data.

Before performing a logical recovery, ensure that all transactions you want to restore are contained in backed-up logical logs. The **archecker** utility cannot replay transactions from the current log. You cannot perform a logical restore on an external table.

If a table is altered, dropped, or truncated during a logical restore, the restore terminates for that table. Termination occurs at the point that the alter was performed. A message in the **archecker** message log file records that an alter operation occurred.

The **archecker** utility cannot process compression dictionaries during a logical restore of compressed tables in non-logged databases. A logical restore stops for a table if it finds that a new compression dictionary was created for that table.

When performing a logical restore, **archecker** uses two processes that run simultaneously:

Stager
: Assembles the logical logs and saves them in tables.

Applier
: Converts the log records to SQL statements and executes the statements.

- The stager
- The applier

The stager

To collect the pertinent logical log records, the stager performs the following steps:

1. Scans only the backed-up logical logs
   The stager reads the backed-up logical log files and assembles complete log records.
2. Tests the logical log records
   Any log record that is not applicable to the tables being restored is rejected.
3. Inserts the logical log information in to a table
   If the logical log record is not rejected, it is inserted into a stage table.

The applier

The *applier* reads data from the control table created by the stager. It begins processing the required transaction and updates the control table to show that this transaction is in process. Next, it operates on each successive log record, row by row, until the transaction commits.

All updates to the control table occur in the same transaction as the log record modification. This allows all work to be completed or undone as a single unit, maintaining integrity at all times. If an error occurs, the transaction is rolled back and the error is recorded in the control table entry for this transaction.

When data is being restored and the DBA has elected to include a logical restore, two additional work columns and an index are added to the destination table. These columns contain the original rowid and original part number. These columns provide a unique key which identifies the location of the row on the original source archive. To control the storage of the index, use the SET WORKSPACE command (see The SET statement). Otherwise, the index is stored in the same space as the table.

After the applier has finished and the restore is complete, these columns, and any indexes created on them, are dropped from the destination table.

Syntax for archecker utility commands

The **archecker** utility provides a command-line interface for restoring data from an archive. To use **archecker**, you must specify both a configuration file and a schema command file.

```
>>-archecker-------------------------------------------------->

>--+-+-+- -b-+--| Table-level restore |--+-----+-+--+-----+--+-----+-+-><
   | | '- -t-'                          '- -d-' |  '- -v-'  '- -s-' |
   | +- -D-------------------------------------+                    |
   | '- -i-------------------------------------'                    |
   +- -V-------------------------------------------------------------+
   '- -version-------------------------------------------------------'
```

Table-level restore

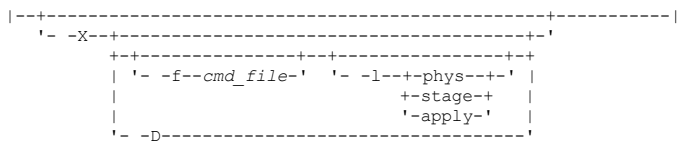


Table 1. Options for the archecker command

Element	Description
-b	Provides direct XBSA access for backups created with ON-Bar.
-d	Deletes previous **archecker** restore files, except the **archecker** message log. For more information, see When to delete restore files.
-D	Deletes previous **archecker** restore files, except the **archecker** message log, and then exits. The **-D** option can be used with the **-X** option to delete previous restore files plus any table-level-restore working tables in the **sysutils** database. For more information, see When to delete restore files.
-f *cmdfile*	Specifies that **archecker** use the command file specified by *cmdfile*. This option overrides the value of the AC_SCHEMA configuration parameter. For more information, see Schema command file.
-i	Manually initializes the system.
-lphys,stage,apply	Specifies the level of logical restore: phys Starts a logical restore of the system, but stops after physical recovery is complete. The backed up logical logs must be available. stage After physical recovery is complete, extracts the logical logs from the storage manager and stages them in their corresponding tables, and starts the stager. apply Starts the applier. The applier takes the transactions stored in the stage tables and converts them to SQL and replays the operations. The default level of logical restore if **-l** is not listed is **-lphys,stage,apply**. You can specify any combination of the logical restore levels, separated with commas. Spaces are not allowed between **-l** and levels. For more information, see Manually control a logical restore.
-s	Prints a status message to the screen.
-t	Specifies **ontape** as the backup utility.
-v	Specifies verbose mode.
-X	Specifies a table-level restore.
-V	Displays IBM® Informix® version information.
-version	Displays additional version information about the build operation system, build number, and build date for IBM Informix.

When you use ON-Bar, you can use an ON-Bar command to access **archecker** information to verify a backup. For information on the syntax for this command, see onbar -v syntax: Verifying backups.

- Manually control a logical restore
- Performing a restore with multiple storage managers
- Perform a parallel restore
- Restore tables with large objects
- When to delete restore files

Manually control a logical restore

You can manually control the stager and applier with the -l command-line option.

The following examples show how to perform a logical restore. In all examples, the name of the schema command file is cmdfile.

The following example is a typical usage:

```
archecker -bvs -f cmdfile
```

This command is equivalent to the following command:

```
archecker -bvs -f cmdfile -lphys,stage,apply
```

After the physical restore is complete, the **archecker** utility starts the stager. After the stager has started, the applier is automatically started.

In the following example, the -lphys option performs a physical-only restore:

```
archecker -bvs -f cmdfile -lphys
```

In the following example, the -lstage option starts the **archecker** stager. The stager extracts the logical log records from the storage manager and saves the applicable records to a table.

```
archecker -bvs -f cmdfile -lstage
```

The stager should only be started after physical recovery has completed.

In the following example, the -lapply option starts the **archecker** applier. It looks in the **acu_control** table for the transaction to recover. The applier should only be started after the stager has been started.

```
archecker -bvs -f cmdfile -lapply
```

Performing a restore with multiple storage managers

If you use multiple storage managers, you can perform a table-level restore with **archecker** by configuring **archecker** on every node.

To perform a table-level restore that involves multiple storage managers:

1. Create an **archecker** configuration file on every node.
2. Create a schema command file on every node.
3. Remove old restores by executing the **archecker -DX** command on a single node.
4. Start the physical restore by executing the **archecker -bX -lphys** command on each node.
   Restriction: Do not use the -d option.
5. After the physical restore completes, start the logical restore by executing the **archecker -bX -lstage** command on each node that contains logical log records.
   Restriction: Do not use the -d option.
6. After starting all stagers, complete the restore by executing the **archecker -bX -lapply** command on a single node.

Perform a parallel restore

If you have a fragmented table that resides in separate dbspaces, you can perform a physical table-level restore in parallel by executing multiple **archecker** commands with different schema command files for each dbspace.

During a level-0 archive, there cannot be any open transactions that would change the schema of the table. The table or table fragments being recovered must exist in the level-0 archive. The table or fragment cannot be created or added during the logical recovery. Tables created or fragments added during the logical recovery are ignored.

Because a detached fragment is no longer part of the original table, the applier does not process the detached fragment log record or any other log records for this fragment from this point forward. A message in the **archecker** message log file indicates a detach occurred.

In this example, the table is fragmented across three dbspaces. The corresponding schema command files are named cmdfile1, cmdfile2, cmdfile3. The following commands delete previous restores and then perform physical restores on each dbspace in parallel:

- **archecker -DX**
- **archecker -bvs -f cmdfile1 -lphys**
- **archecker -bvs -f cmdfile2 -lphys**
- **archecker -bvs -f cmdfile3 -lphys**

You cannot perform a logical restore in parallel.

Restore tables with large objects

ON-Bar supports table-level restores of smart large objects and binary large objects.

- Smart large objects
  Table-level restore also supports smart large objects for physical restore only (restore from level-0 archive).

  The storage location of the smart large object columns being restored must be specified with the PUT clause in the CREATE TABLE statement. The restored smart large objects are created with the create-time flags LO_NOLOG and LO_NOKEEP_LASTACCESS_TIME. These flags override the LOG and KEEP ACCESS TIME column attributes if they are specified in the target table for the smart large object column.

- Binary large objects
  Table-level restore supports restoring tblspace binary large objects, but not blobspace binary large objects. If you attempt to restore a blobspace binary large object, the value is set to NULL and a warning is issued.

When to delete restore files

If you repeatedly run the same **archecker** table-level restore, you must clean up the **archecker** table-level restore working files and tables from the previous runs. These working tables refer to acu_ tables in the **sysutils** database that are created during an **archecker** table-level restore. The **archecker** table-level restore working files and tables are kept after an **archecker** table-level restore completes in case these files and tables are needed for diagnosing problems.

You can remove the working files and tables by explicitly running the command **archecker -DX** or by using the -d option when you run the next **archecker** table-level restore command. The -d option indicates that all files and tables from the previous run of **archecker** table-level restore are removed before the new restore begins.

- **ontape** example: **archecker -tdvs -f***schema_command_file*
- **onbar** example: **archecker -bdvs -f***schema_command_file*

The archecker schema reference

The topics in this section describe the SQL-like statements used by the **archecker** schema command file. This file provides information that the **archecker** utility uses to perform data recovery.

Use the schema command file to specify the source and destination tables and to define the table schema.

For information about specifying which command file **archecker** uses, see Schema command file.

The following are statements supported by **archecker**:

- CREATE TABLE
- DATABASE
- INSERT INTO
- RESTORE
- SET

Important: Standard SQL comments are allowed in the **archecker** utility file and are ignored during processing.
The syntax of these statements is described in the following topics.

- The CREATE TABLE statement
- The CREATE EXTERNAL TABLE statement
- The DATABASE statement
  In the **archecker** utility, the DATABASE statement sets the current database.
- The INSERT statement
  The INSERT statement tells the **archecker** utility what tables to extract and where to place the extracted data.
- The RESTORE statement
  The RESTORE statement is an optional command to restore tables to a specific point in time.
- The SET statement
- Schema command file examples
  This section contains examples that show different command file syntax for different data recovery scenarios.

The CREATE TABLE statement

The CREATE TABLE statement describes the schema of the source and target tables. If the target table is external, use the CREATE EXTERNAL TABLE statement described in the section The CREATE EXTERNAL TABLE statement.

Syntax

The syntax of the CREATE TABLE used in the **archecker** schema command file is identical to the corresponding IBM® Informix® SQL statement. For a description of this syntax, see the *IBM Informix Guide to SQL: Syntax*.

Usage

You must include the schema for the source table in the **archecker** schema command file. This schema must be identical to the schema of the source table at the time the archive was created.

The schema of the source table is not validated by **archecker**. Failing to provide an accurate schema leads to unpredictable results.

The source table cannot be a synonym or view. The schema of the source table only needs the column list and storage options. Other attributes such as extent sizes, lock modes, and so on are ignored. For an ON-Bar archive, **archecker** uses the list of storage spaces for the source table to create its list of objects to retrieve from the storage manager. If the source table is fragmented, you must list all dbspaces that contain data for the source table. The **archecker** utility only extracts data from the dbspaces listed in the schema command file.

If the source table contains constraints, indexes, or triggers, they are automatically disabled during the restore. Foreign constraints that reference the target table are also disabled. After the restore is complete, the constraints, indexes, and triggers are enabled. For better performance, remove constraints, indexes, and triggers prior to performing a restore.

You must also include the schema of the target table in the command file. If the target table does not exist at the time the restore is performed, it is created using the schema provided.

If the target table exists, its schema must match the schema specified in the command file. Data is then appended to the existing table.

Examples

The schema of the source and target tables do not have to be identical. The following example shows how you can repartition the source data after performing the data extraction:

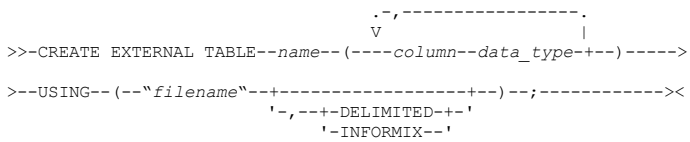
```
CREATE TABLE source (col1 integer, ...) IN dbspace1;
CREATE TABLE target (col1 integer, ...)
   FRAGMENT BY EXPRESSION
      MOD(col1, 3) = 0 in dbspace3,
      MOD(col1, 3) = 1 in dbspace4,
      MOD(col1, 3) = 2 in dbspace5;
INSERT INTO target SELECT * FROM source;
```

The CREATE EXTERNAL TABLE statement

The CREATE EXTERNAL TABLE statement describes the schema of an external target table.

Syntax

The syntax of the CREATE EXTERNAL TABLE statement for the **archecker** schema file is not identical to the SQL CREATE EXTERNAL TABLE statement.



Element	Description
column	The name of the column. Must conform to SQL identifier syntax rules. For more information, see the *IBM® Informix® Guide to SQL: Syntax*.
data_type	The built-in data type of the column. For more information about data types, see the *IBM Informix Guide to SQL: Reference*.
filename	Either the name of the file in which to place the data or a pipe device. The pipe device must exist before starting the **archecker** utility.
name	The name of the table to store the external data. Must be unique among names of tables, views, and synonyms in the current database. Must conform to SQL database object name rules. For more information, see the *IBM Informix Guide to SQL: Syntax*.

Usage

When you use the CREATE EXTERNAL TABLE statement to send data to an external table, the data is only extracted from a level-0 archive. Logical logs are not rolled forward on an external table.

You can specify either of the following formats for external files:

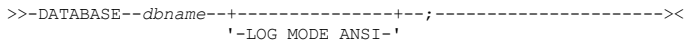
- DELIMITED: ASCII delimited file. This is the default format.
- INFORMIX: internal binary representation. To optimize performance, filters are not applied to external tables. If filters exist, a warning indicates that they are ignored.

For an example of using the CREATE EXTERNAL TABLE statement, see Restore to an external table.

The DATABASE statement

In the **archecker** utility, the DATABASE statement sets the current database.

Syntax



Element	Description
dbname	The name of the current database.

Usage

Multiple DATABASE statements can be used. All table names referenced following this statement are associated with the current database.

If the logging mode of the source database is ANSI and default decimal columns are used in the table schemas, then the logging mode of the database must be declared.

If the logging mode of the source database is not declared no error will be returned, but unexpected results and data can occur.

Examples

In the following example, both the source and target tables reside in the same database **dbs**.

```
DATABASE dbs;
CREATE TABLE source (...);
CREATE TABLE target (...);
INSERT INTO target SELECT * from source;
```

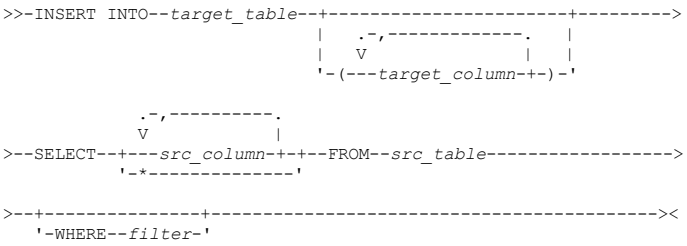
You can use multiple database statements to extract a table from one database into another database.

```
DATABASE dbs1;
CREATE TABLE source (...) IN dbspace1;
DATABASE dbs2;
CREATE TABLE target (...) IN dbspace2;
INSERT INTO dbs2:target SELECT * FROM dbs1:source;
```

The INSERT statement

The INSERT statement tells the **archecker** utility what tables to extract and where to place the extracted data.

Syntax



Element	Description
filter	The following filters are supported by the INSERT statement: • =, !=, <> • >, >=, <, <= • [NOT] MATCHES, [NOT] LIKE • IS [NOT] NULL • AND, OR • TODAY, CURRENT The following operators are not supported by the **archecker** utility: • Aggregates • Functions and procedures • Subscripts • Subqueries • Views • Joins Filters can only be applied to physical-only restore.
src_column	A list of columns to be extracted.
src_table	The source table on the archive where the data is restored from.
target_column	The destination column or columns where the data will be restored.
target_table	The destination table where the data will be restored.

Examples

The following example demonstrates the simplest form of the INSERT statement. This statement extracts all rows and columns from the source to the target table.

```
INSERT INTO target SELECT * FROM source;
```

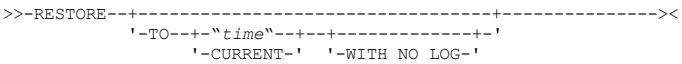
You can also extract a subset of columns. In the following example, only two columns from the source table are inserted into the destination table.

```
CREATE TABLE source (col1 integer, col2 integer, col3 integer, col4 integer);
CREATE TABLE target (col1 integer, col2 integer);
INSERT INTO target (col1, col2) SELECT col3, col4 FROM source;
```

The RESTORE statement

The RESTORE statement is an optional command to restore tables to a specific point in time.

Syntax



Element	Description
"*time*"	The date and time the table is to be restored to.

Usage

The TO clause is used to restore the table to a specific point in time, which is specified by a date and time or the reserved word CURRENT.

Only one RESTORE statement can be specified in a command file. If this statement is not present in the command file, then the system will be restored to the most current time using logical logs.

If the WITH NO LOG clause is present, only a physical restore is performed. In addition, the two extra columns and the index are not added to the destination table. Physical-only restores are based on level-0 archives only.

Tip: Use this option when you do not have logical logs. You will not receive any messages about logical recovery.

Example

```
RESTORE TO CURRENT WITH NO LOG;
```

The SET statement

The SET statement controls the different features in the table-level unload library.

Syntax

```
>>-SET--+-COMMIT TO--number-------------+---------------------><
        |               .-,-----------. |
        |               V             | |
        '-WORKSPACE TO------dbspace---+-'
```

Element	Description
number	Sets the number of records to insert before committing during a physical restore. The default is 1000.
dbspace	The dbspaces to use for the working storage space. The default is the root dbspace. You cannot use temporary dbspaces for the working storage space.

The **archecker** utility creates several tables for the staging of logical log records during a logical restore. These tables are created in the **sysutils** database and stored in the working storage space.

Examples

```
SET COMMIT TO 20000;
SET WORKSPACE to dbspace1;
```

Schema command file examples

This section contains examples that show different command file syntax for different data recovery scenarios.

- Simple schema command file
- Restore a table from a previous backup
- Restore to a different table
- Extract a subset of columns
- Use data filtering
- Restore to an external table
- Restore multiple tables
- Perform a distributed restore

Simple schema command file

The schema command file in this example extracts a table from the most recent level-0 backup of **dbspace1**. The data is placed in the **table test1:tlr** and the logs are applied to bring the table **tlr** to the current point in time.

```
database test1;
create table tlr (
   a_serial serial,
   b_integer integer,
   c_char char,
   d_decimal decimal
   ) in dbspace1;
insert into tlr select * from tlr;
```

Restore a table from a previous backup

The schema command file in this example extracts a table from the level-0 backup of **dbspace1**. The logical logs are used to bring the table to the time of "2003-01-01 01:01:01". The data is placed in the table **test1:tlr**.

```
database test1;
create table tlr (
   a_serial serial,
   b_integer integer,
```

```
   c_char char,
   d_decimal decimal
   ) in dbspace1;
insert into tlr select * from tlr;
restore to '2003-01-01 01:01:01';
```

Restore to a different table

The schema command file in this example extracts a table called **test1:tlr** from the most recent backup of **dbspace1** and places the data in the table **test1:tlr_dest**.

```
database test1;
create table tlr    (
   a_serial   serial,
   b_integer  integer,
   c_char     char(20),
   d_decimal  decimal,
   ) in dbspace1;
create table tlr_dest    (
   a_serial   serial,
   b_integer  integer,
   c_char     char(20),
   d_decimal  decimal
   ) in dbspace2;
insert into tlr_dest select * from tlr;
```

Extract a subset of columns

The schema command file in this example extracts a table **test1:tlr** from the most recent backup of **dbspace1** and places a subset of the data into the table **test1:new_dest**

```
database test1;
create table tlr    (
   a_serial   serial,
   b_integer  integer,
   c_char     char(20),
   d_decimal  decimal
   ) in dbspace1;
create table new_dest (
   X_char     char(20),
   Y_decimal  decimal,
   Z_name     char(40)
   ) in dbspace2;
insert into new_dest (X_char, Y_decimal) select c_char,d_decimal from tlr;
```

Use data filtering

The schema command file in this example extracts a table **test1:tlr** from the most recent backup of **dbspace1** and places the data in the table **test1:tlr** only where the list conditions are true.
Important: Filters can only be applied to a physical restore.

```
database test1;
create table tlr (
   a_serial   serial,
   b_integer  integer,
   c_char     char(20),
   d_decimal  decimal,
   ) in dbspace1;
insert into tlr
  select * from tlr
  where c_char matches 'john*'
  and d_decimal is NOT NULL
  and b_integer > 100;
restore to current with no log;
```

Restore to an external table

The schema command file in this example extracts a table called **test1:tlr** from the most recent backup of **dbspace1** and places the data in a file called /tmp/tlr.unl.

```
database test1;
create table tlr
(a_serial serial,
  b_integer integer
) in dbspace1;
create external table tlr_dest
  (a_serial serial,
   b_integer integer
  ) using ("/tmp/tlr.unl", delimited );
insert into tlr_dest select * from tlr;
restore to current with no log;
```

Restore multiple tables

The schema command file in this example extracts a table **test1:tlr_1** and **test1:tlr_2** from the most recent backup of **dbspace1** and places the data in **test1:tlr_1_dest** and **test1:tlr_2_dest**. This is an efficient way of restoring multiple tables because it requires only one scan of the archive and logical log files.

```
database test1;
create table tlr_1
  (  columns  ) in dbspace1;
create table tlr_1_dest (  columns  );
create table tlr_2
  (  columns  ) in dbspace1;
create table tlr_2_dest (  columns  );
insert into tlr_1_dest select * from tlr_1;
insert into tlr_2_dest select * from tlr_2;
```

Perform a distributed restore

The schema command file in this example extracts a table **test:tlr_1** from the most recent backup of dbspace1 and places the data on the database server **rem_srv** in the table **rem_dbs:tlr_1**.

```
database rem_dbs
create table tlr_1
   ( columns  );
database test1;
create table tlr_1
  (  columns  ) in dbspace1;
insert into rem_dbs@rem_srv.tlr_1
  select * from tlr_1;
```

Backup and restore configuration parameter reference

- Backup and restore configuration parameters
  These topics describe the configuration parameters that you use with the ON-Bar, **ontape**, and **archecker** utilities.

Backup and restore configuration parameters

These topics describe the configuration parameters that you use with the ON-Bar, **ontape**, and **archecker** utilities.

You set most of these configuration parameters in the onconfig file. However, you set some of the **archecker** configuration parameters in the AC_CONFIG file.

Be sure to configure your storage manager. Depending on the storage manager that you choose, you might set different ON-Bar configuration parameters. If you are using a third-party storage manager, see Configuring a third-party storage manager, before you start ON-Bar.

The following table describes the following attributes (if relevant) for each parameter.

Attribute	Description
ac_config.std *value*	For **archecker** configuration variables. The default value that appears in the ac_config.std file.
onconfig.std *value*	For onconfig configuration variables. The default value that appears in the onconfig.std file.
if value not present	The value that the database server supplies if the parameter is missing from your onconfig file. If this value is present in onconfig.std, the database server uses the onconfig.std value. If this value is not present in onconfig.std, the database server uses this value.
units	The units in which the parameter is expressed
range of values	The valid values for this parameter
takes effect	The time at which a change to the value of the parameter affects ON-Bar operation. Except where indicated, you can change the parameter value between a backup and a restore.
refer to	Cross-reference to further discussion

- ON-Bar and ontape configuration parameters and environment variable
  Many properties of the ON-Bar and **ontape** utilities are controlled by configuration parameters in the onconfig file. ON-Bar also has an environment variable.
- The archecker utility configuration parameters and environment variable
  These topics describe the AC_CONFIG environment variable and the configuration parameters that you use with the **archecker** utility.
- Informix Primary Storage Manager configuration parameters
  The IBM® Informix® Primary Storage Manager uses the information in some specific configuration parameters.
- Event alarm configuration parameters
  When you set configuration parameters for use with the ON-Bar and **ontape** utilities, also determine if you need to adjust the ALARMPROGRAM and ALRM_ALL_EVENTS configuration parameters.

Related reference:
ON-Bar backup and restore system

ON-Bar and ontape configuration parameters and environment variable

Many properties of the ON-Bar and **ontape** utilities are controlled by configuration parameters in the onconfig file. ON-Bar also has an environment variable.

Important: ON-Bar does not use the TAPEDEV, TAPEBLK, TAPESIZE, LTAPEBLK, and LTAPESIZE configuration parameters. ON-Bar checks if LTAPEDEV is set to /dev/null on UNIX or NUL on Windows.

- BACKUP_FILTER configuration parameter
  Use the BACKUP_FILTER configuration parameter to specify the path name and any options for an external filter program that you use with the ON-Bar or **ontape** utility.
- BAR_ACT_LOG configuration parameter
  Use the BAR_ACT_LOG configuration parameter to specify the full path name of the ON-Bar activity log.
- BAR_BSALIB_PATH configuration parameter
  Use the BAR_BSALIB_PATH configuration parameter to specify the path name and file name of the XBSA shared library for the storage manager that you use.
- BAR_CKPTSEC_TIMEOUT configuration parameter
  The BAR_CKPTSEC_TIMEOUT configuration parameter specifies the amount of time, in seconds, that an RS secondary server should wait for a checkpoint to arrive from the primary server while performing an external backup.
- BAR_DEBUG configuration parameter
  Use the BAR_DEBUG configuration parameter to specify the amount of debugging information that the database server captures in the ON-Bar activity log.
- BAR_ENCRYPTION configuration parameter
  Use the BAR_ENCRYPTION parameter to encrypt the backups.
- BAR_DECRYPTION configuration parameter
  Use the BAR_DECRYPTION parameter to decrypt the backups.
- BAR_DEBUG_LOG configuration parameter
  Use the BAR_DEBUG_LOG parameter to specify the location and name of the ON-Bar debug log.
- BAR_HISTORY configuration parameter
  Use the BAR_HISTORY configuration parameter to specify whether the **sysutils** database maintains a backup history when you use **onsmsync** to expire old backups.
- BAR_IXBAR_PATH configuration parameter
  Use the BAR_IXBAR_PATH configuration parameter to change the path and name of the ON-Bar boot file.
- BAR_MAX_BACKUP configuration parameter
  Use the BAR_MAX_BACKUP parameter to specify the maximum number of parallel processes that are allowed for each ON-Bar command.
- BAR_MAX_RESTORE configuration parameter
  Use the BAR_MAX_RESTORE parameter to specify the maximum number of parallel restore processes that are allowed during an ON-Bar restore operation.
- BAR_NB_XPORT_COUNT configuration parameter
  Use the BAR_NB_XPORT_COUNT configuration parameter to specify the number of data buffers that each **onbar_d** process can use to exchange data with the database server.
- BAR_PERFORMANCE configuration parameter
  Use the BAR_PERFORMANCE configuration parameter to specify the type of performance statistics to report to the ON-Bar activity log for backup and restore operations.
- BAR_PROGRESS_FREQ configuration parameter
  Use the BAR_PROGRESS_FREQ configuration parameter to specify, in minutes, the frequency of the progress messages in the ON-Bar activity log for backup and restore operations.
- BAR_RETRY configuration parameter
  Use the BAR_RETRY configuration parameter to specify how many times **onbar** should try a data backup, logical-log backup, or restore operation if the first attempt fails.
- BAR_SEC_ALLOW_BACKUP configuration parameter
  Use the BAR_SEC_ALLOW_BACKUP configuration parameter on an RSS node to enable the taking of archives and log backups on that node using either **onbar** or **ontape**.
- BAR_SIZE_FACTOR configuration parameter
  Use the BAR_SIZE_FACTOR configuration parameter to augment the estimate for the size of a backup object, before the backup.
- BAR_XFER_BUF_SIZE configuration parameter
  Use the BAR_XFER_BUF_SIZE configuration parameter to specify the size of each transfer buffer.
- IFX_BAR_NO_BSA_PROVIDER environment variable
  Set the IFX_BAR_NO_BSA_PROVIDER environment variable to force ON-Bar to use the sm_versions file as the source of information about the XBSA library for the storage manager.
- IFX_BAR_NO_LONG_BUFFERS environment variable
  Set the IFX_BAR_NO_LONG_BUFFERS environment variable to prevent the size of transfer buffers from exceeding 64 KB when the BAR_XFER_BUF_SIZE configuration parameter is set to a long transfer buffer size value.
- IFX_BAR_USE_DEDUP environment variable
  Set the IFX_BAR_USE_DEDUP environment variable to optimize the deduplication capabilities of storage managers.
- IFX_TSM_OBJINFO_OFF environment variable
  Set the IFX_TSM_OBJINFO_OFF environment variable to disable support for restoring backup objects that are replicated, imported, or exported between IBM® Spectrum Protect servers.
- LTAPEBLK configuration parameter
  Use the LTAPEBLK configuration parameter to specify the block size of the device to which the logical logs are backed up when you use **ontape** for dbspace backups.
- LTAPEDEV configuration parameter
  Use the LTAPEDEV configuration parameter to specify the device or directory file system to which the logical logs are backed up when you use **ontape** for backups.
- LTAPESIZE configuration parameter
  Use the LTAPESIZE configuration parameter to specify the maximum tape size of the device to which the logical logs are backed up when you use **ontape** for backups.
- RESTARTABLE_RESTORE configuration parameter
  Use the RESTARTABLE_RESTORE configuration parameter to enable or disable restartable restores.
- RESTORE_FILTER configuration parameter
  Use the RESTORE_FILTER configuration parameter to specify the path name of a filter program, and any options.
- TAPEBLK configuration parameter
  Use the TAPEBLK configuration parameter to specify the block size of the device to which **ontape** writes during a storage-space backup.

- TAPEDEV configuration parameter
  Use the TAPEDEV configuration parameter to specify the device or directory file system to which the **ontape** utility backs up storage spaces.
- TAPESIZE configuration parameter
  Use the TAPESIZE parameter specifies the size of the device to which the **ontape** utility backs up storage spaces.

Related tasks:
Configuring Informix Primary Storage Manager

BACKUP_FILTER configuration parameter

Use the BACKUP_FILTER configuration parameter to specify the path name and any options for an external filter program that you use with the ON-Bar or **ontape** utility.

onconfig.std value
: Not set. Backup data is not filtered.

values
: The path name of a command and any options. By default, the path name is relative to the $INFORMIXDIR/bin directory, otherwise, the path name must be the absolute path of the program. If you include command-line options, both the filter name and the options must be surrounded by single quotation marks.

takes effect
: After you edit your onconfig file and ON-Bar or **ontape** starts.

Usage

This filter transforms data before backing it up, such as compressing it. The transformed data is then backed up and stored as a single file. When you perform a restore, you must transform the data back to its original format. Specify the appropriate program to transform data before a restore by setting the RESTORE_FILTER configuration parameter.

For security purposes, filters should not have write permission to non-privileged users. Permission on the filters is the same as that of permission on other executable files that are called by the IBM® Informix® server or utilities.

Note: If the BACKUP_FILTER parameter is set in the onconfig file, the LTAPESIZE configuration parameter cannot be set to 0. Otherwise the ON-Bar or **ontape** utility returns an error when backing up logical logs to a directory on disk. The error message is:

```
The LTAPESIZE configuration parameter cannot be set to 0 when the BACKUP_FILTER
configuration parameter is set; change the value of LTAPESIZE.
Program over.
```

A workaround is to set the LTAPESIZE configuration parameter to a high value. Log files are not much higher than the LOGSIZE configuration parameter. Use the value in the LOGSIZE as the upper limit for this database.

When you specify filter information in the BACKUP_FILTER configuration parameter, specify the path name of a filter program, and any options, as shown in this example:

```
BACKUP_FILTER           /bin/compress
```

Output produced by this filter is saved as a single object in the storage manager.

The BACKUP_FILTER configuration parameter can include command-line options as well as the filter name. For example, specify:

```
BACKUP_FILTER   'my_encrypt -file /var/adm/encryption.pass'
```

Related reference:
RESTORE_FILTER configuration parameter

BAR_ACT_LOG configuration parameter

Use the BAR_ACT_LOG configuration parameter to specify the full path name of the ON-Bar activity log.

onconfig.std value
: UNIX: BAR_ACT_LOG $INFORMIXDIR/tmp/bar_act.log
  Windows: BAR_ACT_LOG %INFORMIXDIR%\tmp\bar_act.log

range of values
: Full path name

takes effect
: When **onbar-driver** starts
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

You should specify a path to an existing directory with an appropriate amount of space available or use $INFORMIXDIR/bar_act.log.

Whenever a backup or restore activity or error occurs, ON-Bar writes a brief description to the activity log. The format of the file resembles the format of the database server message log. You can examine the activity log to determine the results of ON-Bar actions.

The file specified by the BAR_ACT_LOG configuration parameter is created if it does not exist. If the ON-Bar command (or any ON-Bar-related utility such as the **onsmsync** utility) never ran on the system, then the file does not exist.

The **sysbaract_log** table is a system monitoring interface pseudo table that reads data from the file specified by BAR_ACT_LOG. The following errors are returned if you attempt to query the **sysbaract_log** on a system where the BAR_ACT_LOG file does not exist:

```
244: Could not do a physical-order read to fetch next row.
101: ISAM error:  file is not open.
```

Usage when you specify a file name only

If you specify a file name only in the BAR_ACT_LOG configuration parameter, ON-Bar creates the ON-Bar activity log in the working directory in which you started ON-Bar. For example, if you started ON-Bar from /usr/mydata on UNIX, the activity log is written to that directory.

For UNIX, if the database server launches a continuous logical-log backup, ON-Bar writes to the ON-Bar activity log in the working directory for the database server.

For Windows, if the database server launches a continuous logical-log backup, ON-Bar writes to the activity log in the %INFORMIXDIR%\bin directory instead.

Related concepts:
bar_act.log file: ON-Bar activity log
Related reference:
View ON-Bar backup and restore performance statistics
PSM_ACT_LOG configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_BSALIB_PATH configuration parameter

Use the BAR_BSALIB_PATH configuration parameter to specify the path name and file name of the XBSA shared library for the storage manager that you use.

default value
: UNIX: $INFORMIXDIR/lib/ibsad001.*extension*
: Windows: %INFORMIXDIR%\lib\ibsad001.*extension*
: The *extension* is platform specific.

onconfig.std value
: Not set.

takes effect
: When **onbar-driver** starts
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

ON-Bar and the storage manager rely on a shared library to integrate with each other. Configure the BAR_BSALIB_PATH configuration parameter for your storage-manager library. Support for BAR_BSALIB_PATH is platform-specific. Check your machine notes to determine whether you can use this configuration parameter with your operating system. You can change the value of BAR_BSALIB_PATH between a backup and restore.

To ensure that this integration takes place, specify the shared-library path name. Set one of the following options:

UNIX:

- Place the storage-manager library in the default directory.
  For example, the suffix for Solaris is `so`, so you specify $INFORMIXDIR/lib/libbsapsm.so on a Solaris system.
- Place the storage-manager library in any directory and create a symbolic link from $INFORMIXDIR/lib/ibsad001.*platform_extension* to it.
- Set the LD_LIBRARY_PATH environment variable. For example, set LD_LIBRARY_PATH to $INFORMIXDIR/lib.

Windows:

- Place the storage-manager library in the default directory.

If the parameter BAR_BSALIB_PATH is missing or has no value and the database server cannot open the XBSA shared library for your platform, ON-BAR tries to use the IBM® Informix® Primary Storage Manager as the storage manager in all platforms.

Tip: Be sure that the shared library can access the backup data in the storage manager in a restore. You cannot back up on to one storage manager and restore from a different storage manager.
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_CKPTSEC_TIMEOUT configuration parameter

The BAR_CKPTSEC_TIMEOUT configuration parameter specifies the amount of time, in seconds, that an RS secondary server should wait for a checkpoint to arrive from the primary server while performing an external backup.

onconfig.std value

- UNIX: 15
- Windows: 16

default value

- UNIX: 15
- Windows: 16

units

seconds

range of values
5 through twice the value of the CKPTINTVL configuration parameter

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

When an external backup is performed on an RS secondary server, the secondary server must wait until a checkpoint arrives in the logical logs from the primary server. A checkpoint flushes buffers to disk, and blocks user transactions that involve temporary tables. If the checkpoint on the primary does not complete in the time-out period, the backup on the RS secondary server fails. You can set the BAR_CKPTSEC_TIMEOUT configuration parameter to a longer amount of time, in seconds, that an RS secondary server should wait for a checkpoint to arrive from the primary server while performing an external backup.

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
CKPTINTVL configuration parameter

BAR_DEBUG configuration parameter

Use the BAR_DEBUG configuration parameter to specify the amount of debugging information that the database server captures in the ON-Bar activity log.

onconfig.std value
BAR_DEBUG 0

values
0 = Do not display debugging information.
1 = Print a small amount of information
2 = Print a message every time ON-Bar:

- Enters a function.
- Exits a function. The message includes the return code for the function.

3 = Print exit and entry information with additional details.
4 = Also print information about ON-Bar parallel operations.
5 = Also print information about:

- Objects that are being backed up or restored.
- The act_node structure corresponding with the bar_action table.

6 = Print additional information about:

- Objects that are being backed up or restored.
- The act_node structure corresponding with the bar_action table.

7 = Also print:

- Information about the contents of the ins_node structure corresponding with the bar_instance table.
- Information about modifications to the bar_action table.
- Information about logical logs and objects that are restored.
- SQL statements done on the **sysutils** database and SQLCODES that were returned.

8 = Also print page headers of all pages archived and restored. This setting requires a large amount of space.
9 = Print the contents of:

- The bar_ins structure after it was initialized.
- The object descriptors that are cold restored.

takes effect
Immediately after you edit your onconfig file for any currently executing ON-Bar command and any subsequent commands. Any ON-Bar command that is currently executing when you update BAR_DEBUG reads the new value of BAR_DEBUG and prints debug messages at the new level.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

The default value of 0 displays no debugging information. Set the BAR_DEBUG configuration parameter to a higher value to display more detailed debugging information in the ON-Bar activity log.

You can dynamically update the value of BAR_DEBUG in the onconfig file during a session.

Related reference:
BAR_DEBUG_LOG configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_ENCRYPTION configuration parameter

Use the BAR_ENCRYPTION parameter to encrypt the backups.

onconfig.std *value*
Not set. Backup data is not encrypted.

takes effect
When ON-Bar or ontape starts.

Usage

Use the BAR_ENCRYPTION configuration parameter to enable backup encryption, set the path to the keystore containing the credentials to access a remote key server or the path of a keyfile containing the backup encryption key, and specify the encryption cipher.

Syntax for the BAR_ENCRYPTION configuration parameter

```
>>-BAR_ENCRYPTION--keystore--=--keystore_name------------------>
                 '--keyfile---=--keyfile_name--'
>--+--------------------------+--------------------------------->
   '-,--cipher--=--+-aes128-+-'
                   +-aes192-+
                   '-aes256-'
```

Table 1. Options for the BAR_ENCRYPTION configuration parameter value

Field	Value
keystore	The *keystore* specifies the name of the keystore and stash file names. The files are created in the INFORMIXDIR/etc directory: • *keystore*.p12 = The keystore file that contains the security certificates. • *keystore*.sth = The stash file that contains the encryption password. You must manually back up the keystore and password stash files. These files are not backed up when you run a back up with the ON-Bar or **ontape** utilities.
keyfile	The *keyfile* specifies the full path of a text file that contains a backup encryption key of suitable size for the cipher chosen. The key must be encoded in base64 format.
cipher	Specifies the encryption cipher: • **aes128** = Default. Advanced Encryption Standard cipher with 128-bit keys. • **aes192** = Advanced Encryption Standard cipher with 192-bit keys. • **aes256** = Advanced Encryption Standard cipher with 256-bit keys.

BAR_DECRYPTION configuration parameter

Use the BAR_DECRYPTION parameter to decrypt the backups.

onconfig.std *value*
Not set. If BACKUP_DECRYPTION is not set, backup data that will not be decrypted.

takes effect
When ON-Bar, ontape, onlog and archecker starts.

Usage

Use the BAR_DECRYPTION to override the BAR_ENCRYPTION configuration parameter during a restore. Like with BAR_DECRYPTION, you can set the path to a keystore or to a keyfile. You cannot set the cipher in this parameter as the cipher is set by the object to be restored.

Syntax for the BAR_DECRYPTION configuration parameter

```
>>-BAR_DECRYPTION--keystore--=--keystore_name------------------>
                 '--keyfile---=--keyfile_name--'
```

Table 1. Option for the BAR_DECRYPTION configuration parameter value

Field	Value
keystore	The *keystore* specifies the name of the keystore and stash file names. The files are created in the INFORMIXDIR/etc directory: • *keystore*.p12 = The keystore file that contains the security certificates. • *keystore*.sth = The stash file that contains the decryption password. You must manually back up the keystore and password stash files. These files are not backed up when you run a back up with the ON-Bar or **ontape** utilities.
keyfile	The *keyfile* specifies the full path of a text file that contains a backup decryption key. The key must be encoded in base64 format.

BAR_DEBUG_LOG configuration parameter

Use the BAR_DEBUG_LOG parameter to specify the location and name of the ON-Bar debug log.

onconfig.std *value*
    /usr/informix/bar_dbug.log
if value not present
    UNIX: /tmp/bar_dbug.log
    Windows: \tmp\bar_dbug.log
takes effect
    When ON-Bar starts
    When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

For security reasons, set the BAR_DEBUG_LOG configuration parameter to a directory with restricted permissions, such as the $INFORMIXDIR directory.

Related reference:
BAR_DEBUG configuration parameter
PSM_DEBUG_LOG configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_HISTORY configuration parameter

Use the BAR_HISTORY configuration parameter to specify whether the **sysutils** database maintains a backup history when you use **onsmsync** to expire old backups.

onconfig.std value
    Not in the onconfig.std file.
default value
    0
range of values
    0 = Remove records for expired backup objects from the **sysutils** database
    1 = Keep records for expired backup objects in the **sysutils** database
takes effect
    When **onsmsync** starts
    When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

If you set the value to 0, **onsmsync** removes the **bar_object**, **bar_action**, and **bar_instance** rows for the expired backup objects from the **sysutils** database. If you set the value to 1, **onsmsync** sets the **act_type** value to 7 in the **bar_action** row and keeps the **bar_action** and **bar_instance** rows for expired backup objects in the **sysutils** database. If you do not set BAR_HISTORY to 1, the restore history is removed.

Regardless of the value of BAR_HISTORY, **onsmsync** removes the line that describes the backup object from the emergency boot file and removes the object from the storage manager when the storage manager expires the object.

For more information about **onsmsync**, see The onsmsync utility.

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_IXBAR_PATH configuration parameter

Use the BAR_IXBAR_PATH configuration parameter to change the path and name of the ON-Bar boot file.

onconfig.std value
    Not set in onconfig.std.
default value
    UNIX or Linux: $INFORMIXDIR/etc/ixbar.*servernum*
    Windows: %INFORMIXDIR%\etc\ixbar.*servernum*
range of values
    Full path name for the ON-Bar boot file
takes effect
    When ON-Bar or **onsmsync** starts
    When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

By default, the ON-Bar boot file is created in the %INFORMIXDIR%\etc folder on Windows and in the $INFORMIXDIR/etc folder on UNIX or Linux. The default name for this file is ixbar.*servernum*, where *servernum* is the value of the SERVERNUM configuration parameter.

For example, in an instance with the SERVERNUM configuration parameter equal to 41, the ON-Bar boot file is created by default with this path and name in UNIX:

```
BAR_IXBAR_PATH $INFORMIXDIR/etc/ixbarboot.41
```

You can change the path to create the file in another location. For example, if you want to create the ON-Bar boot file in the directory /usr/informix with the name ixbar.new, specify:

```
BAR_IXBAR_PATH=/usr/informix/ixbar.new
```

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_MAX_BACKUP configuration parameter

Use the BAR_MAX_BACKUP parameter to specify the maximum number of parallel processes that are allowed for each ON-Bar command.

onconfig.std *value*
0
if value not present
4
units
ON-Bar processes
values
0 = Maximum number of processes allowed on system
1 = Serial backup or restore
n = Specified number of processes created
takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Although the database server default value for BAR_MAX_BACKUP is 4, the onconfig.std value is 0.

Both UNIX and Windows support parallel backups.

Specify serial backups and restores

To perform a serial backup or restore, including a serial whole system backup or restore, set BAR_MAX_BACKUP to 1.

Specify parallel backups and restores

To specify parallel backups and restores, including parallel whole system backups and restores, set BAR_MAX_BACKUP to a value higher than 1. For example, if you set BAR_MAX_BACKUP to 5 and execute an ON-Bar command, the maximum number of processes that ON-Bar creates concurrently is 5. Configure BAR_MAX_BACKUP to any number up to the maximum number of storage devices or the maximum number of streams available for physical backups and restores. ON-Bar groups the dbspaces by size for efficient use of parallel resources.

If you set BAR_MAX_BACKUP to 0, the system creates as many ON-Bar processes as needed. The number of ON-Bar processes is limited only by the number of storage spaces or the amount of memory available to the database server, whichever is less.

The amount of memory available is based on SHMTOTAL. ON-Bar performs the following calculation where N is the maximum number of ON-Bar processes that are allowed:

```
N = SHMTOTAL / (# transport buffers * size of transport buffers / 1024)
```

If SHMTOTAL is 0, BAR_MAX_BACKUP is reset to 1. If N is greater than BAR_MAX_BACKUP, ON-Bar uses the BAR_MAX_BACKUP value. Otherwise, ON-Bar starts N backup or restore processes.

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_MAX_RESTORE configuration parameter

Use the BAR_MAX_RESTORE parameter to specify the maximum number of parallel restore processes that are allowed during an ON-Bar restore operation.

onconfig.std value
0
if value not present
The value of the BAR_MAX_BACKUP configuration parameter
units
ON-Bar processes
values
0 = Maximum number of restore processes allowed on system
1 = Serial restore
n = Specified number of restore processes created
takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Both UNIX and Windows support parallel restores.

Specify serial restores

To perform a serial restore, including a serial whole system restore, set BAR_MAX_RESTORE to 1.

Specify parallel restores

To specify parallel restores, including parallel whole system restores, set BAR_MAX_RESTORE to a value higher than 1. For example, if you set BAR_MAX_RESTORE to 5 and start a restore, the maximum number of restore processes that ON-Bar creates concurrently is 5. Configure BAR_MAX_RESTORE to any number up to the maximum number of storage devices or the maximum number of streams available for physical restores. ON-Bar groups the dbspaces by size for efficient use of parallel resources.

If you set BAR_MAX_RESTORE to 0, the system creates as many ON-Bar restore processes as needed. The number of restore processes is limited only by the number of storage spaces or the amount of memory available to the database server, whichever is less.

The amount of memory available is based on SHMTOTAL. ON-Bar performs the following calculation where N is the maximum number of ON-Bar processes that are allowed:

```
N = SHMTOTAL / (# transport buffers * size of transport buffers / 1024)
```

If SHMTOTAL is 0, BAR_MAX_RESTORE is reset to 1. If N is greater than BAR_MAX_RESTORE, ON-Bar uses the BAR_MAX_RESTORE value. Otherwise, ON-Bar starts N restore processes.

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_NB_XPORT_COUNT configuration parameter

Use the BAR_NB_XPORT_COUNT configuration parameter to specify the number of data buffers that each **onbar_d** process can use to exchange data with the database server.

onconfig.std *value*
20
if value not present
20
units
Buffers
range of values
3 to unlimited
takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

The value of this parameter affects ON-Bar performance. For example, if you set BAR_NB_XPORT_COUNT to 5 and then issue five ON-Bar commands, the resulting 25 ON-Bar processes use a total of 125 buffers.

To calculate the amount of memory that each **onbar_d** process requires, use the following formula. For information about the page size for your system, see the release notes:

```
required_memory = (BAR_NB_XPORT_COUNT * BAR_XFER_BUF_SIZE
                   * page_size) + 5 MB
```

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_PERFORMANCE configuration parameter

Use the BAR_PERFORMANCE configuration parameter to specify the type of performance statistics to report to the ON-Bar activity log for backup and restore operations.

onconfig.std *value*
0
units
Levels of statistics
values
0 = Does not collect performance statistics
1 = Reports time spent transferring data between the database server and the storage manager.
2 = Reports ON-Bar processing performance, in microseconds, in the timestamps in the activity log and the error log
3 = Reports both microsecond timestamps and transfer statistics.
takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

For example, if you set BAR_PERFORMANCE to 3, ON-Bar reports the time spent transferring data between the IBM® Informix® instance and the storage manager, in the activity log. If you set BAR_PERFORMANCE to 0 or do not set it, ON-Bar does not report performance statistics.

- To turn performance monitoring off, set the value to 0. This is the default.
- To display the time spent transferring data between the instance and the storage manager, set the parameter to 1.
- To display timestamps in microseconds, set the parameter to 2.
- To display both timestamps and transfer statistics, set the parameter to 3.

Related reference:
View ON-Bar backup and restore performance statistics
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_PROGRESS_FREQ configuration parameter

Use the BAR_PROGRESS_FREQ configuration parameter to specify, in minutes, the frequency of the progress messages in the ON-Bar activity log for backup and restore operations.

onconfig.std *value*
0
if value not present
0
units
minutes
range of values
0, then 5 to unlimited
takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

Example: If you set BAR_PROGRESS_FREQ to 5, ON-Bar reports the percentage of the object backed up or restored every 5 minutes. If you set BAR_PROGRESS_FREQ to 0 or do not set it, ON-Bar does not write any progress messages to the activity log.

Specify a value 5 minutes or over. Do not set BAR_PROGRESS_FREQ to 1, 2, 3, or 4, ON-Bar automatically resets it to 5 to prevent overflow in the ON-Bar activity log.

If ON-Bar cannot determine the size of the backup or restore object, it reports the number of transfer buffers sent to the database server instead of the percentage of the object backed up or restored.

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_RETRY configuration parameter

Use the BAR_RETRY configuration parameter to specify how many times **onbar** should try a data backup, logical-log backup, or restore operation if the first attempt fails.

onconfig.std *value*
1
if value not present
1
units
integer
range of values
0 = BAR_ABORT, stop the rest of the backup/restore
1 = BAR_CONT, continue the rest of the backup/restore
n = 2 to 32766
takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

The setting of the BAR_RETRY parameter determines ON-Bar behavior in the following ways:

- If set to 0 (BAR_ABORT), ON-Bar stops the backup or restore session when an error occurs for a storage space or logical log, returns an error, and quits. If ON-Bar is running in parallel, the already running processes finish but no new ones are started.
- If set to 1 (BAR_CONT), ON-Bar stops the backup or restore attempt for that particular storage space, returns an error, and attempts to back up or restore any storage spaces or logical logs that remain.
- If set to a specific number (retry backup and restore operations 2 to 32766 times), ON-Bar attempts to back up or restore this storage space or logical log the specified number of times before it gives up and moves on to the next one.

Related reference:
onbar -RESTART syntax: Restarting a failed restore
Resolve a failed restore
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_SEC_ALLOW_BACKUP configuration parameter

Use the BAR_SEC_ALLOW_BACKUP configuration parameter on an RSS node to enable the taking of archives and log backups on that node using either **onbar** or **ontape**.

onconfig.std *value*
0
default value
0
range of values
0 = Do not allow archives or log backups to be taken on this RSS node.
1 = Allow archives and log backups to be taken on this RSS node.
takes effect
When the RSS starts

Usage

The setting of the BAR_SEC_ALLOW_BACKUP parameter on an RSS node determines whether the **ontape** or **onbar** utilities may take archives or log backups on that node.

- If set to 0, **ontape** and **onbar** will return an error if one attempts to archive spaces or back up logical logs.
- If set to 1 and the RSS node has been properly configured, either **ontape** or **onbar** may be used to take archives and log backups locally.

The value of BAR_SEC_ALLOW_BACKUP is ignored on any non-RSS node.

Related concepts:
Backup and restore a Remote Secondary Server(RSS)

BAR_SIZE_FACTOR configuration parameter

Use the BAR_SIZE_FACTOR configuration parameter to augment the estimate for the size of a backup object, before the backup.

onconfig.std value
Not in onconfig.std.
default value
0
range of values
0 = The estimated size of the backup is not augmented.
Positive integers = The percentage of the original backup size.
takes effect
When the database server starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

The estimate is handled before the backup and is calculated so that the storage manager can allocate the storage media appropriately. Because the backup is done online, the number of pages to back up can change during the backup. Some storage managers are strict and if the backup estimate is too low, the backup results in an error.

The value of BAR_SIZE_FACTOR is taken as percentage of the original backup object size, and then added to the estimate, before communicating it to the storage manager. BAR_SIZE_FACTOR is used only for dbspace backup objects, not for logical log backup objects.

The formula used for calculating the new estimated backup object size is:

```
new_estimate = original_estimate x (1 + (BAR_SIZE_FACTOR / 100))
```

The value to which you set this parameter in a specific server environment depends on the activity on the system during backup or archive. Therefore, determining the value needs to be based on the individual experience with that system.
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

BAR_XFER_BUF_SIZE configuration parameter

Use the BAR_XFER_BUF_SIZE configuration parameter to specify the size of each transfer buffer.

onconfig.std value
31 if the page size is 2 KB
15 if the page size is 4 KB
units
pages
range of values
For storage managers that support long transfer buffers:

- 1 - 16383 pages when the page size is 4 KB
- 1 - 32766 pages when the page size is 2 KB

For storage managers that do not support long transfer buffers:

- 1 - 15 if the database server base page size is 4 KB
- 1 - 31 if the database server base page size is 2 KB

takes effect
When ON-Bar starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** or equivalent SQL administration API command.

Usage

The database server passes the transfer buffer to ON-Bar and the storage manager.

The value of BAR_XFER_BUF_SIZE is in database server base page sizes. For Linux, Solaris, and HP, the database server base page size is 2 KB; and for AIX®, Windows, and Mac, the database server base page size is 4 KB. To calculate the size of the transfer buffer in a storage space or logical-log backup, multiply the value of the BAR_XFER_BUF_SIZE configuration parameter by the system page size, as shown in the following formula:

```
one transfer buffer KB = BAR_XFER_BUF_SIZE * pagesize
```

You can determine the system page size by running the **onstat -b** command.
The maximum size of the transfer buffer for many storage managers is 64 KB. IBM® Spectrum Protect and IBM Informix® Primary Storage Manager support long transfer buffer sizes of up to 65532 KB.

To calculate how much memory, in KB, the database server needs for each transfer buffer, use the following formula:

```
memory KB = (BAR_XFER_BUF_SIZE * pagesize) + 500 bytes/1028
```

The extra 500 bytes is for system use. For example, if BAR_XFER_BUF_SIZE is 15, the transfer buffer can be 66,292 bytes, or 64.5 KB.

The number of transfer buffers per backup stream is specified by the value of the BAR_NB_XPORT_COUNT configuration parameter, and the number of parallel backup streams is specified by the BAR_MAX_BACKUP configuration parameter.

Restriction: You cannot change the buffer size between a backup and restore. The values of the AC_TAPEBLOCK and AC_LTAPEBLOCK configuration parameters must be the same value as the value of the BAR_XFER_BUF_SIZE configuration parameter was at the time of backup.

Example

For example, for a transfer buffer size of 128*2 KB (a value of 256 KB) on Linux, specify:

```
BAR_XFER_BUF_SIZE  128
```

Related concepts:
Configuring ON-Bar for optional Spectrum Protect features
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
onstat -b command: Print buffer information for buffers in use

IFX_BAR_NO_BSA_PROVIDER environment variable

Set the IFX_BAR_NO_BSA_PROVIDER environment variable to force ON-Bar to use the sm_versions file as the source of information about the XBSA library for the storage manager.

```
>>-setenv--IFX_BAR_NO_BSA_PROVIDER--1-------------------------><
```

By default, ON-Bar communicates directly with the XBSA library for some storage managers. ON-Bar does not require that the sm_versions file is updated to contain information about the XBSA library for those storage managers.

Set the IFX_BAR_NO_BSA_PROVIDER environment variable if you are instructed to do so by Software Support.

To unset the IFX_BAR_NO_BSA_PROVIDER environment variable, run the following command:

```
unset IFX_BAR_NO_BSA_PROVIDER
```

IFX_BAR_NO_LONG_BUFFERS environment variable

Set the IFX_BAR_NO_LONG_BUFFERS environment variable to prevent the size of transfer buffers from exceeding 64 KB when the BAR_XFER_BUF_SIZE configuration parameter is set to a long transfer buffer size value.

```
>>-setenv--IFX_BAR_NO_LONG_BUFFERS--1-------------------------><
```

IBM® Spectrum Protect and IBM Informix® Primary Storage Manager support long transfer buffer sizes of up to 65532 KB.

Set the IFX_BAR_NO_LONG_BUFFERS environment variable if you are instructed to do so by Software Support.

To unset the IFX_BAR_NO_LONG_BUFFERS environment variable, run the following command:

```
unset IFX_BAR_NO_LONG_BUFFERS
```

Related concepts:
Configuring ON-Bar for optional Spectrum Protect features

IFX_BAR_USE_DEDUP environment variable

Set the IFX_BAR_USE_DEDUP environment variable to optimize the deduplication capabilities of storage managers.

```
>>-setenv--IFX_BAR_USE_DEDUP----------------------------------><
```

You are not required to set the IFX_BAR_USE_DEDUP environment variable to a value. You can set it to any value or to no value.

Deduplication is a storage manager feature that reduces the size of backups by removing data that exists in older backups. To use deduplication, enable deduplication for your storage manager and set the IFX_BAR_USE_DEDUP environment variable in the database server environment and then restart the database server. When you set the IFX_BAR_USE_DEDUP environment variable, the backup format has fewer unique pages, which optimizes the deduplication process.

Deduplication is incompatible with incremental backups. Do not make incremental backups while the IFX_BAR_USE_DEDUP environment variable is set.

Important: Backups that you take while the IFX_BAR_USE_DEDUP environment variable is set must be restored while the IFX_BAR_USE_DEDUP environment variable is set.
IBM® Informix® Primary Storage Manager does not support deduplication.

To unset the IFX_BAR_USE_DEDUP environment variable, run the following command:

```
unset IFX_BAR_USE_DEDUP
```

After you unset the IFX_BAR_USE_DEDUP environment variable, you must perform a level-0 backup.

Related concepts:
Configuring ON-Bar for optional Spectrum Protect features
Related tasks:
Configuring a third-party storage manager
Related reference:
Editing the Spectrum Protect client system options file
Configure ontape

IFX_TSM_OBJINFO_OFF environment variable

Set the IFX_TSM_OBJINFO_OFF environment variable to disable support for restoring backup objects that are replicated, imported, or exported between IBM® Spectrum Protect servers.

```
>>-setenv--IFX_TSM_OBJINFO_OFF--1------------------------------><
```

By default, ON-Bar stores unique IDs in the metadata of backup objects that are created by Spectrum Protect. During a restore, ON-Bar checks the metadata to identify the object. Therefore, ON-Bar can restore a backup whose original ID, which is stored as CopyID columns in the ON-Bar catalog, changed because the object was replicated, exported, or imported between Spectrum Protect servers. To prevent the ability to restore backup objects that are moved between Spectrum Protect servers, set the IFX_TSM_OBJINFO_OFF environment variable to 1.

To unset the IFX_TSM_OBJINFO_OFF environment variable, run the following command:

```
unset IFX_TSM_OBJINFO_OFF
```

Related concepts:
Configuring ON-Bar for optional Spectrum Protect features

LTAPEBLK configuration parameter

Use the LTAPEBLK configuration parameter to specify the block size of the device to which the logical logs are backed up when you use **ontape** for dbspace backups.

LTAPEBLK also specifies the block size for the device to which data is loaded or unloaded when you use the -l option of **onload** or **onunload**. If you are using **onload** or **onunload**, you can specify a different block size at the command line.

onconfig.std *value*

- On UNIX: 32
- On Windows: 16

units
Kilobytes
range of values
Values greater than (*page size*/1024)
To obtain the page size, run the **onstat -b** command.

takes effect
For **ontape**:

- When you execute **ontape**.
- When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

For **onload** and **onunload**: When the database server is shut down and restarted

Usage

Specify LTAPEBLK as the largest block size permitted by your tape device. The database server does not check the tape device when you specify the block size. Verify that the LTAPEDEV tape device can read the block size that you specify. If not, you might not be able to read from the tape.

UNIX only: The UNIX **dd** utility can verify that the LTAPEDEV tape device can read the block size. It is available with most UNIX systems.

If you specify a LTAPEBLK value, ON-Bar ignores the value.

Related reference:
Changing your ontape configuration
LTAPEDEV configuration parameter
LTAPESIZE configuration parameter
ontape backup and restore system
TAPEBLK configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
The onunload and onload utilities

LTAPEDEV configuration parameter

Use the LTAPEDEV configuration parameter to specify the device or directory file system to which the logical logs are backed up when you use **ontape** for backups.

The LTAPEDEV configuration parameter also specifies the device to which data is loaded or unloaded when you use the **-l** option of **onload** or **onunload**. If you are using LTAPEDEV to specify a device for **onunload** or **onload**, the same information for TAPEDEV is relevant for LTAPEDEV.

onconfig.std *value*
On UNIX: **/dev/tapedev**On Windows: **NUL**
if not present
On UNIX: **/dev/null**On Windows:**NUL**
takes effect
For **ontape**:

- When you execute **ontape**, if set to a tape device.
- When the database server is shut down and restarted, if set to **/dev/null** on UNIX or **nul** on Windows.
- When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
- When you reset the value in memory by running the **onmode -wm** command.

For **onload** and **onunload**: When the database server is shut down and restarted

Usage

Warning: Do not set LTAPEDEV to **/dev/null** or **nul** when you use ON-Bar to back up logical logs.
If you specify a tape device in the LTAPEDEV configuration parameter, ON-Bar ignores the value.
Important: Set LTAPEDEV to /dev/null or leave it blank on UNIX or NUL on Windows only if you do not want to back up the logical logs. You must take the database server offline before you change the value of LTAPEDEV to /dev/null.
When you set LTAPEDEV to /dev/null:

- The database server frees the logical logs without requiring that you back up those logs. The logical logs do not get marked as free, but the database server can reuse them.
- The ON-Bar activity log shows a warning and return code 152. Because the database server marks the logical logs as backed up when they are no longer current, ON-Bar cannot find logical logs to back up. All transactions in those logs are lost, and you are not able to restore them.

If you performed a whole-system backup with LTAPEDEV set to null, you must use the **onbar -r -w -p** command during restore to notify ON-Bar that you do not want to restore the logs. .

Related reference:
Changing your ontape configuration
LTAPEBLK configuration parameter
ontape backup and restore system
TAPEDEV configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
The onunload and onload utilities

LTAPESIZE configuration parameter

Use the LTAPESIZE configuration parameter to specify the maximum tape size of the device to which the logical logs are backed up when you use **ontape** for backups.

The LTAPESIZE configuration parameter also specifies the maximum tape size of the device to which data is loaded or unloaded when you use the **-l** option of the **onload** or **onunload** utility. If you are using **onload** or **onunload**, you can specify a different tape size on the command line. If you want to use the full capacity of a tape, set

LTAPESIZE to 0.

onconfig.std value
: LTAPESIZE 0

units
: KB

range of values
: 0 - 9223372036854775807 (9 ZB)

takes effect
: For the **ontape** utility:

  - When you run an **ontape** command.
  - When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

  For the **onload** and **onunload** utilities: When the database server is shut down and restarted

Usage

LTAPESIZE specifies the maximum tape size of the device to which the logical logs are backed up when you run the **ontape** utility for backups. LTAPESIZE also specifies the maximum tape size of the device to which data is loaded or unloaded when you use the **-l** option of the **onload** or **onunload** utility. If you are using the **onload** or **onunload** utility, you can specify a different tape size on the command line. If you want to use the full capacity of a tape, set LTAPESIZE to 0.

Note: If the BACKUP_FILTER parameter is set in the ONCONFIG file, the LTAPESIZE cannot be set to 0. Otherwise the **ontape** utility returns an error when backing up logical logs to a directory on disk. The error message is:

```
The LTAPESIZE configuration parameter cannot be set to 0 when the BACKUP_FILTER
configuration parameter is set; change the value of LTAPESIZE.
Program over.
```

A workaround is to set the LTAPESIZE configuration parameter to a very high value. Log files are not much higher than the LOGSIZE configuration parameter. Use the value in the LOGSIZE as the upper limit for this database.

If you specify a LTAPESIZE value, ON-Bar ignores the value.

Related reference:
Changing your ontape configuration
Specify the tape size
LTAPEBLK configuration parameter
ontape backup and restore system
TAPESIZE configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
The onunload and onload utilities

RESTARTABLE_RESTORE configuration parameter

Use the RESTARTABLE_RESTORE configuration parameter to enable or disable restartable restores.

onconfig.std value
: RESTARTABLE_RESTORE ON

values
: OFF
  : Disables restartable restore. If a restore fails and RESTARTABLE_RESTORE is OFF, you are not able to restart it.
  ON
  : Enables restartable restore. Set RESTARTABLE_RESTORE to ON before you begin a restore. Otherwise, you will be unable to restart the restore after a failure.

takes effect
: After you edit your onconfig file. If you need to restart a physical restore, you do not need to restart the database server before you can use RESTARTABLE_RESTORE. If you need to restart a logical restore, you must restart the database server before you can use restartable restore.

Turning on RESTARTABLE_RESTORE slows down logical restore performance. For more information, see onbar -RESTART syntax: Restarting a failed restore.

Related reference:
onbar -RESTART syntax: Restarting a failed restore
Resolve a failed restore

RESTORE_FILTER configuration parameter

Use the RESTORE_FILTER configuration parameter to specify the path name of a filter program, and any options.

onconfig.std value
: Not set. Restored data is not filtered.

values
: The path name of a command and any options. By default, the path name is relative to the $INFORMIXDIR/bin directory, otherwise, the path name must be the absolute path of the program. If you include command-line options, both the filter name and the options must be surrounded by single quotation marks.

takes effect
: After you edit your onconfig file and ON-Bar or **ontape** starts.

Usage

This filter transforms data that was transformed during backup to its original format prior to a restore. The filter specified by the RESTORE_FILTER configuration parameter must match the filter specified by the BACKUP_FILTER configuration parameter. For example, if data was compressed during backup, data must be uncompressed during a restore.

For security purposes, filters should not have write permission to non-privileged users. Permission on the filters are the same as that of permission on other executable files that are called by the IBM® Informix® server or utilities.

For example, if you want to compress backed up data, you could set the BACKUP_FILTER and RESTORE_FILTER configuration parameters to the following values:

```
BACKUP_FILTER /bin/compress
RESTORE_FILTER /bin/uncompress
```

The RESTORE_FILTER configuration parameter can include command-line options as well as the filter name. For example, specify:

```
RESTORE_FILTER  'my_decrypt -file /var/adm/encryption.pass'
```

In this example, the command in quotation marks is used as the filter.

Related reference:
BACKUP_FILTER configuration parameter

TAPEBLK configuration parameter

Use the TAPEBLK configuration parameter to specify the block size of the device to which **ontape** writes during a storage-space backup.

onconfig.std *value*

- On UNIX: 32
- On Windows: 16

units
Kilobytes
range of values
Values greater than `pagesize/1024`
To obtain the page size, run the **onstat -b** command.

takes effect
For **ontape**:

- When you execute **ontape**.
- When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
- When you reset the value in memory by running the **onmode -wm** command.

For **onload** and **onunload**: When the database server is shut down and restarted

Usage

TAPEBLK also specifies the default block size of the device to which data is loaded or unloaded when you use the **onload** or **onunload** utilities. If you are using **onload** or **onunload**, you can specify a different block size on the command line.

The database server does not check the tape device when you specify the block size. Verify that the TAPEBLK tape device can read the block size that you specify. If not, you might not able to read from the tape.

If you specify a TAPEBLK value, ON-Bar ignores the value.

Related reference:
Changing your ontape configuration
TAPEDEV configuration parameter
TAPESIZE configuration parameter
LTAPEBLK configuration parameter
ontape backup and restore system
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
The onunload and onload utilities

TAPEDEV configuration parameter

Use the TAPEDEV configuration parameter to specify the device or directory file system to which the **ontape** utility backs up storage spaces.

onconfig.std ***value***
On UNIX: **/dev/tapedev**
On Windows: **\\.\TAPE0**

if not present
On UNIX: **/dev/null**
On Windows: **NUL**

units
Path name
takes effect
For the **ontape** utility:

- If it is set to /dev/null on UNIX or NUL on Windows, when the database server is shut down and restarted
- If it is set to a tape device, when you run the **ontape** utility
- When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
- When you reset the value in memory by running the **onmode -wm** command.

For the **onload** and **onunload** utilities: When the database server is shut down and restarted

Usage

The **ontape** utility reads the value of the TAPEDEV parameter at the start of processing. If you set TAPEDEV to /dev/null, you must do it before you start **ontape** to request the backup. When you set TAPEDEV to /dev/null and request a backup, the database server bypasses the backup but still updates the dbspaces with the new backup time stamps.

You can set the TAPEDEV configuration parameter to `STDIO` to direct **ontape** utility back up and restore operations to standard I/O instead of to a device.

The TAPEDEV configuration parameter also specifies the default device to which data is loaded or unloaded when you use the **onload** or **onunload** utilities. However, if TAPEDEV is set to `STDIO`, the **onunload** utility will not be able to unload data.

If you change the tape device, verify that the TAPEBLK and TAPESIZE configuration parameter values are correct for the new device.

If you specify a TAPEDEV value, ON-Bar ignores the value.

Remote devices (UNIX)

You can perform a storage-space backup across your network to a remote device attached to another host computer on UNIX and Linux platforms. The remote device and the database server computer must have a trusted relationship so that the **rsh** or the **rlogin** utility can connect from the database server computer to the remote device computer without asking for password. You can establish a trusted relationship by configuring the /etc/hosts.equiv file, the user's ~/.rhosts files, or any equivalent mechanism for your system on the remote device computer. If you want to use a different utility to handle the remote session than the default utility used by your platform, you can set the DBREMOTECMD environment variable to the specific utility that you want to use.

Symbolic links to remote devices (UNIX)

The TAPEDEV configuration parameter can be a symbolic link, enabling you to switch between tape devices without changing the path name that the TAPEDEV configuration parameter specifies.

Use the following syntax to specify a tape device attached to another host computer:

```
host_machine_name:tape_device_pathname
```

The following example specifies a tape device on the host computer **kyoto**:

```
kyoto:/dev/rmt01
```

Rewinding tape devices before opening and on closing

The tape device that The TAPEDEV configuration parameter specifies must perform a rewind before it opens and when it closes. The database server requires this action because of a series of checks that it performs before it writes to a tape.

When the database server attempts to write to any tape other than the first tape in a multivolume dbspace or logical-log backup, the database server first reads the tape header to make sure that the tape is available for use. Then the device is closed and reopened. The database server assumes the tape was rewound when it closed, and the database server begins to write.

Whenever the database server attempts to read a tape, it first reads the header and looks for the correct information. The database server does not find the correct header information at the start of the tape if the tape device did not rewind when it closed during the write process.

Related reference:
Changing your ontape configuration
TAPEBLK configuration parameter
LTAPEDEV configuration parameter
ontape backup and restore system
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
The onunload and onload utilities

TAPESIZE configuration parameter

Use the TAPESIZE parameter specifies the size of the device to which the **ontape** utility backs up storage spaces.

onconfig.std value
TAPESIZE 0
units
KB
range of values

0 - 9223372036854775807 (9 ZB)

takes effect

For **ontape**:

- When you run an **ontape** command.
- When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
- When you reset the value in memory by running the **onmode -wm** command.

For the **onload** and **onunload** utilities: When the database server is shut down and restarted.

Usage

The TAPESIZE also specifies the size of the default device to which data is loaded or unloaded when you use the **onload** or **onunload** utility. When you run the **onload** or **onunload** utility, you can specify a different tape size on the command line. If you want to use the full physical capacity of a tape, set TAPESIZE to `0`.

Note: Tape size is irrelevant if TAPEDEV is set to `STDIO`.
If you specify a TAPESIZE value, ON-Bar ignores the value.

Related reference:
Changing your ontape configuration
Specify the tape size
TAPEBLK configuration parameter
ontape backup and restore system
LTAPESIZE configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
The onunload and onload utilities

The archecker utility configuration parameters and environment variable

These topics describe the AC_CONFIG environment variable and the configuration parameters that you use with the **archecker** utility.

The **archecker** utility uses the configuration parameters in the ac_config.std template to verify a backup or perform a table-level restore. If you need to change these parameters, copy the ac_config.std template to the AC_CONFIG file. The AC_CONFIG environment variable specifies the location of the AC_CONFIG file.

Because ON-Bar calls the **archecker** utility to verify backups, you must configure the **archecker** environment variable and parameters before you can use the **onbar -v** option.

You can also use other **archecker** configuration parameters that do not have default have default values in the ac_config.std file, but are valid in that file.

Table 1. Configuration parameters that the **archecker** utility uses

Configuration parameter	Description
AC_DEBUG	Prints debugging messages in the **archecker** message log.
AC_IXBAR	Specifies the path name to the IXBAR file. If not set in the ac_config file, the value of the BAR_IXBAR_PATH configuration parameter is used.
AC_LTAPEBLOCK	Specifies the **ontape** block size for reading logical logs. If not set in the ac_config file, the value of the LTAPEBLOCK configuration parameter is used.
AC_LTAPEDEV	Specifies the local device name used by **ontape** for reading logical logs. If not set in the ac_config file, the value of the LTAPEDEV configuration parameter is used.
AC_MSGPATH	Specifies the location of the **archecker** message log. This configuration parameter is in the default ac_config file.
AC_SCHEMA	Specifies the path name to the **archecker** schema command file.
AC_STORAGE	Specifies the location of the temporary files that **archecker** builds. This configuration parameter is in the default ac_config file.
AC_TAPEBLOCK	Specifies the tape block size in kilobytes. If not set in the ac_config file, the value of the TAPEBLOCK configuration parameter is used.
AC_TAPEDEV	Specifies the local device name used by the **ontape** utility. If not set in the ac_config file, the value of the TAPEDEV configuration parameter is used.
AC_TIMEOUT	Specifies the timeout value for the **onbar** and the **archecker** processes if one of them exits prematurely.
AC_VERBOSE	Specifies either verbose or terse mode for **archecker** messages. This configuration parameter is in the default ac_config file.
BAR_BSALIB_PATH	Identical to the BAR_BSALIB_PATH server configuration parameter that is in the onconfig.std file. For more information, see BAR_BSALIB_PATH configuration parameter.

If you use the **ontape** utility and the AC_TAPEDEV, AC_TAPEBLK, AC_LTAPEDEV and AC_LTAPEBLK configuration parameters are not set in the AC_CONFIG file, the **archecker** utility will use the values specified in the TAPEDEV, TAPEBLK, LTAPEDEV, LTAPEBLK configuration parameters specified in the onconfig file.

- AC_CONFIG file environment variable
  Set the AC_CONFIG environment variable to the full path name for the **archecker** configuration file (either ac_config.std or user defined).
- AC_DEBUG configuration parameter
  The AC_DEBUG configuration parameter causes debugging messages to be printed in the **archecker** message file. Use this parameter only as directed by technical support.

- AC_IXBAR configuration parameter
  Use the AC_IXBAR configuration parameter to specify the location of the IXBAR file.
- AC_LTAPEBLOCK configuration parameter
  Use the AC_LTAPEBLOCK configuration parameter to the **ontape** block size for reading logical logs.
- AC_LTAPEDEV parameter
  Use the AC_LTAPEDEV configuration parameter to specify the local device name that is used by the **ontape** utility.
- AC_MSGPATH configuration parameter
  Use the AC_MSGPATH parameter in the AC_CONFIG file to specify the location of the **archecker** message log (ac_msg.log).
- AC_SCHEMA configuration parameter
  Use the AC_SCHEMA configuration parameter to specify the path name to the **archecker** schema command file.
- AC_STORAGE configuration parameter
  Use the AC_STORAGE configuration parameter in the AC_CONFIG file to specify the location of the directory where **archecker** stores its temporary files.
- AC_TAPEBLOCK configuration parameter
  Use the AC_TAPEBLOCK configuration parameter to specify the size of the tape block in kilobytes when an archive is performed either the **onbar -b** command or the **ontape -t** command.
- AC_TAPEDEV configuration parameter
  Use the AC_TAPEDEV configuration parameter to specify the local device name that is used by the **ontape** utility.
- AC_TIMEOUT configuration parameter
  Use the AC_TIMEOUT configuration parameter to specify the timeout value for the **onbar** and the **archecker** processes if one of them exits prematurely.
- AC_VERBOSE configuration parameter
  Use the AC_VERBOSE parameter in the AC_CONFIG file to specify either verbose or terse output in the **archecker** message log (ac_msg.log).

Related reference:
archecker table level restore utility

AC_CONFIG file environment variable

Set the AC_CONFIG environment variable to the full path name for the **archecker** configuration file (either ac_config.std or user defined).

```
>>-setenv--AC_CONFIG -pathname--------------------------------><
```

default *value*
UNIX: $INFORMIXDIR/etc/ac_config.std
Windows: %INFORMIXDIR%\etc\ac_config.std

takes effect
When ON-Bar starts

The following are examples of valid AC_CONFIG path names:

- UNIX: /usr/dbserver/etc/ac_config.std and /usr/local/my_ac_config.std
- Windows: c:\dbserver\etc\ac_config.std and c:\dbserver\etc\my_ac_config.std

If AC_CONFIG is not set, the **archecker** utility sets the default location for the **archecker** configuration file to $INFORMIXDIR/etc/ac_config.std on UNIX or %INFORMIXDIR%\etc\ac_config.std on Windows.

Important: If you do not specify the entire path, including the configuration file name in the AC_CONFIG file, the **archecker** utility might not work correctly.

AC_DEBUG configuration parameter

The AC_DEBUG configuration parameter causes debugging messages to be printed in the **archecker** message file. Use this parameter only as directed by technical support.

The use of this configuration parameter can cause the **archecker** message log file to grow very large and can substantially slow down **archecker** processing.

Default value
Off
Range
1-16

AC_IXBAR configuration parameter

Use the AC_IXBAR configuration parameter to specify the location of the IXBAR file.

Default value
None
Range
Any valid path name

AC_LTAPEBLOCK configuration parameter

Use the AC_LTAPEBLOCK configuration parameter to the **ontape** block size for reading logical logs.

Default value
: 32 kilobytes

Range
: 0 - 2,000,000,000

Usage

When you perform an archive with:

- **onbar -b**, the value of AC_TAPEBLOCK should be the value the BAR_XFER_BUF_SIZE configuration parameter multiplied by the current page size. For more information, see BAR_XFER_BUF_SIZE configuration parameter.
- **ontape -t**, the value of AC_LTAPEBLOCK should be the value that the TAPEBLK ONCONFIG configuration parameter was set to at the time of the archive. For more information, see Specify the tape-block-size.

AC_LTAPEDEV parameter

Use the AC_LTAPEDEV configuration parameter to specify the local device name that is used by the **ontape** utility.

If the tape device is set to STDIO, **archecker** receives input from standard input.

Default value
: None

Range
: Any valid path name or STDIO

AC_MSGPATH configuration parameter

Use the AC_MSGPATH parameter in the AC_CONFIG file to specify the location of the **archecker** message log (ac_msg.log).

ac_config.std value
: UNIX: AC_MSGPATH /tmp/ac_msg.log
: Windows: AC_MSGPATH c:\temp\ac_msg.log

takes effect
: When ON-Bar starts

Usage

You must specify the entire path of the message log in the AC_CONFIG file or else the **archecker** utility might not work correctly.

When you verify backups with **onbar -v**, the **archecker** utility writes summary messages to the bar_act.log and indicates whether the verification succeeded or failed. It writes detailed messages to the ac_msg.log. If the backup fails verification, discard the backup and try another backup, or give the ac_msg.log to Software Support. For sample messages, see onbar -v syntax: Verifying backups.

AC_SCHEMA configuration parameter

Use the AC_SCHEMA configuration parameter to specify the path name to the **archecker** schema command file.

Default value
: None

Range
: Any valid path name

This configuration parameter is overridden by the -f *cmdfile* command line option.

AC_STORAGE configuration parameter

Use the AC_STORAGE configuration parameter in the AC_CONFIG file to specify the location of the directory where **archecker** stores its temporary files.

ac_config.std *value*
: UNIX: /tmp
: Windows: c:\temp

takes effect
: When ON-Bar starts

Usage

You must specify the entire path of the storage location in the AC_CONFIG file or else the **archecker** utility might not work correctly.

The following table lists the directories and files that **archecker** builds. If verification is successful, these files are deleted.

Table 1. The archecker temporary files

Directory	Files
CHUNK_BM	Bitmap information for every backed up storage space.
INFO	Statistical analysis and debugging information for the backup.
SAVE	Partition pages in the PT.######## file. Chunk-free pages in the FL.######## file. Reserved pages in the RS.######## file. Blob-free map pages in the BF.######## file

To calculate the amount of free space that you need, see Temporary space for backup verification. It is recommended that you set AC_STORAGE to a location with plenty of free space.

AC_TAPEBLOCK configuration parameter

Use the AC_TAPEBLOCK configuration parameter to specify the size of the tape block in kilobytes when an archive is performed either the **onbar -b** command or the **ontape -t** command.

Default value
: 32 kilobytes

Range
: 0 - 2,000,000,000

Usage

When you perform an archive with:

- **onbar -b**, the value of AC_TAPEBLOCK should be the value the BAR_XFER_BUF_SIZE configuration parameter multiplied by the current page size. For more information, see BAR_XFER_BUF_SIZE configuration parameter.
- **ontape -t**, the value of AC_TAPEBLOCK should be the value that the TAPEBLK ONCONFIG configuration parameter was set to at the time of the archive. For more information, see Specify the tape-block-size.

AC_TAPEDEV configuration parameter

Use the AC_TAPEDEV configuration parameter to specify the local device name that is used by the **ontape** utility.

If the tape device is set to STDIO, **archecker** receives input from standard input.

Default value
: None

Range
: Any valid path name or STDIO

AC_TIMEOUT configuration parameter

Use the AC_TIMEOUT configuration parameter to specify the timeout value for the **onbar** and the **archecker** processes if one of them exits prematurely.

ac_config.std *value*
: UNIX: 300
: Windows: 300

units
: seconds

takes effect
: When the **onbar-v** command starts

The AC_TIMEOUT configuration parameter was introduced to avoid **onbar** and **archecker** processes waiting for each other indefinitely if one of them exits prematurely, thus avoiding the creation of an orphan and zombie process during data server initialization.

AC_VERBOSE configuration parameter

Use the AC_VERBOSE parameter in the AC_CONFIG file to specify either verbose or terse output in the **archecker** message log (ac_msg.log).

ac_config.std *value*
: 1

range of values
: 1 = verbose messages in ac_msg.log
: 0 = terse messages in ac_msg.log

takes effect
When ON-Bar starts

Informix Primary Storage Manager configuration parameters

The IBM® Informix® Primary Storage Manager uses the information in some specific configuration parameters.

- PSM_ACT_LOG configuration parameter
  Use the PSM_ACT_LOG configuration parameter to specify the location of the IBM Informix Primary Storage Manager activity log if you do not want the log information included in the ON-Bar activity log.
- PSM_CATALOG_PATH configuration parameter
  Use the PSM_CATALOG_PATH configuration parameter to specify the full path to the directory that contains the IBM Informix Primary Storage Manager catalog tables. These catalog tables contain information about the pools, devices, and objects managed by the storage manager.
- PSM_DBS_POOL configuration parameter
  Use the PSM_DBS_POOL configuration parameter to change the name of the pool in which the IBM Informix Primary Storage Manager places backup and restore dbspace data.
- PSM_DEBUG configuration parameter
  Use the PSM_DEBUG configuration parameter to specify the amount of debugging information that prints in the Informix Primary Storage Manager debug log if you want to use a debug level that is different from the one used by ON-Bar.
- PSM_DEBUG_LOG configuration parameter
  Use the PSM_DEBUG_LOG configuration parameter to specify the location of the debug log to which the IBM Informix Primary Storage Manager writes debugging messages if you do not want the log information included in the ON-Bar debug log.
- PSM_LOG_POOL configuration parameter
  Use the PSM_LOG_POOL configuration parameter to change the name of the pool in which the IBM Informix Primary Storage Manager places backup and restore log data.

Related concepts:
IBM Informix Primary Storage Manager
Setting up Informix Primary Storage Manager
Related tasks:
Configuring Informix Primary Storage Manager
Examples: Manage storage devices with Informix Primary Storage Manager

PSM_ACT_LOG configuration parameter

Use the PSM_ACT_LOG configuration parameter to specify the location of the IBM® Informix® Primary Storage Manager activity log if you do not want the log information included in the ON-Bar activity log.

onconfig.std *value*
none
if value not present
The value of the BAR_ACT_LOG configuration parameter is used
range of values
Full path name
takes effect
When the **onpsm** utility starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

Usage

Specify a path to an existing directory with an appropriate amount of space available or use $INFORMIXDIR/psm_act.log. If you specify a file name only, the storage manager creates the activity log in the working directory in which you started the storage manager.

If the PSM_ACT_LOG configuration parameter is not set, the Informix Primary Storage Manager puts activity information in the directory specified with the BAR_ACT_LOG configuration parameter. To clearly distinguish ON-Bar and Informix Primary Storage Manager activity information, use the PSM_ACT_LOG to specify a different location for the storage manager activity log.

The format of the file resembles the format of the database server message log. You can examine the activity log to determine the results of storage manager actions.

The file specified by the PSM_ACT_LOG configuration parameter is created if it does not exist.

You can also use the PSM_ACT_LOG environment variable to specify the location of the Informix Primary Storage Manager activity log for your environment, for example, for a single session.

Related reference:
Message logs for Informix Primary Storage Manager
BAR_ACT_LOG configuration parameter
Related information:
PSM_ACT_LOG environment variable
onmode -wf, -wm: Dynamically change certain configuration parameters

PSM_CATALOG_PATH configuration parameter

Use the PSM_CATALOG_PATH configuration parameter to specify the full path to the directory that contains the IBM® Informix® Primary Storage Manager catalog tables. These catalog tables contain information about the pools, devices, and objects managed by the storage manager.

onconfig.std value
: Not set. The default value is used.

default value
: UNIX or Linux: $INFORMIXDIR/etc/psm
: Windows: %INFORMIXDIR%\etc\psm

range of values
: Full path name for the directory that contains the Informix Primary Storage Manager catalog tables

takes effect
: When the ON-Bar or **onpsm** utility starts
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

You can change the default path to another location. The Informix Primary Storage Manager places the file that information about the devices and objects in whatever directory you specify.

If you move the backup file to another location, change the value of the PSM_CATALOG_PATH configuration parameter.

You can back up the contents of the file whenever you want.

If you have multiple instances, all instances contain the same catalog tables if the PSM_CATALOG_PATH in each instance is set to the same path. You can specify a different path for each instance.

The storage manager automatically creates the catalog tables the first time you run an **onpsm** utility command or the first time that the XBSA shared library is used.

You can also use the PSM_CATALOG_PATH environment variable to specify the location of the Informix Primary Storage Manager catalog tables for your environment, for example, for a single session.

Related information:
PSM_CATALOG_PATH environment variable
onmode -wf, -wm: Dynamically change certain configuration parameters

PSM_DBS_POOL configuration parameter

Use the PSM_DBS_POOL configuration parameter to change the name of the pool in which the IBM® Informix® Primary Storage Manager places backup and restore dbspace data.

onconfig.std *value*
: DBSPOOL

takes effect
: When the **onpsm** utility starts
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

The storage manager automatically places the dbspace data in the `DBSPOOL` or the pool you specify. You can use any combination of letters and digits.

You can also use the PSM_DBS_POOLenvironment variable to change the name of the pool for your environment, for example, for a single session.

Related concepts:
Device pools
Related information:
PSM_DBS_POOL environment variable
onmode -wf, -wm: Dynamically change certain configuration parameters

PSM_DEBUG configuration parameter

Use the PSM_DEBUG configuration parameter to specify the amount of debugging information that prints in the Informix® Primary Storage Manager debug log if you want to use a debug level that is different from the one used by ON-Bar.

onconfig.std value
: Not set. The default value is used.

default value
: The value of the BAR_DEBUG configuration parameter is used.

units
: One digit to represent the level of debugging information that you want

range of values
: `0` = No debugging messages.
: `1` = Prints only internal errors.
: `2` = Prints information about the entry and exit of functions and prints internal errors.
: `3` = Prints the information specified by `1`-`2` with additional details.

4 = Prints information about parallel operations and the information specified by 1-3.
5 = Prints information about internal states in the Informix Primary Storage Manager.
6 = Prints the information specified by 1-5 with additional details.
7 = Prints information specified by 1-6 with additional details.
8 = Prints information specified by 1-7 with additional details.
9 = Prints all debugging information.

takes effect

When the **onpsm** utility starts
When the ON-Bar utility executes commands and reads information that is specified in the BAR_DEBUG configuration parameter
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If you set the PSM_DEBUG configuration parameter to a valid value that is higher than 0, the Informix Primary Storage Manager logs debug messages to its debug log.

You can experiment with the debug values to find the right amount of information. Generally, if the PSM_DEBUG configuration parameter is set to 5, the storage manager prints enough information for tracing and debugging purposes.

Settings of 8 and 9 require a large amount of space.

You can also use the PSM_DEBUG environment variable to specify the amount of debugging information that prints in the storage manager debug log for your environment, for example, for a single session.

Related reference:
Message logs for Informix Primary Storage Manager
Related information:
PSM_DEBUG environment variable
onmode -wf, -wm: Dynamically change certain configuration parameters

PSM_DEBUG_LOG configuration parameter

Use the PSM_DEBUG_LOG configuration parameter to specify the location of the debug log to which the IBM® Informix® Primary Storage Manager writes debugging messages if you do not want the log information included in the ON-Bar debug log.

onconfig.std value

Not set. The default value is used.

default value

The value of the BAR_DEBUG_LOG configuration parameter is used.

takes effect

When the **onpsm** utility starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

If the PSM_DEBUG_LOG configuration parameter is not set, the Informix Primary Storage Manager puts activity information in the directory specified with the BAR_DEBUG_LOG configuration parameter. To clearly distinguish ON-Bar and Informix Primary Storage Manager activity information, use the PSM_DEBUG_LOG to specify a different location for the Informix Primary Storage Manager activity log.

For security reasons, set the PSM_DEBUG_LOG configuration parameter to a directory with restricted permissions, such as the $INFORMIXDIR directory.

If the directory that holds the debug file becomes too large, you can erase the file. You need to retain information only if there are problems that need to be debugged.

You can also use the PSM_DEBUG_LOG environment variable to specify the location of the debug log for your environment, for example, for a single session.

Related reference:
Message logs for Informix Primary Storage Manager
BAR_DEBUG_LOG configuration parameter
Related information:
PSM_DEBUG_LOG environment variable
onmode -wf, -wm: Dynamically change certain configuration parameters

PSM_LOG_POOL configuration parameter

Use the PSM_LOG_POOL configuration parameter to change the name of the pool in which the IBM® Informix® Primary Storage Manager places backup and restore log data.

onconfig.std *value*

LOGPOOL

takes effect

When the **onpsm** utility starts
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

The storage manager automatically places the log data in the `LOGPOOL` or the pool you specify. You can use any combination of letters and digits.

You can also use the PSM_LOG_POOL environment variable to change the name of the pool for your environment, for example, for a single session.

Related concepts:
Device pools
Related information:
PSM_LOG_POOL environment variable
onmode -wf, -wm: Dynamically change certain configuration parameters

Event alarm configuration parameters

When you set configuration parameters for use with the ON-Bar and **ontape** utilities, also determine if you need to adjust the ALARMPROGRAM and ALRM_ALL_EVENTS configuration parameters.

Use the ALARMPROGRAM configuration parameter to set the **log_full.sh** script to automatically back up log files when they become full.

Use the ALRM_ALL_EVENTS configuration parameter to cause ALARMPROGRAM to execute every time an alarm event is invoked.

Cloud Backup

These topics provide information about storing and retrieving the backups directly to the ecosystem of selected cloud providers, namely Amazon S3 and Softlayer Object Storage.

- Back up to Amazon Simple Storage Service using ON-Bar and the PSM
  You can use **ON-Bar** and the Primary Storage Manager to back up and restore data to or from the Amazon Simple Storage Service (S3). You are responsible for terms and any charges associated with your use of the Amazon Simple Storage Service.
- Back up to Softlayer using ON-Bar and the PSM
  You can use ON-Bar and the Primary Storage Manager to back up and restore data to or from the Softlayer Object Storage. You are responsible for terms and any charges associated with your use of the Softlayer.

Back up to Amazon Simple Storage Service using ON-Bar and the PSM

You can use **ON-Bar** and the Primary Storage Manager to back up and restore data to or from the Amazon Simple Storage Service (S3). You are responsible for terms and any charges associated with your use of the Amazon Simple Storage Service.

Prerequisites:

- You must have an Amazon account to perform cloud storage backups. See the Amazon website for instructions about setting up an account.

The following steps show how to back up data to the Amazon Simple Storage Service (S3) System and restore from it by using **ON-Bar** and the PSM. In this context, cloud storage refers to an online storage service over the Internet. If you choose to back up to cloud storage, you do not need to provide local devices. Instead, you back up the data to a virtual device, most likely located on the Internet.

1. Create a group to access S3
   a. Using a web browser, navigate to the Amazon S3 website and log on.
   b. Select Groups tab and click Create New Group.
   c. Specify the Group name and click Next Step.
   d. In the Attach Policy page, attach a policy to the group. In this case we will select **AmazonS3FullAccess** . As the name implies, this policy will allow any member of this group to do anything in all the containers in S3. Click Next Step.
      Note:
      If you want to use S3 for other purposes or for multiple instances, you can change this by going in to the Policies tab before creating the group and creating a customized policy that suits your needs. Ensure to be careful before granting full access to the Amazon S3, as it will provide the user to access all the S3 data in the account .
   e. In the Review page, review your entries and click Create Group.
   The new group is created.
2. Create a User to Access S3
   a. Select Users tab and click Create New Users.
   b. Specify the User name and select Generate and access key for each user.
   c. Click Create.
      Access Key and the Secret Access Key are generated. These keys are equivalent to username and password that can be used to store and retrieve data from S3 using APIs.

      Note: Only authorized user can access these keys. You need to download these credentials into a text file and store in a safe location. If you lose them, you need to create a new user again.
   d. Click Close.
      User page with newly created user will be displayed.
3. Assign the user to the group.
   a. Click the check box next to the user and select User Actions.
   b. Click Add User to Groups.

4. Create a bucket.
    a. Select Services tab and click Amazon S3.
    b. Click Create Bucket.
    c. Specify a bucket name and select the appropriate region.
5. Configure Primary Storage Manager to use the Bucket and Credentials you just created.
   Create a device in PSM of type CLOUD using S3 as provider.

```
onpsm -D add S3_CLOUD_DEV \
-g DBSPOOL \
-p HIGHEST \
-t CLOUD \
--url https://ifmx-s3-dev.s3.amazonaws.com \
--provider S3 \
--user AKIAIT1111155555X4PA \
--password A2nB21111155555nvTI0X9ZxGzUJNJivoBQY9MrD \
--container ifmx-s3-dev \
--region    us-east-1
--max_part_size 25600
```

   In this command line:
    a. **S3_CLOUD_DEV,** is the arbitrary name given to this device. With FILE type devices, this is actually the full path of the directory that will store the data, but in the case of CLOUD type devices, it is just any name that will help you organize your devices. This name and the pool (DBSPOOL in this case) must be unique.
    b. **'-t CLOUD',** is the device type that enable PSM to store/retrieve the data to/from a CLOUD infrastructure.
    c. **'--provider S3',** is the target cloud provider (Amazon S3) in this case. At present, only S3 and SWIFT (OpenStack SWIFT) are supported.
    d. **'--url** https://ifmx-s3-dev.s3.amazonaws.com is the URL where your backups will go. In the specific case of S3, the bucket name is part of the URL provided.
    e. **--user AKIAIT1111155555X4PA**, for S3 is the Access Key provided to you when your user was created.
    f. **--password A2nB21111155555nvTI0X9ZxGzUJNJivoBQY9MrD**, for S3 is the Secret Key provided to you when the user was created.
    g. **--container ifmx-s3-dev** is the amazon bucket.
    h. **--region us-east-1** is used for V4 authentication.
       Note: If **--region** is not specified, PSM will use V2 authentication.
    i. **—-max_part_size 25600** will fragment your objects in 25MB pieces, in the case of S3, size between 25 and 100 MB is recommended.
6. Check the created device:

```
$ onpsm -D list

Informix Primary Storage Manager Device List

Type    Prio     Block/Size (MB)   Pool Name      Device Name
CLOUD   HIGHEST       --/--        DBSPOOL        S3_CLOUD_DEV

CLOUD   HIGHEST       --/--        LOGPOOL        S3_CLOUD_DEV
```

7. Take a Level Zero Backup:

```
$ onbar -b -L 0 -w

$ echo $?

$ 0
```

8. Check Backup Data in your Bucket using S3 management console.

Back up to Softlayer using ON-Bar and the PSM

You can use ON-Bar and the Primary Storage Manager to back up and restore data to or from the Softlayer Object Storage. You are responsible for terms and any charges associated with your use of the Softlayer.

Prerequisites:

- You must have a Softlayer account to perform cloud storage backups. See the Softlayer website for instructions about setting up an account.

The following steps show how to back up data to the Softlayer System and restore from it by using ON-Bar and the PSM. In this context, cloud storage refers to an online storage service over the Internet. If you choose to back up to cloud storage, you do not need to provide tapes. Instead, you back up the data to a virtual device, most likely located on the Internet.

1. Create an Account Name.
    a. Using a web browser, navigate to the Softlayer website and log on.
    b. Select Storage tab and click Object Storage.
    c. Click Order Object Storage.
    d. Select Storage type and click Continue.
    e. In the Confirm Order page, accept the Master Service Agreement and click Place Order.
   The account is created. Now, you can retrieve the credentials which will include an authentication endpoint (URL) both private and public, a username, and a Key or password.
   Note: If you have a machine with access to the internal private network to Softlayer, you should use the private URL if not, you should use the public one.
2. The PSM command line will request you to provide a container, you can provide any name you want, the container does not need to be created in the SOFTLAYER GUI, it will be created automatically by PSM.
3. Configure Primary Storage Manager.
   Create a device in PSM of type CLOUD using SWIFT as provider.

```
onpsm -D add SOFTLAYER1 \
-g DBSPOOL \
-p HIGHEST \
-t CLOUD \
--url https://dal05.objectstorage.softlayer.net/auth/v1.0 \
--provider SWIFT \
```

```
--user IBM33456-2:arturo_cavero  \
--password bc076837ba9959859c306051c1cda7fb2fe3f012cba15a60c \
--container ifmx_dev \
--max_part_size 25600
```

In this command line:

a. **SOFTLAYER1,** is the arbitrary name given to this device. With FILE type devices, this is actually the full path of the directory that will store the data, but in the case of CLOUD type devices, it is just any name that will help you organize your devices. This name and the pool (DBSPOOL in this case) must be unique.
b. **'-t CLOUD',** is the device type that enable PSM to store/retrieve the data to/from a CLOUD infrastructure.
c. **'--provider SWIFT',** is the target cloud provider (Softlayer) in this case. At present, only SWIFT (OpenStack SWIFT) is supported.
d. **'--url** https://dal05.objectstorage.softlayer.net/auth/v1.0 is the URL where your backups will go.
e. **--user IBM33456-2:arturo_cavero**, for SWIFT is the User Name provided to you when your user was created.
f. **--password bc076837ba9959859c306051c1cda7fb2fe3f012cba15a60c**, for SWIFT is the Secret Key provided to you when the user was created.
g. **--container ifmx_-dev** is the Softlayer container.
h. **—-max_part_size 25600** will fragment your objects in 25MB pieces, in the case of SWIFT, size between 25 and 100 MB is recommended.

4. Check the created device:

```
$ onpsm -D list

Informix Primary Storage Manager Device List

Type    Prio     Block/Size (MB)  Pool Name      Device Name
CLOUD   HIGHEST       --/--        DBSPOOL        SOFTLAYER1

CLOUD   HIGHEST       --/--        LOGPOOL        SOFTLAYER1
```

5. Take a Level Zero Backup:

```
$ onbar -b -L 0 -w

$ echo $?

$ 0
```

6. Check Backup Data in your Bucket using Softlayer management console.

Appendixes

- Troubleshooting some backup and restore errors
  This appendix lists some error and informational messages that you can receive during a backup or restore, describes under what circumstances the errors might occur or the message might appear, and provides possible solutions or workarounds.
- Migrate data, servers, and tools
- GLS support
  This appendix contains information about using Global Language Support (GLS) with ON-Bar.

Troubleshooting some backup and restore errors

This appendix lists some error and informational messages that you can receive during a backup or restore, describes under what circumstances the errors might occur or the message might appear, and provides possible solutions or workarounds.

Find errors by viewing the ON-Bar activity log

The database server does not show errors in standard output (**stdout**) if an error occurs when you use **onbar -b** to back up storage spaces or **onbar -r** to restore storage spaces. Therefore, when you use **onbar -b** or **onbar -r**, you must check information in the ON-Bar activity log (bar_act_log). As ON-Bar backs up and restores data, it writes progress messages, warnings, and error messages to the bar_act.log.

- Corrupt page during an archive
  The message `Archive detects that page is corrupt` indicates that page validation failed. If you receive this message, you can identify the table that has the corrupt page.
- Log backup already running
  When using ON-Bar to create a backup, the informational messages `log backup is already running` in the bar_act.log file and `Process exited with return code 152` in the online.log file might appear under some circumstances.
- No server connection during a restore
  During a whole system restore with ON-Bar, the error `archive api error: no server connection` might appear in the bar_act.log file. ON-Bar then connects to the storage manager successfully, but eventually fails with the error `archive api error: not yet open`. If you receive these message, you can take steps to solve the problem.
- Drop a database before a restore
  If you perform a level-0 archive using ON-Bar and a storage manager, then drop a database, and then perform a restore with the **onbar -r** command, the database remains dropped. The restore salvages the logs and the logs contains the DROP DATABASE statement. When the logs are salvaged, or replayed, the database is dropped. If you receive these message, you can take steps to solve the problem.
- No dbspaces or blobspaces during a backup or restore
  If the emergency boot file, ixbar.*servernum*, does not have the correct entries for objects in the backup, the message `There are no DB/BLOBspaces to backup/restore` appears in bar_act.log file during a restore started with the **onbar -r** or **onbar -r -w** command.
- Changing the system time on the backup system
  In some circumstances when there is a problem with the system time, ON-Bar fails with the message `There are no storage spaces or logical logs to backup or restore`. If this occurs, you can take steps to solve the problem.

Corrupt page during an archive

The message `Archive detects that page is corrupt` indicates that page validation failed. If you receive this message, you can identify the table that has the corrupt page.

During an archive, the database server validates every page before writing it to the archive device. This validation checks that the elements on the page are consistent with the expected values. When a page fails this validation, a message similar to the following is written to the online.log file:

```
16:27:49 Assert Warning: Archive detects that page 1:10164 is corrupt.
16:27:49 Who: Session(5, informix@cronus, 23467, 10a921048)
Thread(40, arcbackup1, 10a8e8ae8, 1)
File: rsarcbu.c Line: 2915
16:27:49 stack trace for pid 23358 written to /tmp/af.41043f4
16:27:49 See Also: /tmp/af.41043f4
16:27:49 Archive detects that page 1:10164 is corrupt.
16:27:50 Archive on rootdbs Completed with 1 corrupted pages detected.
```

The archive stops after detecting 10 corrupt pages. The online.log file displays the full error message, including the page address, for the first 10 errors. Subsequently, only the count of the number of corrupt pages is put in to the online.log.

After you receive this message, identify which table the corrupt page belongs to by examining the output of the **oncheck –pe** command. To determine the extent of the corruption, execute the **oncheck –cID** command for that table.

A corrupt page is saved onto the backup media. During a restore, the corrupt page is returned in its corrupt form. No errors messages are written to the online.log when corrupt pages are restored, only when they are archived.

Log backup already running

When using ON-Bar to create a backup, the informational messages `log backup is already running` in the bar_act.log file and `Process exited with return code 152` in the online.log file might appear under some circumstances.

These messages can appear under the following circumstances:

- When the ALARMPROGRAM configuration parameter is set to log_full.sh.
  Periodically, events cause log_full.sh to trigger the **onbar -b -l** command. If a log fills while the **onbar -b -l** command is running, then ON-Bar backs up that log as well. If the backup has not completed by the time of the next event trigger, it generates a warning in the bar_act.log file. At the time of the next event trigger, the log backup can continue.
- When the **onbar -b -l** command is started automatically.
  A level-0 archive (especially when started with the -w option) first archives the database and then automatically start the **onbar -b -l** command to back up any logical logs that are currently full and not yet backed up. There might not be a log_full.sh message in online.log, because the **onbar -b -l** command is started directly.
- When you mount a new tape after filling a previous tape, a log_full.sh event is scheduled but not triggered.
  As soon as the next log fills and generates an event trigger in the log_full.sh file, all available logs are archived.

  You can force the archive by running **onbar -b -l** or force log_full.sh to be triggered by running **onmode -l**.

No server connection during a restore

During a whole system restore with ON-Bar, the error `archive api error: no server connection` might appear in the bar_act.log file. ON-Bar then connects to the storage manager successfully, but eventually fails with the error `archive api error: not yet open`. If you receive these message, you can take steps to solve the problem.

The bar_act.log file contains information similar to the following messages:

```
2000-03-09 10:51:06 19304  19303 /usr/informix/bin/onbar_d -r -w
2000-03-09 10:51:09 19304  19303 ERROR: Unable to start the physical restore:
Archive API error: no server connection.
2000-03-09 10:51:09 19304  19303 Successfully connected to Storage Manager.
2000-03-09 10:51:36 19304  19303 Process 19304 received signal 3. Process will
exit after cleanup.
2000-03-09 10:59:13 19811  19810 /usr/informix/bin/onbar_d -r -w
2000-03-09 10:59:16 19811  19810 ERROR: Unable to start the physical restore:
Archive API error: no server connection.
2000-03-09 10:59:16 19811  19810 Successfully connected to Storage Manager.
2000-03-09 11:01:12 19811  19810 Begin cold level 0 restore llog1.
2000-03-09 11:01:12 19811  19810 ERROR: Unable to write restore data to the
database server: Archive API error: not yet open.
```

To solve this problem, check if the database server is still running. If it is, shut down the database server and run the command again.

Drop a database before a restore

If you perform a level-0 archive using ON-Bar and a storage manager, then drop a database, and then perform a restore with the **onbar -r** command, the database remains dropped. The restore salvages the logs and the logs contains the DROP DATABASE statement. When the logs are salvaged, or replayed, the database is dropped. If you

receive these message, you can take steps to solve the problem.

To prevent this situation, perform a physical restore using the **onbar -r -p** command, and then a logical restore using the **onbar -r -l** command. This sequence does not salvage the logs and does restore the database.

No dbspaces or blobspaces during a backup or restore

If the emergency boot file, ixbar.*servernum*, does not have the correct entries for objects in the backup, the message `There are no DB/BLOBspaces to backup/restore` appears in bar_act.log file during a restore started with the **onbar -r** or **onbar -r -w** command.

This error can appear under the following circumstances:

- During an external restore, if the emergency boot file was not copied from the source system.
- If the emergency boot file was recreated after the archive backup was made. The previous file is saved in the form: `ixbar.xx.xxxx`.
- An attempt to execute the **onbar -r -w** command with a backup that is not a full system backup.

- Restore blobspace BLOBs
  You can use table-level restore to restore a BLOB that is stored in a table. However, restoring a BLOB that is stored in a blobspace is not supported. If you attempt to restore a blobspace BLOB, the column is set to NULL.

Restore blobspace BLOBs

You can use table-level restore to restore a BLOB that is stored in a table. However, restoring a BLOB that is stored in a blobspace is not supported. If you attempt to restore a blobspace BLOB, the column is set to NULL.

Changing the system time on the backup system

In some circumstances when there is a problem with the system time, ON-Bar fails with the message `There are no storage spaces or logical logs to backup or restore`. If this occurs, you can take steps to solve the problem.

Time lines use the UNIX time as the archive checkpoint time for dbspaces and the closing time for logical logs. If logs are not automatically backed up and the system clock is changed, the time line can get corrupted.

For example, if you have logical logs that were closed before the archive checkpoint time, they have a timestamp that is higher than the archive checkpoint time. The dbspace does not need the logs and ON-Bar will try to restore the backup immediately. if a log cannot be found, ON-Bar fails with the following message: `There are no storage spaces or logical logs to backup or restore.`

To restore the storage space and logical logs:

1. Change the clock back to its original value.
2. Recover the system from backup.
3. Change the clock back to the new time.

Migrate data, servers, and tools

- Backing up before a database server or storage-manager upgrade
  Before you upgrade to a new version of the database server, you must perform a complete backup.
- Upgrading a third-party storage manager
  If you upgrade a third-party storage manager vendors, do not remove the old storage manager until you verify that the new storage manager works for both backup and restore operations.
- Changing storage-manager vendors
  If you change storage manager vendors, do not remove the old storage manager until you have proof that the new storage manager works for both backup and restore operations. You can use the old storage manager as a backup storage manager to use in case the new storage manager does not meet your needs.
- Switching from ontape to ON-Bar
  You cannot back up data with **ontape** and restore it with ON-Bar, or conversely because the data storage formats and backup capabilities are different. However, you can back up data with **ontape**, prepare to use ON-Bar, and then back up with ON-Bar.

Backing up before a database server or storage-manager upgrade

Before you upgrade to a new version of the database server, you must perform a complete backup.

Important: The database server conversion software automatically recreates the **sysutils** database when you upgrade to the latest version of the database server. All backup and restore information from the old database server version is lost. Backups that you make under the older version of the database server are not compatible with the newer version of the database server.
To prepare for an upgrade:

1. Use ON-Bar to perform a level-0 backup of all your data before you upgrade your database server or change storage managers.
2. Save these backups so that you can restore the data in case you need to revert to the old database server version.
3. Before you upgrade, back up the administrative files.

4. After you upgrade the database server, back up all storage spaces and logical logs.
   For complete information about database server migration, see the *IBM® Informix® Migration Guide*.

   If you change storage manager vendors, do not remove the old storage manager until you verify that the new storage manager works for both backup and restore operations.

Related reference:
Upgrading a third-party storage manager
Changing storage-manager vendors

Upgrading a third-party storage manager

If you upgrade a third-party storage manager vendors, do not remove the old storage manager until you verify that the new storage manager works for both backup and restore operations.

Prerequisite: Before upgrading, perform a complete backup of the database server.

To use a new version of a third-party storage manager with the database server:

1. Install the new storage manager before you bring up the database server.
2. Update the sm_versions file with the new storage-manager definition.

Verify that the new storage manager operates correctly before you use it in a production environment:

- Make sure that the storage manager can find the backup objects that ON-Bar requests.
- Make sure that the new storage-manager version is able to read media written with your old version.

If you have continuous logical-log backup set up on the database server, ON-Bar can start backing up the logical logs soon after the database server comes online.

Use the **onsmsync** utility to expire old backup history in the **sysutils** database and emergency boot files.

Related tasks:
Backing up before a database server or storage-manager upgrade

Changing storage-manager vendors

If you change storage manager vendors, do not remove the old storage manager until you have proof that the new storage manager works for both backup and restore operations. You can use the old storage manager as a backup storage manager to use in case the new storage manager does not meet your needs.

ON-Bar supports working with multiple storage managers at the same time. To set up to test one storage manager and keep the other as a backup storage manager, specify information for both of the storage managers in the BAR_BSALIB_PATH configuration parameter and in the $INFORMIXDIR/etc/sm_versions file.

If you cannot use the old and new storage managers at the same time, use ON-Bar and the IBM® Informix® Primary Storage Manager or **ontape** as an alternative for backups while you check that backup and restore operations work correctly with the new storage manager. Until you confirm that the new storage manager works correctly, perform all your backups as whole system level-0 backups (**onbar -b -L 0 -w**)

If you change physical connectivity, such as moving a storage device from a local connection to a network server, make sure the new storage manager can move the data across the network. Also ensure that the new storage manager can send multiple data streams to storage devices. It also might use a different version of XBSA.

Related tasks:
Backing up before a database server or storage-manager upgrade

Switching from ontape to ON-Bar

You cannot back up data with **ontape** and restore it with ON-Bar, or conversely because the data storage formats and backup capabilities are different. However, you can back up data with **ontape**, prepare to use ON-Bar, and then back up with ON-Bar.

To switch from **ontape** to ON-Bar:

1. Use **ontape** to perform a full backup.
2. Take the backup media offline to prevent possible reuse or erasure.
3. Configure the storage manager to be used with ON-Bar.
4. Configure your environment:
   a. Set the configuration parameters that you use with ON-Bar and the storage manager.
   b. If you use a storage manager other than the IBM® Informix® Primary Storage Manager, create the sm_versions file with the storage-manager definition. The Informix Primary Storage Manager does not use the **sm_versions.std** file.
5. Use ON-Bar (**onbar -b** or **onbar -b -w**) to perform a full backup.
6. Verify the backup with the **onbar -v** command.

GLS support

This appendix contains information about using Global Language Support (GLS) with ON-Bar.

- Use GLS with the ON-Bar utility
  The ON-Bar utility supports Global Language Support (GLS), which allows users to work in their native language. The language that the client application uses is called the *client locale*. The language that the database uses for its server-specific files is called the *server locale*.
- Use the GL_DATETIME environment variable with ON-Bar
  The database server must know how to interpret and convert the end-user formats when they appear in date or time data that the client application sends. You can use the GL_DATE and GL_DATETIME environment variables to specify alternative date and time formats.
- Use GLS with the ontape utility
  The **ontape** utility supports GLS in the same way as ON-Bar does. You can specify the database name in the national locale.

Use GLS with the ON-Bar utility

The ON-Bar utility supports Global Language Support (GLS), which allows users to work in their native language. The language that the client application uses is called the *client locale*. The language that the database uses for its server-specific files is called the *server locale*.

ON-Bar must run on the same computer as the database server. However, you can run ON-Bar in any locale for which you have the supporting message and globalization files. For example, if the server locale is English and the client locale is French, you can issue ON-Bar commands in French.

The following command performs a level-0 backup of the dbspaces specified in the file, tomb: **onbar -b -L 0 -f tomb**

On Windows, you cannot use multibyte file names in backup or restore commands because they are not supported.

The **sysutils** database, the emergency boot files, and the storage-manager boot file are created with the en_us.8859-1 (default English) locale. The ON-Bar catalog tables in the **sysutils** database are in English. Change the client and database locales to en_us.8859-1 before you attempt to connect to the **sysutils** database with DB-Access or third-party utilities.

- Identifiers that support non-ASCII characters
  You can use non-ASCII characters in the database names and filenames with the ON-Bar and **ondblog** commands, and for file names in the onconfig file.
- Identifiers that require 7-bit ASCII characters
  You must use 7-bit ASCII characters for storage space names and database server names.
- Locale of ON-Bar messages
  All ON-Bar messages appear in the activity log in the client locale except the messages that the database server issues.

Identifiers that support non-ASCII characters

You can use non-ASCII characters in the database names and filenames with the ON-Bar and **ondblog** commands, and for file names in the onconfig file.

The *IBM® Informix® GLS User's Guide* describes the SQL identifiers that support non-ASCII characters. Non-ASCII characters include both 8-bit and multibyte characters.

For example, you can specify a non-ASCII file name for the ON-Bar activity login BAR_ACT_LOG and a non-ASCII path name for the storage-manager library in BAR_BSALIB_PATH.

Identifiers that require 7-bit ASCII characters

You must use 7-bit ASCII characters for storage space names and database server names.

Locale of ON-Bar messages

All ON-Bar messages appear in the activity log in the client locale except the messages that the database server issues.

For example, the part of the message that tells you that a database server error occurred appears in the client locale, and the server-generated part appears in the server locale.

Use the GL_DATETIME environment variable with ON-Bar

The database server must know how to interpret and convert the end-user formats when they appear in date or time data that the client application sends. You can use the GL_DATE and GL_DATETIME environment variables to specify alternative date and time formats.

If you do not set these environment variables, ON-Bar uses the date and time format of the client locale.

If you perform a point-in-time restore, enter the date and time in the format specified in the GL_DATETIME environment variable if it is set.

Use GLS with the ontape utility

The **ontape** utility supports GLS in the same way as ON-Bar does. You can specify the database name in the national locale.

Replication

The topics in this group contain information about replicating data in IBM® Informix® databases by using Enterprise Replication.

Enterprise Replication is a good solution if you have geographically dispersed servers and you do not want to replicate all data. You specify what data is replicated between servers, how data conflicts are handled, and how servers are connected. Enterprise Replication servers can be on heterogeneous hardware and database server versions. If a server fails, Enterprise Replication stores the data from other servers that is not yet replicated until the network or system is operational. You can also configure a grid to simplify administration by propagating SQL statements and files from one server to all other servers in the grid.

Tip: If you want to replicate data only for high-availability, you do not need to use Enterprise Replication. You can use high-availability clusters to provide one or more copies of the primary database server. See High availability and scalability.

- Enterprise Replication
  These topics describe the concepts of data replication using IBM Informix Enterprise Replication, including how to design your replication system, as well as administer and manage data replication throughout your enterprise.

Enterprise Replication

These topics describe the concepts of data replication using IBM® Informix® Enterprise Replication, including how to design your replication system, as well as administer and manage data replication throughout your enterprise.

These topics are for database server administrators with the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with database server administration, operating- system administration, and network administration

These topics are taken from IBM Informix Enterprise Replication *Guide*.

- About Enterprise Replication
  IBM Informix Enterprise Replication generates and manages multiple copies of data at one or more sites, which allows an enterprise to share corporate data throughout its organization.
- Planning and designing for Enterprise Replication
  Before you set up your replication system, plan how to include Enterprise Replication into your database server environment, design your database schema by following Enterprise Replication requirements, and then design your replication system between database servers.
- Setting up and managing Enterprise Replication
  After you design your replication system, you define it and start replication.
- Push data feature
- Loopback Replication
- Appendixes

About Enterprise Replication

IBM® Informix® Enterprise Replication generates and manages multiple copies of data at one or more sites, which allows an enterprise to share corporate data throughout its organization.

These topics provide an overview of IBM Informix Enterprise Replication and how to administer it.

- IBM Informix Enterprise Replication technical overview
  IBM Informix Enterprise Replication is an asynchronous, log-based tool for replicating data between IBM Informix database servers. Enterprise Replication on the source server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.
- How Enterprise Replication Replicates Data

IBM Informix Enterprise Replication technical overview

IBM® Informix® Enterprise Replication is an asynchronous, log-based tool for replicating data between IBM Informix database servers. Enterprise Replication on the source server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.

At each target server, Enterprise Replication receives and applies each transaction contained in the replication data to the appropriate databases and tables as a normal, logged transaction.

- Enterprise Replication Terminology
  You must understand Enterprise Replication terminology.
- Asynchronous Data Replication
- Log-Based Data Capture
- High Performance

- High Availability
- Consistent Information Delivery
- Repair and Initial Data Synchronization
  Enterprise Replication provides initial data synchronization and multiple methods to repair replicated data.
- Flexible Architecture
  Enterprise Replication allows replications based on specific business and application requirements and does not impose model or methodology restrictions on the enterprise.
- Centralized Administration
- Ease of Implementation
- Network Encryption
  Enterprise Replication supports the same network encryption options that you can use with communications between server and clients to provide complete data encryption.

Related concepts:
How Enterprise Replication Replicates Data

Enterprise Replication Terminology

You must understand Enterprise Replication terminology.

The following terms define the data in an Enterprise Replication system and how it is treated:

- Enterprise Replication server
- Shard server
- Replication key
- Replicate
- Master Replicate
- Shadow Replicate
- Participant
- Replicate Set
- Template
- Global Catalog
- Grid

Enterprise Replication server

An Enterprise Replication server, or *replication server*, is the IBM® Informix® database server that participates in data replication.

The replication server maintains information about the replication environment, which columns are replicated, and the conditions under which the data is replicated. This information is stored in a database, **syscdr**, that the database server creates when it is initialized. Multiple database servers can be on the same physical computer, and each database server can participate in Enterprise Replication.

Shard server

A *shard server* is a database server that participates in data replication and receives horizontally partitioned (sharded) data. A *shard cluster* is the group of database servers over which a table or collection is partitioned.

Replication key

A *replication key* consists of one or more columns that uniquely identifies each replicated row. The replication key must be the same on all servers that participate in the replicate. Typically, the replication key is a primary key constraint. Otherwise, you can specify ERKEY shadow columns or another unique index as the replication key.

The replication key for a shard cluster consists of a single column, and is called a *shard key*.

Replicate

A *replicate* defines the replication *participants* and various attributes of how to replicate the data, such as frequency and how to handle any conflicts during replication.

For more information, see Define a replicate and cdr define replicate.

Master replicate

A *master replicate* is a replicate that guarantees data integrity by verifying that replicated tables on different servers have consistent column attributes. Master replicates also support alter operations on replicated tables.

Shadow replicate

A *shadow replicate* is a copy of an existing (primary) replicate. Shadow replicates allow Enterprise Replication to manage alter and repair operations on replicated tables.

Participant

A *participant* specifies the data (database, table, and columns) to replicate and the database servers to which the data replicates.

Replicate set

A *replicate set* combines several replicates to form a set that can be administered together as a unit.

Template

A *template* provides a mechanism to set up and deploy replication for a group of tables on one or more servers. A template is especially useful if you have many tables to replicate between many servers. A template defines a group of master replicates and a replicate set for a specified group of tables that are based on attributes such as database, tables, columns, and primary keys from the master node.

You create a template by running the **cdr define template** command and then instantiate, or realize, it on servers with the **cdr realize template** command.

Global catalog

Each database server that participates in Enterprise Replication maintains tables in the **syscdr** database to track Enterprise Replication configuration information and state. For all root and nonroot replication servers, this catalog is a *global catalog* that maintains a global inventory of Enterprise Replication configuration information. The global catalog is created when you define the server for replication.

The global catalog includes the following information:

- Enterprise Replication server definitions and state
- Routing and connectivity information
- Replicate definitions and state
- Participant definitions and state
- Replicate set definitions and state
- Conflict detection and resolution rules and any associated SPL routines

Grid

A grid is a set of replication servers that you can administer as a unit. When you run SQL data definition statements from within a grid context on one server in the grid, they are propagated to all other servers in the grid. You can run SQL data manipulation statements and routines through grid routines. You can propagate external files to other servers in the grid. You can run grid queries to consolidate data from multiple grid servers.

Related concepts:
Connect Option
Related tasks:
Customizing the Replication Server Definition
Connect to another replication server

Asynchronous Data Replication

Enterprise Replication uses *asynchronous* data replication to update the databases that reside at a replicated site after the primary database has committed a change.

With asynchronous replication, the delay to update the replicated-site databases can vary depending on the business application and user requirements. However, the data eventually synchronizes to the same value at all sites. The major benefit of this type of data replication is that if a particular database server fails, the replication process can continue and all transactions in the replication system will be committed.

In contrast to this, *synchronous* data replication replicates data immediately when the source data is updated. Synchronous data replication uses the *two-phase commit* technology to protect data integrity. In a two-phase commit, a transaction is applied only if *all* interconnected distributed sites agree to accept the transaction. Synchronous data replication is appropriate for applications that require immediate data synchronization. However, synchronous data replication requires that all hardware components and networks in the replication system be available at all times. For more information about synchronous replication, refer to the discussion of two-phase commit in your *IBM Informix Administrator's Guide*.

Asynchronous replication is often preferred because it allows for system and network failures.

Asynchronous replication allows the following replication models:

- Primary-target (Primary-Target Replication System)
  All database changes originate at the primary database and are replicated to the target databases. Changes at the target databases are not replicated to the primary.

- Update-anywhere (Update-Anywhere Replication System)
  All databases have read and write capabilities. Updates are applied at all databases.

The update-anywhere model provides the greater challenge in asynchronous replication. For example, if a replication system contains three replication sites that all have read and write capabilities, conflicts occur when the sites try to update the same data at the same time. Conflicts must be detected and resolved so that the data elements eventually have the same value at every site. For more information, see Conflict Resolution.

Log-Based Data Capture

Enterprise Replication uses *log-based data capture* to gather data for replication. Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication and then evaluates the row images.

Log-based data capture takes changes from the logical log and does not compete with transactions for access to production tables. Log-based data-capture systems operate as part of the normal database-logging process and thus add minimal overhead to the system.

Two other methods of data capture, which Enterprise Replication does not support, include:

- Trigger-based data capture
  A trigger is code in the database that is associated with a piece of data. When the data changes, the trigger activates the replication process.
- Trigger-based transaction capture
  A trigger is associated with a table. Data changes are grouped into transactions and a single transaction might trigger several replications if it modifies several tables. The trigger receives the whole transaction, but the procedure that captures the data runs as a part of the original transaction, thus slowing down the original transaction.

High Performance

Enterprise Replication provides high performance by not overly burdening the data source and by using networks and all other resources efficiently.

Because Enterprise Replication captures changes from the logical log instead of competing with transactions that access production tables, Enterprise Replication minimizes the effect on transaction performance. Because the capture mechanism is internal to the database, the database server implements this capture mechanism efficiently. For more information, see Log-Based Data Capture.

All Enterprise Replication operations are performed in parallel, which further extends the performance of Enterprise Replication.

High Availability

Because Enterprise Replication implements asynchronous data replication, network and target database server outages are tolerated. In the event of a database server or network failure, the local database server continues to service local users. The local database server stores replicated transactions in persistent storage until the remote server becomes available.

If high availability is critical, you can use high-availability clusters in conjunction with Enterprise Replication. High-availability clusters support synchronous data replication between database servers: a primary server, which can participate in Enterprise Replication, and one or more secondary servers, which do not participate in Enterprise Replication. If a primary server in a high-availability cluster fails, a secondary server can take over the role of the primary server, allowing it to participate in Enterprise Replication. Client connections to the original primary server can be automatically switched to the new standard server.

For more information on using high-availability clusters with Enterprise Replication, see Using High-Availability Clusters with Enterprise Replication.

Consistent Information Delivery

IBM® Informix® Enterprise Replication protects data integrity. All IBM Informix Enterprise Replication transactions are stored in a reliable queue to maintain the consistency of transactions.

IBM Informix Enterprise Replication uses a data-synchronization process to ensure that transactions are applied at the target database servers in any order equivalent to the order that they were committed on the source database server. If Enterprise Replication can preserve the consistency of the database, Enterprise Replication might commit transactions in a slightly different order on the target database.

If update conflicts occur, IBM Informix Enterprise Replication provides built-in automatic conflict detection and resolution. You can configure the way conflict resolution behaves to meet the needs of your enterprise. For more information, see Conflict Resolution.

Repair and Initial Data Synchronization

Enterprise Replication provides initial data synchronization and multiple methods to repair replicated data.

You can easily bring a new table up-to-date with replication when you start a new replicate, or when you add a new participant to an existing replicate, by specifying an initial synchronization. Initial synchronization can be run online while replication is active.

If replication has failed for some reason, you can repair replicated data by running the **cdr sync replicate** or **cdr sync replicateset** command to resynchronize data and correct data mismatches between replicated tables. You can repair data while replication is active.

You can also repair data after replication has failed by using ATS and RIS files.Enterprise Replication examines the specified ATS or RIS file and attempts to reconcile the rows that failed to be applied.

Related concepts:
Resynchronizing Data among Replication Servers
Related tasks:
Initially Synchronizing Data Among Database Servers
Repairing Failed Transactions with ATS and RIS Files

Flexible Architecture

Enterprise Replication allows replications based on specific business and application requirements and does not impose model or methodology restrictions on the enterprise.

Enterprise Replication supports both primary-target and update-anywhere replication models.

Enterprise Replication supports the following network topologies:

- Fully connected
  Continuous connectivity between all participating database servers.
- Hierarchical tree
  A parent-child configuration that supports continuous and intermittent connectivity.
- Forest of trees
  Multiple hierarchical trees that connect at the root database servers.

You can add High-Availability Data Replication to any of these topologies.

Enterprise Replication supports all built-in IBM® Informix® data types, as well as extended and user-defined data types.

Enterprise Replication operates in LAN, WAN, and combined LAN/WAN configurations across a range of network transport protocols.

Enterprise Replication supports the Global Language Support (GLS) feature, which allows IBM Informix products to handle different languages, regional conventions, and code sets.

Related concepts:
Primary-Target Replication System
Update-Anywhere Replication System
Choosing a Replication Network Topology
Replication and data types
Global language support for replication

Centralized Administration

Enterprise Replication allows administrators to easily manage all the distributed components of the replication system from a single point of control.

You can use the command-line utility (CLU) to administer the replication system from your system command prompt and connect to other servers involved in replication, as necessary. For information, see The cdr utility.

Ease of Implementation

Enterprise Replication provides templates to allow easy set up and deployment of replication for clients with large numbers of tables to replicate. Administrators of Enterprise Replication can use templates to develop scripts and with only a few commands can set up replication over a large number of server nodes. Without using templates, many individual commands must be run. Using templates, you can also easily add a new server into your replication environment and optionally create and populate new database tables.

First, you create a template using the **cdr define template** command. This defines the database, tables, and columns and the characteristics of the replicates that will be created. You can view information about a template by using the **cdr list template** command from a non-leaf node.

Second, you instantiate the template on the servers where you want to replicate this data by running the **cdr realize template** command. If the table already exists on a node, Enterprise Replication verifies it matches the template definition. If the table does not exist on a node, Enterprise Replication can optionally create the table. Enterprise Replication can also optionally perform an initial data synchronization on all servers where you realize the template.

You can delete templates that you no longer need using the **cdr delete template** command.

See Set up replication through templates for more information. All replication commands mentioned in this section are described in detail in The cdr utility.

Network Encryption

Enterprise Replication supports the same network encryption options that you can use with communications between server and clients to provide complete data encryption.

You can use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures privacy and integrity of data transmitted over the network, for connections between Enterprise Replication servers. For information on using the SSL protocol, see Secure sockets layer protocol.

You can use encryption configuration parameters to provide data encryption with a standard cryptography library. A message authentication code (MAC) is transmitted as part of the encrypted data transmission to ensure data integrity. This is the same type of encryption provided by the ENCCSM communications support module for non-replication communication. Enterprise Replication shares the same ENCRYPT_CIPHERS, ENCRYPT_MAC, ENCRYPT_MACFILE, and ENCRYPT_SWITCH configuration parameters with high availability clusters. Enterprise Replication encryption configuration parameters are documented in Enterprise Replication configuration parameter and environment variable reference.

Enterprise Replication cannot accept a connection that is configured with a communications support module. To combine client/server network encryption with Enterprise Replication encryption, configure two network connections for each database server, one with CSM and one without. For more information, see Configuring network encryption for replication servers.

How Enterprise Replication Replicates Data

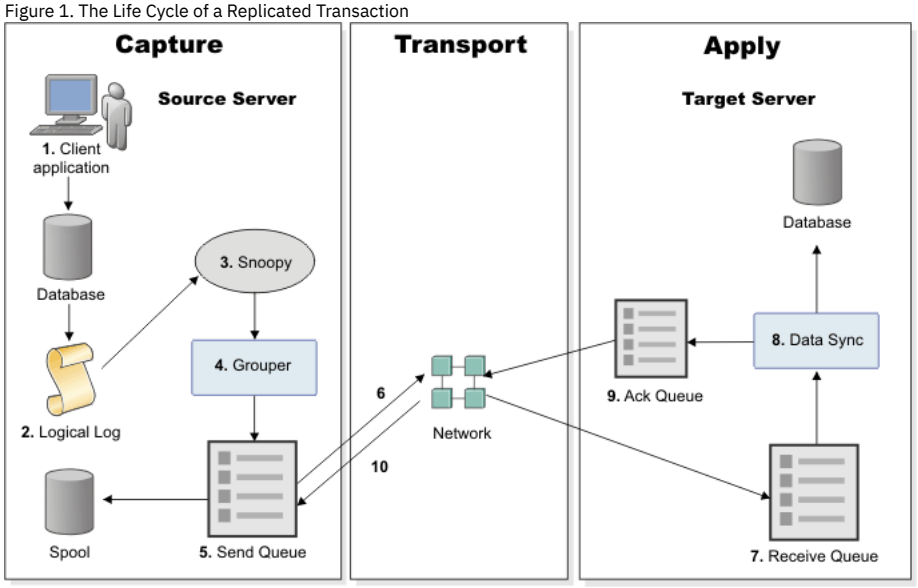
Before you can replicate data, you must declare a database server for replication and define the *replicates* (the data to replicate and the database servers that participate in replication). To declare a database server for replication, see Defining Replication Servers. To define replicates, see Define a replicate. Replication Examples, has simple examples of declaring replication servers and defining replicates.

After you define the servers and replicates, Enterprise Replication replicates data in three phases:

1. Data Capture
2. Data Transport
3. Applying replicated data

The following diagram shows these three phases of replication and the Enterprise Replication components that perform each task.

Figure 1. The Life Cycle of a Replicated Transaction



As shown in the diagram, the following process describes how Enterprise Replication replicates a transaction:

1. A client application performs a transaction in a database that is defined as a replicate.
2. The transaction is put into the logical log.
3. The log capture component, also known as the snoopy component, reads the logical log and passes the log records onto the grouper component.
4. The grouper component evaluates the log records for replication and groups them into a message that describe the operations that were in the original transaction.
5. The grouper component places the message in the send queue. Under certain situations, the send queue spools messages to disk for temporary storage.
6. The send queue transports the replication message across the Enterprise Replication network to the target server.
7. The replication message is placed in the receive queue at the target server.
8. The data sync component applies the transaction in the target database. If necessary, the data sync component performs conflict resolution.
9. An acknowledgment that the message was successfully applied is placed in the acknowledgment queue.
10. The acknowledgment message is sent back to the source server.

- Data Capture
  As the database server writes rows to the logical log, it marks rows that should be replicated. Later, Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication.
- Data Transport

- Applying replicated data
  IBM Informix Enterprise Replication uses a data-synchronization process to apply the replicated data to target database servers.

Related concepts:
IBM Informix Enterprise Replication technical overview

Data Capture

As the database server writes rows to the logical log, it marks rows that should be replicated. Later, Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication.

IBM® Informix® database servers manage the logical log in a circular fashion; the most recent logical-log entries write over the oldest entries. Enterprise Replication must read the logical log quickly enough to prevent new logical-log entries from overwriting the logs Enterprise Replication has not yet processed.

If the database server comes close to overwriting a logical log that Enterprise Replication has not yet processed, by default, user transactions are blocked until Enterprise Replication advances. You can specify other responses to the potential for overwriting the Enterprise Replication replay position.

The row images that participate in replication are passed to Enterprise Replication for further evaluation.

- Row Images
  Enterprise Replication evaluates the initial and final images of a row and any changes that occur between the two row images to determine which rows to replicate. Each row image contains the data in the row and the action that is performed on that row.
- Evaluate rows for updates
  Enterprise Replication evaluates rows for replication-key updates, for WHERE-clause column updates, and for multiple updates to the same row.
- Send queues and receive queues
  Enterprise Replication uses send and receive queues to receive and deliver replication data to and from database servers that participate in a replicate.
- Data Evaluation Examples

Row Images

Enterprise Replication evaluates the initial and final images of a row and any changes that occur between the two row images to determine which rows to replicate. Each row image contains the data in the row and the action that is performed on that row.

A row might change more than once in a particular transaction. For example, a transaction might insert and then update a row before committing. Enterprise Replication evaluates the net effect (final state) of a transaction based on the row buffers in the log. Enterprise Replication then determines what must be replicated, based on the net effect, the initial state of the row, and whether the replicate definition (in particular, the WHERE clause) applies to the initial and final state. Enterprise Replication evaluates the row-image type (INSERT, UPDATE, DELETE), the results of evaluating the replicate WHERE clause for both the initial and final image, and whether the replication key changes as a result of the transaction.

The following table shows the logic that determines which rows are candidates for replication. The source and destination tables are assumed to be initially synchronized (identical before replication begins). If the tables were not synchronized, anomalous behavior might result.

Table 1. Enterprise Replication Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Replication-Key Changed?	Send to Destination Database Server	Comments
INSERT	T *or* F	DELETE	T *or* F	Yes *or* no	Nothing	Net change of transaction results in no replication
INSERT	T *or* F	UPDATE	T	Yes *or* no	INSERT with final row image	Inserts final data of transaction
INSERT	T *or* F	UPDATE	F	Yes *or* no	Nothing	Final evaluation determines no replication
UPDATE	T	DELETE	T *or* F	Yes *or* no	DELETE with initial row image	Net result of transaction is delete
UPDATE	F	DELETE	T *or* F	Yes *or* no	Nothing	Net change of transaction results in no replication
UPDATE	T	UPDATE	T	Yes	DELETE with initial row image and INSERT with final row image	Ensures old replication key does not replicate
UPDATE	T	UPDATE	T	No	UPDATE with final row image	Simple update
UPDATE	T	UPDATE	F	Yes *or* no	DELETE with initial row image	Row no longer matches replicate definition
UPDATE	F	UPDATE	T	Yes *or* no	INSERT with final row image	Row now matches replicate definition
UPDATE	F	UPDATE	F	Yes *or* no	Nothing	No match exists, and therefore, no replication

The following rules apply to the information in the table:

- The initial image is the before image of the transaction in the logical log.
- The replicate evaluates to T (true) or F (false).
- The final image is the image of the transaction that is replicated.

After Enterprise Replication identifies transactions that qualify for replication, Enterprise Replication transfers the transaction data to a queue.

Related concepts:
Evaluate rows for updates
Send queues and receive queues
Related reference:
Data Evaluation Examples

Evaluate rows for updates

Enterprise Replication evaluates rows for replication-key updates, for WHERE-clause column updates, and for multiple updates to the same row.

The following list describes an occurrence in a transaction and the Enterprise Replication action:

- Replication-key updates
  Enterprise Replication translates an update of the replication key into a delete of the original rows and an insert of the row images with the new replication key. If triggers are enabled on the target system, insert triggers are run.
- WHERE-clause column updates
  If a replicate includes a WHERE clause in its data selection, the WHERE clause imposes selection criteria for rows in the replicated table.
  - If an update changes a row so that it no longer passes the selection criteria on the source, it is deleted from the target table. Enterprise Replication translates the update into a delete and sends it to the target.
  - If an update changes a row so that it passes the selection criteria on the source, it is inserted into the target table. Enterprise Replication translates the update into an insert and sends it to the target.
- Multiple-row images in a transaction
  Enterprise Replication compresses multiple-row images and only sends the net change to the target. Because of this, triggers might not execute on the target database. For more information, see Triggers.

Enterprise Replication supports the replication of BYTE and TEXT data types (simple large objects) and BLOB and CLOB data types (smart large objects), and opaque user-defined data types, as well as all built-in IBM® Informix® data types. However, Enterprise Replication implements the replication of these types of data somewhat differently from the replication of other data types. For more information, see Replication of large objects, and Replication of opaque user-defined data types.

Related concepts:
Send queues and receive queues
Related reference:
Row Images
Data Evaluation Examples

Send queues and receive queues

Enterprise Replication uses send and receive queues to receive and deliver replication data to and from database servers that participate in a replicate.

Send queue
: Enterprise Replication stores replication data in memory to be delivered to target database servers that participate in a replicate. If the send queue fills, Enterprise Replication spools the send-queue transaction records to a dbspace and the send-queue row data to an sbspace.

Receive queue
: Enterprise Replication stores replication data in memory at the target database server until the target database server acknowledges receipt of the data. If the receive queue fills as a result of a large transaction, Enterprise Replication spools the receive queue transaction header and replicate records to a dbspace and the receive queue row data to an sbspace.

The data contains the filtered log records for a single transaction. Enterprise Replication stores the replication data in a stable (recoverable) send queue on the source database server. Target sites acknowledge receipt of data when it is applied to or rejected from the target database.

If a target database server is unreachable, the replication data remains in a stable queue at the source database server. Temporary failures are common, and no immediate action is taken by the source database server; it continues to queue transactions. When the target database server becomes available again, queued transactions are transmitted and applied.

If the target database server is unavailable for an extended period, the send queue on the source database server might use excessive resources. In this situation, you might not want to save all transactions for the target database server. To prevent unlimited transaction accumulation, you can remove the unavailable target database server from the replicate. Before the database server that is removed rejoins any replicate, however, you must synchronize (bring tables to consistency) with the other database servers.

Related concepts:
Evaluate rows for updates
Transaction processing impact
Setting Up Send and Receive Queue Spool Areas
Applying replicated data

Related tasks:
Preventing Memory Queues from Overflowing
Related reference:
Row Images
Data Evaluation Examples

Data Evaluation Examples

Figure 1, Figure 2, and Figure 3 show three examples of how Enterprise Replication uses logic to evaluate transactions for potential replication.

Figure 1. Insert Followed by a Delete

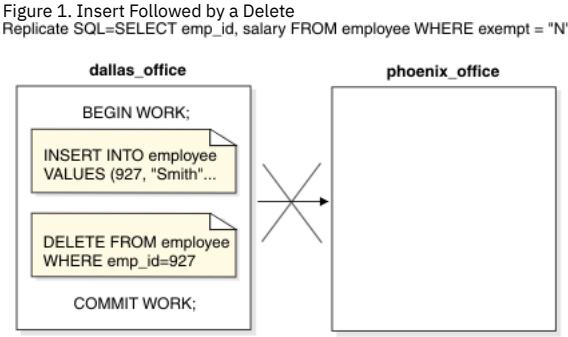


Figure 1 shows a transaction that takes place at the Dallas office. Enterprise Replication uses the logic in Table 1 to evaluate whether any information is sent to the destination database server at the Phoenix office.

Table 1. Insert Followed by a Delete Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
INSERT	T *or* F	DELETE	T *or* F	Yes *or* no	Nothing

Enterprise Replication determines that the insert followed by a delete results in no replication operation; therefore, no replication data is sent.

In Figure 2, Enterprise Replication uses the logic in Table 2 to evaluate whether any information is sent to the destination database server.

Figure 2. Insert Followed by an Update

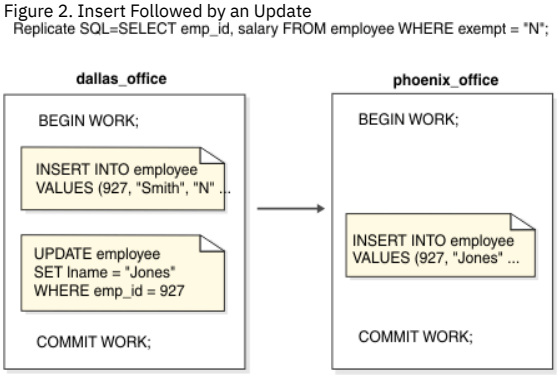


Table 2. Insert Followed by An Update Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
INSERT	T *or* F	UPDATE	T	Yes or no	INSERT with final row image

The replicate WHERE clause imposes the restriction that only rows are replicated where the exempt column contains a value of "N." Enterprise Replication evaluates the transaction (an insert followed by an update) and converts it to an insert to propagate the updated (final) image.

In Figure 3, Enterprise Replication uses the logic in Table 3 to evaluate whether any information is sent to the destination database server.

Figure 3. Update; Not Selected to Selected

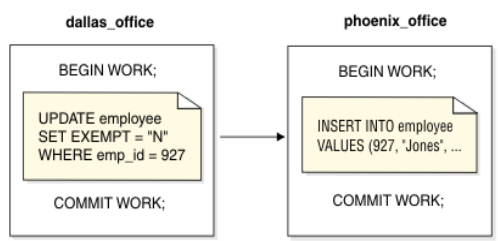


Table 3. Update; Not Selected to Selected Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
UPDATE	F	UPDATE	T	Yes *or* no	INSERT with final row image

The example shows a replicate WHERE clause column update. A row that does not meet the WHERE clause selection criteria is updated to meet the criteria. Enterprise Replication replicates the updated row image and converts the update to an insert.

Related concepts:
Evaluate rows for updates
Send queues and receive queues
Related reference:
Row Images

Data Transport

Enterprise Replication ensures that all data reaches the appropriate server, regardless of a network or system failure. In the event of a failure, Enterprise Replication stores data until the network or system is operational. Enterprise Replication replicates data efficiently with a minimum of data copying and sending.

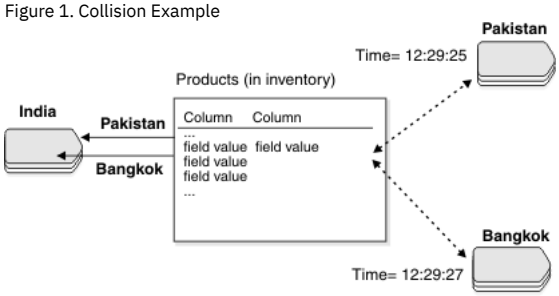
Applying replicated data

IBM® Informix® Enterprise Replication uses a data-synchronization process to apply the replicated data to target database servers.

The target database servers acknowledge receipt of data when the data is applied to the target database. Data modifications that results from synchronization, including modifications that result from trigger invocation, are not replicated. The data-synchronization process ensures that transactions are applied at the target database servers in an order equivalent to the order that they were committed on the source database server. If consistency can be preserved, Enterprise Replication might commit transactions out of order on the target database.

When Enterprise Replication applies replication data, it checks to make sure that no *collisions* exist. A collision occurs when two database servers update the same data simultaneously. Enterprise Replication reviews the data one row at a time to detect a collision.

If Enterprise Replication finds a collision, it must resolve the conflict before applying the replication data to the target database server.
Figure 1. Collision Example



The previous illustration shows a situation that yields a conflict. Pakistan updates the row two seconds before Bangkok updates the same row. The Bangkok update arrives at the India site first, and the Pakistan update follows. The Pakistan time is earlier than the Bangkok time. Because both updates involve the same data and a time discrepancy exists, Enterprise Replication detects a collision.

For more information, see Conflict Resolution.

Enterprise Replication scans to see if the same replication key exists in the target table or in the associated *delete table*, or if a replication order error is detected. A delete table stores the row images of deleted rows. A replication order error is the result of replication data that arrives from different database servers with one of the following illogical results:

- A replicated DELETE that finds no row to DELETE on the target
- An UPDATE that finds no row to UPDATE on the target
- An INSERT that finds a row that exists on the target

Related concepts:
Send queues and receive queues

Planning and designing for Enterprise Replication

Before you set up your replication system, plan how to include Enterprise Replication into your database server environment, design your database schema by following Enterprise Replication requirements, and then design your replication system between database servers.

- Plan for Enterprise Replication
  Before you design a replication system, you must understand how Enterprise Replication interacts with the database server and the other requirements of Enterprise Replication. Many aspects of the Informix database server can affect how you deploy Enterprise Replication.
- Schema design for Enterprise Replication
  When you design the database and tables for replication, you must follow the requirements and restrictions for Enterprise Replication.

- Replication system design
  When you design a replication system, you make three main decisions: how the information flows between servers, how to resolve conflicts between replicated data, and the topology of the network of servers.

Plan for Enterprise Replication

Before you design a replication system, you must understand how Enterprise Replication interacts with the database server and the other requirements of Enterprise Replication. Many aspects of the Informix® database server can affect how you deploy Enterprise Replication.

- Enterprise Replication Server administrator
  You need special privileges to run most Enterprise Replication commands.
- Asynchronous propagation conflicts
  Enterprise Replication asynchronously propagates many control operations through the Enterprise Replication network. Avoid operations that might conflict during propagation.
- Back up and restore of replication servers
  You can back up and restore database servers that participate in Enterprise Replication.
- Compression of replicated data
  You can compress and uncompress data in replicated tables to reduce the amount of needed disk space.
- Transaction processing impact
  Many variables affect the impact that replicating data has on your transaction processing.
- SQL statements and replication
  You can run most SQL statements while replication is active. For some statements, however, you must set alter mode or stop replication.
- Global language support for replication
  You can replicate data in non-default locales.
- Replication between multiple server versions
  You can set up Enterprise Replication across servers of different version levels.

Enterprise Replication Server administrator

You need special privileges to run most Enterprise Replication commands.

To configure and manage Enterprise Replication, you must have one of the following roles or privileges:

- Be the owner of a non-root server
- Have the Database Server Administrator (DBSA) privilege
- Be user **informix** (UNIX) or a be a member of the **Informix-Admin** group (Windows)

All servers in the replication domain must have the same owner.

Related concepts:
Interpret the cdr utility syntax
Related tasks:
Defining Replication Servers
Related information:
grant admin argument: Grant privileges to run SQL administration API commands

Asynchronous propagation conflicts

Enterprise Replication asynchronously propagates many control operations through the Enterprise Replication network. Avoid operations that might conflict during propagation.

When you perform administrative functions using Enterprise Replication, the status that returns from those operations indicates the success or failure of the operation at the database server to which you are directly connected. The operation might still be propagating through the other Enterprise Replication database servers in the network at that time. It might take a significant amount of time before the operation is propagated to database servers that are not connected to the Enterprise Replication network at all times.

Due to this asynchronous propagation, avoid performing control operations in quick succession that might directly conflict with one another without verifying that the first operation was successfully propagated through the entire enterprise network. Specifically, avoid deleting Enterprise Replication objects such as replicates, replicate sets, and Enterprise Replication servers, and immediately recreating those objects with the same name. Doing so can cause failures in the Enterprise Replication system at the time of the operation or later. These failures might manifest themselves in ways that do not directly indicate the source of the problem.

If you must recreate a deleted definition with the same name, run the **cdr check queue** command to make sure that the command is complete on all servers before recreating the definition.

You can also use a different name for the new object (for example, delete replicate **a.001** and recreate it as **a.002**) or wait until the delete action was successfully propagated through the entire Enterprise Replication system before you recreate the object.

Back up and restore of replication servers

You can back up and restore database servers that participate in Enterprise Replication.

Do not stop Enterprise Replication before performing a backup on database servers that participate in replication.

Warm restores are not permitted. You must perform a cold restore up to the current log of all relevant dbspaces on Enterprise Replication servers before resuming replication.

If the restore did not include all the log files from the replay position, or the system was not restored to the current log file, you must advance the log file unique ID past the latest log file unique ID prior to the restore, and then run the **cdr cleanstart** command followed by the **cdr sync replicate** command to synchronize the server.

Compression of replicated data

You can compress and uncompress data in replicated tables to reduce the amount of needed disk space.

You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Performing these operations on one Enterprise Replication server does not affect the data on any other Enterprise Replication server.

Attention: After you uncompress data on one server, do not remove any compression dictionaries that another Enterprise Replication server needs.
Related information:
Compression

Transaction processing impact

Many variables affect the impact that replicating data has on your transaction processing.

Replication volume
: To determine replication volume, you must estimate how many data rows change per day. For example, an application issues a simple INSERT statement that inserts 100 rows. If this table is replicated, Enterprise Replication must propagate and analyze these 100 rows before applying them to the targets.

Distributed transactions
: A *distributed transaction* is a transaction that commits data in a single transaction over two or more database servers.
Outside of the replication environment, Informix® uses a two-phase commit protocol to ensure that the transaction is either committed completely across all servers involved or is not committed on any server. For more information about the two-phase commit protocol, see the *IBM® Informix Administrator's Guide*.

  In a replication environment, when a distributed transaction is committed across the source servers, each part of the transaction that applies to the local server is written to the local logical logs. When Enterprise Replication retrieves the transaction from the logical logs and forms its transaction data, it is unable to identify the separate transaction data as the original single transaction.

  This situation might result in Enterprise Replication applying one transaction successfully while aborting another. Another result might be a time lapse between the application of one transaction and another (depending on how much transaction data is in each server's send queue and the state of the server).

Large transactions
: While Enterprise Replication is able to handle large transactions, it is optimized for small transactions. For best performance, avoid replicating large transactions. Large transactions are handled with a grouper paging file in temporary smart large objects. Enterprise Replication can process transactions up to 4 TB in size. For more information, see Setting Up the Grouper Paging File. You can view Enterprise Replication grouper paging statistics with the **onstat -g grp pager** command (see onstat -g grp: Print grouper statistics).

  Instead of using Enterprise Replication to perform a batch job, use BEGIN WORK WITHOUT REPLICATION to run the batch job locally on each database server. For more information, see Blocking Replication.

Related concepts:
Send queues and receive queues
Related tasks:
Preventing Memory Queues from Overflowing

SQL statements and replication

You can run most SQL statements while replication is active. For some statements, however, you must set alter mode or stop replication.

You can run the following SQL statements with no limitations while Enterprise Replication is active:

- ADD INDEX
- ALTER INDEX . . . TO CLUSTER

- ALTER FRAGMENT
- ALTER INDEX
- ALTER TABLE (except for the replication key)
- CREATE CLUSTER INDEX
- CREATE SYNONYM
- CREATE TRIGGER
- CREATE VIEW
- DROP INDEX
- DROP SYNONYM
- DROP TRIGGER
- DROP VIEW
- RENAME COLUMN
- RENAME DATABASE
- RENAME TABLE
- SET *object mode* (no disabling of replication key constraint)
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE
- TRUNCATE TABLE

After you define Enterprise Replication on a table by including that table as a participant in a replicate, you cannot exclusively lock a database that is involved in replication (or perform operations that require an exclusive lock). However, you can exclusively lock a table in a database.

You can rename both dbspaces and sbspaces while IBM® Informix® Enterprise Replication is active.

You cannot use the DROP TABLE SQL statement against a table that is included in a replicate.

You must first set alter mode with the **cdr alter** command before you can make these changes:

- Add shadow columns:
  - ALTER TABLE ... ADD CRCOLS;
  - ALTER TABLE ... ADD REPLCHECK;
  - ALTER TABLE ... ADD ERKEY
- Remove or disable the replication key constraint.
- Modify the replication key columns. For example, alter a column to add default values or other integrity constraints.
- Change the replication key from one or more columns to others. For example, if a replication key is defined on **col1**, you can change the replication key to **col2**.

You must stop replication before you make these changes:

- Drop conflict resolution shadow columns with ALTER TABLE ... DROP CRCOLS.
- Add or drop rowids.

SQL statements are limited to a maximum of 15000 bytes.

Related concepts:
Preparing Tables for a Consistency Check Index
Preparing Tables for Conflict Resolution
Alter, rename, or truncate operations during replication
Related tasks:
Changing or re-creating primary key columns
Preparing tables without primary keys
Related reference:
cdr alter
Related information:
Enterprise Replication shadow columns

Global language support for replication

You can replicate data in non-default locales.

An Enterprise Replication system can include databases in different locales, with the following restrictions:

- When you define a database server for Enterprise Replication, that server must be running in the U. S. English locale.
  The **syscdr** database on every Enterprise Replication server must be in the English locale.
- Replicate names can be in the locale of the database.

Code-set conversion with the GLS library requires only those code-set conversion files found in the $INFORMIXDIR/gls/cv9 directory.

- For U.S. English, locales are handled automatically by the installation and setup.
- For non-U.S. English locales, you might need to explicitly provide the locale and conversion files.

For information about how to specify a nondefault locale and other considerations related to GLS locales, see the *IBM® Informix GLS User's Guide*.

Related concepts:
Flexible Architecture
Related tasks:
Enabling code set conversion between replicates

Replication between multiple server versions

You can set up Enterprise Replication across servers of different version levels.

Enterprise Replication stores an internal version number that it communicates to other servers on initiating a connection with them. Each Enterprise Replication server instance can only use the features supported by its version level. Attempts to use features from later releases with previous versions of Enterprise Replication raise errors.

Schema design for Enterprise Replication

When you design the database and tables for replication, you must follow the requirements and restrictions for Enterprise Replication.

- Unbuffered Logging
- Table Types
  Enterprise Replication has restrictions on the types of tables that can participate in replication.
- Label-based access control
  You cannot apply label-based access control (LBAC) to a table participating in Enterprise Replication. Nor can you define an Enterprise Replication replicate on a table that is protected by LBAC.
- Out-of-Row Data
- Shadow columns
  Shadow columns are hidden columns on replicated tables that contain values that are supplied by the database server. The database server uses shadow columns to perform internal operations.
- Unique key for replication
  All tables that are replicated must have a replication key that is composed of one or more columns that uniquely identifies each row. The replication key must be the same on all servers that participate in the replicate. Typically, the replication key is a primary key constraint.
- Cascading Deletes
  If a table includes a cascading delete, when a parent row is deleted, the children are also deleted. If both the parent and child tables participate in replication, the deletes for both the parent and child are replicated to the target servers.
- Triggers
  A *trigger* is a database object that automatically sets off a specified set of SQL statements when a specified event occurs.
- Constraint and replication
  When you use constraints, ensure that the constraints you add at the target server are not more restrictive than the constraints at the source server. Discrepancies between constraints at the source and target servers can cause some rows to fail to be replicated.
- Sequence Objects
- The NLSCASE database property
  Enterprise Replication supports both case-sensitive databases and NLSCASE INSENSITIVE databases. (Databases created with the NLSCASE INSENSITIVE option ignore letter case in operations on NCHAR and NVARCHAR strings, and on strings of other character data types that are cast explicitly or implicitly to NCHAR or NVARCHAR data types.)
- Replicating Table Hierarchies
  To replicate tables that form a hierarchy, you must define a separate replicate for each table.
- Replication and data types
  Enterprise Replication supports built-in data types and user-defined data types, including row types and collection types.

Unbuffered Logging

Databases on all server instances involved in replication must be created with logging.

Enterprise Replication evaluates the logical log for transactions that modify tables defined for replication. If a table defined for replication resides in a database that uses buffered logging, the transactions are not immediately written to the logical log, but are instead buffered and then written to the logical log in a block of logical records. When this occurs, IBM® Informix® Enterprise Replication evaluates the buffer of logical-log records all at once. Buffered logging can require more time to flush the logs to disk. When you define a table for replication in a database created with unbuffered logging, Enterprise Replication can evaluate the transactions as they are produced.

Unlogged changes to a table, such as when data is added by a light append, can be replicated to other tables.

To create a database with unbuffered logging, use:

```
CREATE DATABASE db_name WITH LOG
```

To minimize impact on the system, IBM Informix Enterprise Replication uses buffered logging whenever possible, even if the database is defined as unbuffered. For more information, see the section on CREATE DATABASE in the *IBM Informix Database Design and Implementation Guide*.

Table Types

Enterprise Replication has restrictions on the types of tables that can participate in replication.

The following table types are not supported by Enterprise Replication:

- RAW tables
- Temporary tables
  Because the database server deletes temporary tables when an application terminates or closes the database, do not include these tables in your replication environment.

Enterprise Replication imposes the following operational limitations:

- Replication is restricted to base tables. That is, you cannot define a replicate on a view or synonym. A *view* is a synthetic table, a synthesis of data that exists in real tables and other views. A *synonym* is an alternative name for a table or a view. For more information about views and synonyms, see the *IBM® Informix® Database Design and Implementation Guide*.
- Replication is not inherited by any child tables in a typed hierarchy.

For more information about table types, see *IBM Informix Database Design and Implementation Guide*.

Label-based access control

You cannot apply label-based access control (LBAC) to a table participating in Enterprise Replication. Nor can you define an Enterprise Replication replicate on a table that is protected by LBAC.

Out-of-Row Data

Enterprise Replication collects out-of-row data for transmission after the user transaction has committed. Due to activity on the replicated row, the data might not exist at the time Enterprise Replication collects it for replication. In such cases, Enterprise Replication normally applies a NULL on the target system, unless the data is a smart large object. Therefore, you should avoid placing a NOT NULL constraint on any replicated column that includes out-of-row data.

If a column has smart large objects and the smart large object data does not exist when Enterprise Replication collects it for replication, then Enterprise Replication creates smart large objects with no data and zero size.

Shadow columns

Shadow columns are hidden columns on replicated tables that contain values that are supplied by the database server. The database server uses shadow columns to perform internal operations.

You can add shadow columns to your replicated tables with the CREATE TABLE or ALTER TABLE statement. To view the contents of shadow columns, you must explicitly specify the columns in the projection list of a SELECT statement; shadow columns are not included in the results of SELECT * statements.

The CRCOLS shadow columns, **cdrserver** and **cdrtime**, support conflict resolution. These two columns are hidden shadow columns because they cannot be indexed and cannot be viewed in the system catalog tables. In an update-anywhere replication environment, you must provide for conflict resolution using a conflict resolution rule. When you create a table that uses the time stamp, time stamp plus SPL, or delete wins conflict resolution rule, you must define the shadow columns, **cdrserver** and **cdrtime** on both the source and target replication servers. If you plan to use only the ignore or always-apply conflict resolution rule, you do not need to define the **cdrserver** and **cdrtime** shadow columns for conflict resolution.

The REPLCHECK shadow column, **ifx_replcheck**, supports faster consistency checking. This column is a visible shadow column because it can be indexed and can be viewed in the system catalog table. If you want to improve the performance of the **cdr check replicate** or **cdr check replicateset** commands, you can add the **ifx_replcheck** shadow column to the replicate table, and then create an index that includes the **ifx_replcheck** shadow column and your replication key columns.

The ERKEY shadow columns, **ifx_erkey1**, **ifx_erkey2**, and **ifx_erkey3**, are used as the replication key on replicated tables. If you create replicated tables through a grid, these ERKEY columns are automatically added.

Related concepts:
Conflict Resolution
Preparing Tables for Conflict Resolution
Shadow column disk space
Preparing Tables for a Consistency Check Index
Load and unload data

Unique key for replication

All tables that are replicated must have a replication key that is composed of one or more columns that uniquely identifies each row. The replication key must be the same on all servers that participate in the replicate. Typically, the replication key is a primary key constraint.

Replicated tables must use a primary key constraint, a unique index or constraint, or the ERKEY shadow columns as the replication key. If your table does not have a primary key or you want to change primary key values while replication is active, you can specify a different key as the replication key. Specify an existing unique index or

constraint, or the ERKEY shadow columns as the replication key when you create a replicate. A unique index and a unique constraint are equivalent as replication keys.

If you specify ERKEY columns as the replication key, Enterprise Replication creates a unique index and a unique constraint on the ERKEY columns. The ERKEY columns require storage space.

Important: Because primary key updates are sent as DELETE and INSERT statement pairs, avoid changing the primary key and updating data in the same transaction.
Related tasks:
Preparing tables without primary keys
Changing the replication key of a replicate

Cascading Deletes

If a table includes a cascading delete, when a parent row is deleted, the children are also deleted. If both the parent and child tables participate in replication, the deletes for both the parent and child are replicated to the target servers.

If the same table definition exists on the target database, Enterprise Replication attempts to delete the child rows twice. Enterprise Replication usually processes deletions on the parent tables first and then the children tables. When Enterprise Replication processes deletions on the children, an error might result, because the rows were already deleted when the parent was deleted. The table in Table 1 indicates how IBM® Informix® Enterprise Replication resolves cascading deletes with conflict resolution scopes and rules.

For more information on cascading deletes, see the ON DELETE CASCADE section in the *IBM Informix Guide to SQL: Syntax*.

Table 1. Resolving Cascade Deletes

Conflict-Resolution Rule	Conflict-Resolution Scope	Actions on Delete Errors
Time stamp	Row-by-row or transaction	Continue processing rest of the transaction
Delete wins	Row-by-row or transaction	Continue processing rest of the transaction
Ignore	Transaction	Abort entire transaction
Ignore	Row-by-row	Continue processing rest of the transaction

Triggers

A *trigger* is a database object that automatically sets off a specified set of SQL statements when a specified event occurs.

If the **--firetrigger** option is enabled on a replicate, any triggers defined on a table that participates in replication are invoked when transactions are processed on the target server. However, because Enterprise Replication only replicates the final result of a transaction, triggers execute only once on the target regardless of how many triggers execute on the source. In cases where the final evaluation of the transaction results in no replication (for example, an INSERT where the final row image is a DELETE, as shown in Table 1), no triggers execute on the target database.

If the same triggers are defined on both the source and target tables, any insert, update, or delete operation that the triggers generate are also sent to the target database server. For example, the target table might receive replicate data caused by a trigger that also executes locally. Depending on the conflict-resolution rule and scope, these operations can result in errors. To avoid this problem, define the replicate to not fire triggers on the target table.

You might want to design your triggers to take different actions depending on whether a transaction is being performed as part of Enterprise Replication. Use the **'cdrsession'** option of the **DBINFO()** function to determine if the transaction is a replicated transaction. The **DBINFO('cdrsession')** function returns `1` if the thread performing the database operation is an Enterprise Replication apply or sync thread; otherwise, the function returns `0`.

For more information on triggers, see Enabling Triggers and the CREATE TRIGGER section in *IBM® Informix® Guide to SQL: Syntax*.

Related information:
DBINFO Function

Constraint and replication

When you use constraints, ensure that the constraints you add at the target server are not more restrictive than the constraints at the source server. Discrepancies between constraints at the source and target servers can cause some rows to fail to be replicated.

If your replicated tables that have referential integrity constraints between them, synchronization the data through the replicate set. For replicate sets, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables).

When you synchronize data, rows that fail to be repaired due to discrepancies between constraints are recorded in the ATS and RIS files.

Sequence Objects

In bi-directional Enterprise Replication, if you replicate tables using sequence objects for update, insert, or delete operations, the same sequence values might be generated on different servers at the same time, leading to conflicts.

To avoid this problem, define sequence objects on each server so that the ranges of generated sequence values are distinct. For more information about the CREATE SEQUENCE and ALTER SEQUENCE statements of SQL, see the *IBM® Informix® Guide to SQL: Syntax*.

The NLSCASE database property

Enterprise Replication supports both case-sensitive databases and NLSCASE INSENSITIVE databases. (Databases created with the NLSCASE INSENSITIVE option ignore letter case in operations on NCHAR and NVARCHAR strings, and on strings of other character data types that are cast explicitly or implicitly to NCHAR or NVARCHAR data types.)

The database server does not prevent a case-sensitive database from being replicated by a database that has the NLSCASE INSENSITIVE property, nor the replication of an NLSCASE INSENSITIVE database by a case-sensitive database. No warning or exception is issued by the database server in either of these cases when you define replication participants.

These two types of database behave differently, however, in operations that classify NCHAR and NVARCHAR strings as duplicates or as distinct values, if the character strings that are being compared differ only in letter case. It is the user's responsibility to make sure that replication participants with different NLSCASE attributes will not cause exceptions or unexpected behavior when replicating the results of operations like the following on NCHAR or NVARCHAR data:

- sorting and collation
- foreign key and primary key dependencies
- enforcing unique constraints
- clustered indexes
- access-method optimizer directives
- queries with WHERE predicates
- queries with UNIQUE or DISTINCT specifications in the projection clause
- queries with ORDER BY clauses
- queries with GROUP BY clauses
- cascading DELETE operations
- table or index storage fragmentation BY EXPRESSION
- table or index storage fragmentation BY LIST
- data distributions from UPDATE STATISTICS operations

To avoid the risk of consistency problems that can result from differences in case-sensitivity, the following policy might be useful when you define replication pairs:

- Replicate case-sensitive databases only with case-sensitive databases.
- Replicate NLSCASE INSENSITIVE databases only with NLSCASE INSENSITIVE databases.

Related information:
Duplicate rows in NLSCASE INSENSITIVE databases

Replicating Table Hierarchies

To replicate tables that form a hierarchy, you must define a separate replicate for each table.

If you define a replicate on a super table, Enterprise Replication does not automatically create implicit replicate definitions on the subordinate tables.
Tip: Enterprise Replication does not require that the table hierarchies be identical on the source and target servers.
You must use conflict resolution uniformly for all tables in the hierarchy. In other words, either no conflict resolution for all tables or conflict resolution for all tables.

Replication and data types

Enterprise Replication supports built-in data types and user-defined data types, including row types and collection types.

If you use SERIAL, SERIAL8, or BIGSERIAL data types, you must be careful when defining serial columns.

For non-master replicates, Enterprise Replication does not verify the data types of columns in tables that participate in replication. The replicated column in a table on the source database server must have the same data type as the corresponding column on the target server. The exception to this rule is cross-replication between simple large objects and smart large objects. By using master replicates, you can verify that all participants in a replicate have columns with matching data types. Master replicates also allow verification that each participant contains all replicated columns, and optionally that column names are the same on each participant.

- Replicating on Heterogeneous Hardware
- Serial data types and replication keys
  You can use a serial data type as a replication key, but you must ensure that the values are unique across all replication servers.

- Replication of TimeSeries data types
  You can replicate tables that have columns with **TimeSeries** data types. You must prepare all replication servers and create time series instances before you create replicates that include **TimeSeries** columns.
- Replication of large objects
  How Enterprise Replication handles simple and smart large objects depends on how the objects are stored.
- Replication of opaque user-defined data types
  Opaque data types can be replicated, but have certain restrictions.

Related concepts:
Flexible Architecture
Serial data types and replication keys

Replicating on Heterogeneous Hardware

Enterprise Replication supports all primitive data types across heterogeneous hardware. If you define a replicate that includes non-primitive data types (for example, BYTE and TEXT data), the application must resolve data-representation issues that are architecture dependent.

If you use floating-point data types with heterogeneous hardware, you might need to use IEEE floating point or canonical format for the data transfers. For more information, see Using the IEEE Floating Point or Canonical Format.

Serial data types and replication keys

You can use a serial data type as a replication key, but you must ensure that the values are unique across all replication servers.

If you plan to use serial data types (SERIAL, SERIAL8, or BIGSERIAL) as the replication key for a table, the same serial value might be generated on two servers at the same time. Use the CDR_SERIAL configuration parameter to generate non-overlapping values for serial columns across all database servers in your replication environment. Set CDR_SERIAL in the onconfig file for each primary source database server in the replication system.

You do not need to set the CDR_SERIAL configuration parameter if your replication key has multiple columns and the other columns identify the server on which each row is created.

Related concepts:
Replication and data types
Related reference:
Set configuration parameters for replication
CDR_SERIAL Configuration Parameter

Replication of TimeSeries data types

You can replicate tables that have columns with **TimeSeries** data types. You must prepare all replication servers and create time series instances before you create replicates that include **TimeSeries** columns.

Server preparation

All database servers must run Informix® version 12.10 or later.

Before you create a replicate, do the following tasks on all replication servers that will participate in replicating time series data:

1. Set the CDR_TSINSTANCEID configuration parameter to a different value on every replication server to ensure that time series instance IDs do not overlap. You cannot replicate time series instances that were created before you set the CDR_TSINSTANCEID configuration parameter.
2. Create containers that have the same names on all replication servers. You cannot use containers that are created automatically or rolling window containers. If you add containers after replication is set up, add containers with the same names on all replication servers at the same time. The containers can be in different locations on each server.
3. Create the same time series calendars that have the same names on all replication servers.
4. Create time series tables on all replication servers. You cannot use the option to automatically create replicated tables when you define a replicate or template. You cannot nest a **TimeSeries** data type within a **TimeSeries** data type.
5. Create time series instances. You must specify the container name.

Tip: You can quickly set up your replication servers by doing all but the first of these steps through a grid, however, all grid servers must be running Informix version 12.10 or later.

Rules for defining a replicate

You must follow these rules when you define a replicate for a table that contains a **TimeSeries** column:

- The replicate must be a mastered replicate.
- The Projection list in the participant definition must include all columns in the table.

- The WHERE clause in the participant definition cannot include a **TimeSeries** column.
- You cannot define a participant as send-only.
- The conflict resolution rule must be always-apply.
- The replication key cannot include an opaque data type.
- You cannot enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets.
- You cannot use the **--autocreate** option to create tables that have **TimeSeries** columns. You must create time series tables on all servers before you define replicates.
- You cannot generate ATS or RIS files in XML format. ATS and RIS files must be in text format.

Restrictions

You cannot run a shared query on a table that includes a **TimeSeries** column. You can, however, run grid queries on a virtual table that is based on a table that has a **TimeSeries** column.

You cannot use the following commands on replicates that include **TimeSeries** columns:

- **cdr alter**
- **cdr remaster**
- **cdr start sec2er**
- **cdr swap shadow**

You cannot use the following options when you check or repair inconsistencies on a replicate that includes a **TimeSeries** column:

- The **--deletewins** option in the **cdr check replicate** or **cdr check replicateset** command
- The **--extratargetrows=merge** option in the **cdr sync replicate**, **cdr sync replicateset**, **cdr check replicate**, or **cdr check replicateset** command
- The **--since** option in the **cdr check replicate** or **cdr check replicateset** command
- The **--timestamp** option in the **cdr check replicate** or **cdr check replicateset** command
- The **--where** option with a **TimeSeries** column in the WHERE clause in the **cdr check replicate** command

Although you can add and index an **ifx_replcheck** column on a replicated table that includes a **TimeSeries** column, the speed of consistency checking is not affected.

Related reference:
cdr define replicate

Replication of large objects

How Enterprise Replication handles simple and smart large objects depends on how the objects are stored.

Enterprise Replication replicates the following types of large objects:

- Simple large object data types (TEXT and BYTE)
  You can store simple large objects either in the tblspace with the rest of the table columns (in a dbspace) or in a blobspace. Simple large objects in tblspaces are logged in the logical log and therefore, Enterprise Replication can evaluate the data for replication directly.
- Smart large object data types (BLOB and CLOB)
  You must store smart large objects in sbspaces. Enterprise Replication cannot evaluate large object data that is stored in a blobspace or sbspace; instead, Enterprise Replication uses information about the large object that is stored in the row to evaluate whether the objects need to be replicated.

By default, Enterprise Replication does not include columns that contain unchanged large objects in replicated rows.

Enterprise Replication allows cross-replication between simple large objects and smart large objects. For example, you can replicate a simple large object on the source database server to a smart large object on the target server or vice versa.

If Enterprise Replication processes a row and discovers undeliverable large object columns, the following actions can occur:

- Any undeliverable columns are set to NULL if the replication operation is an INSERT and the row does not already exist at the target.
- The old value of the local row is retained if the replication operation is an UPDATE or if the row already exists on the target.

- Replicating Simple Large Objects from Tblspaces
  Enterprise Replication evaluates simple large object data that is stored in a tblspace independently from the rest of its row.
- Replication of large objects from blobspaces or sbspaces
  Enterprise Replication retrieves the large object data directly from the blobspace or sbspace and then sends the data to the target database server.

Replicating Simple Large Objects from Tblspaces

Enterprise Replication evaluates simple large object data that is stored in a tblspace independently from the rest of its row.

Simple large object data that is stored in tblspaces (rather than in blobspaces) is placed in the logical log. Enterprise Replication reads the logical log to capture and evaluate the data for potential replication.

By default, Enterprise Replication performs time stamp and delete wins conflict detection and resolution at the row level. However, in some cases, simple large object data that is stored in a tblspace (rather than in a blobspace) is accepted by the target server even if the row is rejected.

For simple large objects, if the column on the target database server is also stored in a tblspace, Enterprise Replication evaluates the values of the shadow columns, **cdrserver** and **cdrtime**, in the source and target columns and uses the following logic to determine if the data is to be applied:

- If the column of the replicated data has a time stamp that is greater than the time stamp of the column on the local row, the data for the column is accepted for replication.
- If the server ID and time stamp of the replicated column are equal to the server ID and time stamp on the column on the local row, the data for the column is accepted for replication.
- If there is no SPL conflict-resolution rule and the time stamps are equal, then Enterprise Replication applies the data to the row with the lowest CDR server ID.

If you use the SPL conflict resolution, simple large objects that are stored in tblspaces are handled differently than large objects in blobspaces.

Related concepts:
Delete wins conflict resolution rule
Time stamp conflict resolution rule
SPL Conflict Resolution for Large Objects

Replication of large objects from blobspaces or sbspaces

Enterprise Replication retrieves the large object data directly from the blobspace or sbspace and then sends the data to the target database server.

It is possible that a transaction subsequent to the transaction that is being replicated can modify or delete a simple or smart large object that Enterprise Replication is trying to retrieve. If Enterprise Replication encounters a row whose large object (simple or smart) was modified or deleted by a subsequent transaction, Enterprise Replication does not send the data in the large object. In most cases, the subsequent transaction that modified or deleted the large object is also replicated, so the data again becomes consistent when that transaction is replicated. The data in the large object is inconsistent for only a short time.

The following conditions apply to replicating large objects that are stored in blobspaces or sbspaces:

- Enterprise Replication does not support replication of large object updates performed outside of a row update.
- After you update a large object that is referenced explicitly in the table schema, you must update the referencing row before Enterprise Replication can replicate the updated smart large object. For example:

  ```
  UPDATE table_name SET large_object_column = x
  ```

- Enterprise Replication replicates updates to in-place smart large objects by sending a new copy of the entire smart large object. Enterprise Replication does not send only the logged changes to update smart large objects.
- Enterprise Replication does not support sharing out-of-row data (multiple references to a large object) during replication. If you try to replicate multiple references to the same large object on the source database server, Enterprise Replication does not re-create those references on the target database server. Instead, Enterprise Replication creates multiple large objects on the target database server.

Related concepts:
SPL Conflict Resolution for Large Objects

Replication of opaque user-defined data types

Opaque data types can be replicated, but have certain restrictions.

You must install and register UDTs and their associated support routines on all database servers that participate in Enterprise Replication before starting replication. If you combine Enterprise Replication with high-availability clusters, you must install UDTs on both the primary and secondary database servers, but only register them on the primary database server.

UDT support functions

If you plan to replicate opaque UDTs, the UDT designer must provide the following types of support functions:

- The **streamwrite()** and **streamread()** functions
  The purpose of these functions is similar to the existing **send()** and **receive()** functions provided for client/server transmissions. For information about writing these support functions, see the section on Enterprise Replication stream support functions in the *IBM® Informix® DataBlade API Programmer's Guide*.

  When a row that includes any UDT columns to queue to the target system is prepared for replication, Enterprise Replication calls the **streamwrite()** function on each UDT column. The function converts the UDT column data from the in-server representation to a representation that can be sent over the network. Enterprise Replication replicates the column without understanding the internal representation of the UDT.

  On the target server, Enterprise Replication calls the **streamread()** function for each UDT column that it transmitted by the **streamwrite()** function.

- The **compare()** function and its supporting **greaterthan()**, **lessthan()**, and **equal()** functions
  Enterprise Replication uses comparison functions to determine whether a replicated column is altered. For example, the comparison functions are used when the replicate definition specifies to replicate only changed columns instead of full rows.

  When you define a **compare()** function, you must also define the **greaterthan()**, **lessthan()**, **equal()**, or other functions that use the **compare()** function.

  For more information about writing these support functions, see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Requirements during replication

The following requirements apply to replicating opaque data types:

- The WHERE clause of the SELECT statement of the participant modifier can reference an opaque UDT if the UDT is always stored in row.
- Any UDRs in a WHERE clause can use only parameters whose values can be extracted fully from the logged row images, plus any optional constants.
- All of the columns in the SELECT statement of each participant definition must be actual columns in that table. Enterprise Replication does not support virtual columns (results of UDRs on table columns).
- You cannot use SPL routines for conflict resolution if the replicate includes any UDTs in the SELECT statement or if the replicate is defined to replicate only changed columns.
- You can define replicates on tables that contain one or more UDT columns as the replication key.

Replication system design

When you design a replication system, you make three main decisions: how the information flows between servers, how to resolve conflicts between replicated data, and the topology of the network of servers.

- Primary-Target Replication System
- Update-Anywhere Replication System
  In update-anywhere replication, changes made on any participating database server are replicated to all other participating database servers. This capability allows users to function autonomously even when other systems or networks in the replication system are not available.
- Conflict Resolution
- Choosing a Replication Network Topology

Primary-Target Replication System

In the primary-target replication system, the flow of information is in one direction.

In primary-target replication, all database changes originate at the primary database and are replicated to the target databases. Changes at the target databases are not replicated to the primary.

A primary-target replication system can provide one-to-many or many-to-one replication:

- One-to-many replication
  In one-to-many (*distribution*) replication, all changes to a primary database server are replicated to many target database servers. Use this replication model when information gathered at a central site must be disseminated to many scattered sites.

- Many-to-one replication
  In many-to-one (*consolidation*) replication, many primary servers send information to a single target server. Use this replication model when many sites are gathering information (for example, local field studies for an environmental study) that needs to be centralized for final processing.

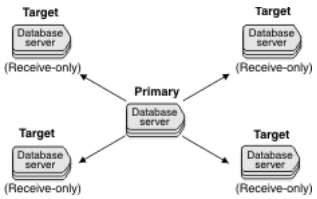
- Primary-Target Data Dissemination
  Data dissemination supports business needs where data is updated in a central location and then replicated to servers that only receive data and do not update data.
- Data consolidation
  Businesses can choose to consolidate data into one or more central database servers.
- Workload Partitioning
  Workload partitioning gives businesses the flexibility of assigning data ownership at the table-partition level, rather than within an application.
- Workflow Replication
- Primary-Target Considerations

Related concepts:
Flexible Architecture
Participant definitions
Related reference:
Participant and participant modifier

Primary-Target Data Dissemination

Data dissemination supports business needs where data is updated in a central location and then replicated to servers that only receive data and do not update data.

This method of distribution can be useful for online transaction processing (OLTP) systems where data is required at several sites, but because of the large amounts of data, read/write capabilities at all sites would slow the performance of the application. The following figure illustrates data dissemination.
Figure 1. Data Dissemination in a Primary-Target Replication System



You specify that a server only receives information when you define the participant for the server as part of the replicate definition. You can specify that all replicates on a server only receive information when you modify the server definition.

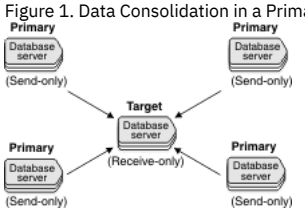
Related reference:
cdr modify server
Participant and participant modifier

Data consolidation

Businesses can choose to consolidate data into one or more central database servers.

Data consolidation allows the migration of data from several database servers to a central database server. For example, several retail stores can replicate inventory and sales information to headquarters. The retail stores do not need information from other stores but headquarters needs the total inventory and sales of all stores.

In the following figure, the remote locations only send data while a single central database server only receives data.
Figure 1. Data Consolidation in a Primary-Target Replication System



You can also use data consolidation to replicate data from many database servers to more than one central database server. For example, a retail chain has two central database servers, one for the eastern half of the United States, and one for the western half of the United States. The retail stores replicate data to their designated central server and the two central servers replicate data to each other. In this configuration, the replication servers in the retail stores only send data to the central servers, but the central servers both send and receive data.

Businesses can use data consolidation to replicate OLTP data to a dedicated computer for decision support (DSS) analysis. For example, data from several OLTP systems can be replicated to a DSS system for read-only analysis.

The replication key for every replicated row must be unique among the multiple primary database servers.

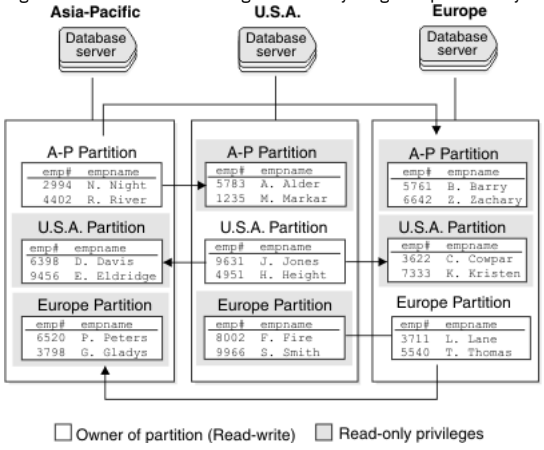
You specify that a server only sends information when you define the participant for the server as part of the replicate definition. You can specify that all replicates on a server only send information when you modify the server definition.

Related reference:
cdr modify server
Participant and participant modifier

Workload Partitioning

Workload partitioning gives businesses the flexibility of assigning data ownership at the table-partition level, rather than within an application.

The following figure illustrates workload partitioning.
Figure 1. Workload Partitioning in a Primary-Target Replication System



The replication model matches the partition model for the **employee** tables. The Asia-Pacific database server owns the partition and can therefore update, insert, and delete employee records for personnel in its region. The changes are then propagated to the US and European regions. The Asia-Pacific database server can query or read the other partitions locally, but cannot update those partitions locally. This strategy applies to other regions as well.

Workflow Replication

Unlike the data dissemination model, in a workflow-replication system, the data moves from site to site. Each site processes or approves the data before sending it on to the next site.

Figure 1. A Workflow-Replication System Where Update Authority Moves From Site to Site

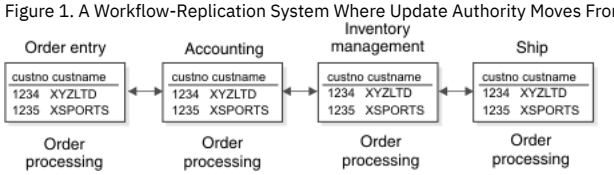


Figure 1 illustrates an order-processing system. Order processing typically follows a well-ordered series of steps: orders are entered, approved by accounting, inventory is reconciled, and the order is finally shipped.

In a workflow-replication system, application modules can be distributed across multiple sites and databases. Data can also be replicated to sites that need read-only access to the data (for example, if order-entry sites want to monitor the progress of an order).

A workflow-replication system, like the primary-target replication system, allows only unidirectional updates. Many facts that you need to consider for a primary-target replication system should also be considered for the workflow-replication system.

However, unlike the primary-target replication system, availability can become an issue if a database server goes down. The database servers in the workflow-replication system rely on the data updated at a previous site. Consider this fact when you select a workflow-replication system.

Related concepts:
Controlling the replication of large objects

Primary-Target Considerations

Consider the following factors when you select a primary-target replication system:

- Administration
  Primary-target replication systems are the easiest to administer because all updates are unidirectional and therefore, no data update conflicts occur. Primary-target replication systems use the *ignore* conflict-resolution rule. See Conflict resolution rule.
- Capacity planning
  All replication systems require you to plan for capacity changes. For more information, see Preparing Data for Replication.
- High-availability planning
  In the primary-target replication system, if a target database server or network connection goes down, Enterprise Replication continues to log information for the database server until it becomes available again. If a database server is unavailable for some time, you might want to remove the database server from the replication system. If the unavailable database server is the read-write database server, you must plan a course of action to change read-write capabilities on another database server.

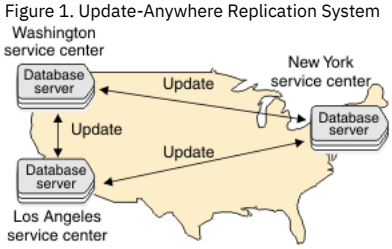
  If you require a fail-safe replication system, you should select a high-availability replication system. For more information, see High-availability replication systems.

Update-Anywhere Replication System

In update-anywhere replication, changes made on any participating database server are replicated to all other participating database servers. This capability allows users to function autonomously even when other systems or networks in the replication system are not available.

The following figure illustrates an update-anywhere replication system where the service centers in Washington, New York, and Los Angeles each replicate changes to the other two servers.

Figure 1. Update-Anywhere Replication System



Because each service center can update a copy of the data, conflicts can occur when the data is replicated to the other sites. To resolve update conflicts, Enterprise Replication uses conflict resolution.

Review the following information before you select your update-anywhere replication system:

- Administration
  Update-anywhere replication systems allow peer-to-peer updates, and therefore require conflict-resolution. Update-anywhere replication systems require more administration than primary-target replication systems.
- Information consistency
  Some risk is associated with delivering consistent information in an update-anywhere replication system. You determine the amount of risk based on the type of conflict-resolution rules and routines you choose for resolving conflicts. You can configure an update-anywhere replication system where no data is ever lost; however, you might find that other factors (for example, performance) outweigh your need for a fail-safe mechanism to deliver consistent information.
- Capacity Planning
  All replication systems require you to plan for capacity changes and prepare the data for replication. If you choose a time-based conflict resolution rule, you need to provide space for delete tables and add shadow columns to replicated tables.
- High Availability
  If any of your database servers are critical, consider using high-availability clusters to provide backup servers.

Related concepts:
Disk Space for Delete Tables
Shadow column disk space
Preparing Data for Replication
High-availability replication systems
Flexible Architecture
Related tasks:
Specifying Conflict Resolution Rules and Scope

Conflict Resolution

When multiple database servers try to update the same row simultaneously (the time stamp for both updates is the same GMT time), a collision occurs. For more information, see Applying replicated data. Enterprise Replication must determine which new data to replicate. To solve conflict resolution, you must specify the following for each replicate:

- A conflict-resolution rule
- The scope of the rule

- Conflict resolution rule
  The conflict resolution rule determines how conflicts between replicated transactions are resolved.
- Conflict Resolution Scope
  Each conflict-resolution rule behaves differently depending on the scope.

Related concepts:
Shadow columns
Time synchronization
Related reference:
Replicate only changed columns
cdr define replicate

Conflict resolution rule

The conflict resolution rule determines how conflicts between replicated transactions are resolved.

Enterprise Replication supports the following conflict resolution rules.

Conflict Resolution Rule	Effect
Ignore	Enterprise Replication does not attempt to resolve conflicts.
Time stamp	The row or transaction with the most recent time stamp is applied.
SPL routine	Enterprise Replication uses a routine written in SPL (Stored Procedure Language) that you provide to determine which data is applied.
Time stamp with SPL routine	If the time stamps are identical, Enterprise Replication uses an SPL routine that you provide to resolve the conflict.
Delete wins	DELETE and INSERT operations win over UPDATE operations; otherwise the row or transaction with the most recent time stamp is applied.
Always-apply	Enterprise Replication does not attempt to resolve conflicts. You must use the always-apply rule when you replicate **TimeSeries** data types.

- Ignore Conflict-Resolution Rule
  The ignore conflict-resolution rule does not attempt to detect or resolve conflicts.

- Time stamp conflict resolution rule
  The time stamp rule evaluates the latest time stamp of the replication against the target and determines how to resolve any conflict.
- SPL Conflict Resolution Rule
  You can write an SPL routine as a primary conflict resolution rule or as secondary conflict resolution rule to the time stamp conflict resolution rule.
- Delete wins conflict resolution rule
  The delete wins rule ensures that DELETE and INSERT operations win over UPDATE operations and that all other conflicts are resolved by comparing time stamps.
- Always-Apply Conflict-Resolution Rule
  The always-apply conflict-resolution rule does not attempt to detect or resolve conflicts.

Related tasks:
Specifying Conflict Resolution Rules and Scope
Creating replicated tables through a grid

Ignore Conflict-Resolution Rule

The ignore conflict-resolution rule does not attempt to detect or resolve conflicts.

A row or transaction either applies successfully or it fails. A row might fail to replicate because of standard database reasons, such as a deadlock situation, when an application locks rows. Use the ignore conflict-resolution rule only with a primary-target replication system. If you use ignore with an update-anywhere replication system, your data might become inconsistent.

The ignore conflict-resolution rule can be used only as a primary conflict- resolution rule and can have either a transaction or a row scope (as described in Conflict Resolution Scope).

The following table describes how the ignore conflict resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 1. Ignore Conflict-Resolution Rule

Row Exists in Target?	INSERT	UPDATE	DELETE
No	Apply row	Discard row	Discard row
Yes	Discard row	Apply row	Apply row

When a replication message fails to apply to a target, you can spool the information to one or both of the following directories:

- Aborted-transaction spooling (ATS)
  If selected, all buffers in a failed replication message that compose a transaction are written to this directory.
- Row-information spooling (RIS)
  If selected, the replication message for a row that cannot be applied to a target is written to this directory.

For more information, see Failed Transaction (ATS and RIS) Files.

Time stamp conflict resolution rule

The time stamp rule evaluates the latest time stamp of the replication against the target and determines how to resolve any conflict.

All time stamps and internal computations are performed in Greenwich mean time (GMT). The time stamp conflict resolution rule assumes time synchronization between cooperating Enterprise Replication servers.

The time stamp resolution rule behaves differently depending on which scope is in effect:

- Row scope
  Enterprise Replication evaluates one row at a time. The row with the most recent time stamp wins the conflict and is applied to the target database servers. If an SPL routine is defined as a secondary conflict-resolution rule, the routine resolves the conflict when the row times are equal.
- Transaction scope
  Enterprise Replication evaluates the most recent row-update time among all the rows in the replicated transaction. This time is compared to the time stamp of the appropriate target row. If the time stamp of the replicated row is more recent than the target, the entire replicated transaction is applied. If a routine is defined as a secondary conflict resolution rule, it is used to resolve the conflict when the time stamps are equal.

A secondary routine is run only if Enterprise Replication evaluates rows and discovers equal time stamps.

If no secondary conflict-resolution rule is defined and the time stamps are equal, the transaction from the database server with the lower value in the **cdrserver** shadow column wins the conflict.

The following table shows how a conflict is resolved based on the latest time stamp with row scope. The time stamp T_{last_update} (the time of the last update) represents the row on the target database server with the last (most recent) update. The time stamp T_{repl} (the time when replication occurs) represents the time stamp on the incoming row.

Enterprise Replication first checks to see whether a row with the same replication key exists in either the target table or its corresponding delete table.

If the row exists, Enterprise Replication uses the latest time stamp to resolve the conflict.

The following table describes how the time stamp conflict resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 1. Conflict Resolution Based on the Time Stamp

Row Exists on Target?	Time Stamp	INSERT	UPDATE	DELETE
No	n/a	Apply row	Apply row (Convert UPDATE to INSERT)	Apply row (INSERT into Enterprise Replication delete table)
Yes	$T_{last\_update} < T_{repl}$	Apply row (Convert INSERT to UPDATE)	Apply row	Apply row
Yes	$T_{last\_update} > T_{repl}$	Discard row	Discard row	Discard row
Yes	$T_{last\_update} = T_{repl}$	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.

Important: Do not remove the delete tables that are created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.
To use time stamp conflict resolution for repairing inconsistencies with the **cdr check replicate** or **cdr check replicateset** command, include the **--timestamp** option with the **--repair** option. If you temporarily stop replication on a server whose replicates use the time stamp conflict resolution rule, disable the replication server with the **cdr disable server** command. When you disable a server, information about deleted rows is kept in the delete tables to be used during the time stamp repair after the server is enabled.

Related concepts:
Conflict Resolution Scope
Time synchronization
Delete wins conflict resolution rule
Replicating Simple Large Objects from Tblspaces
Repair inconsistencies by time stamp
Related reference:
cdr disable server

SPL Conflict Resolution Rule

You can write an SPL routine as a primary conflict resolution rule or as secondary conflict resolution rule to the time stamp conflict resolution rule.

You have complete flexibility to determine which row prevails in the database when you create an SPL routine for conflict resolution. However, for most users, the time stamp conflict resolution rule provides sufficient conflict resolution. You can also use SPL routine to save information about the transactions that were discarded during conflict resolution.

SPL routines must follow the following guidelines:

- The owner of an SPL routine that is used for conflict resolution must be the same as the owner of the replicated table.
- Routines for conflict resolution must be in SPL. Enterprise Replication does not allow user-defined routines in C or in Java™.
- You cannot use an SPL routine or a time stamp with an SPL routine if the replicate is defined to replicate only changed columns or the replicated table contains any extensible data types. See Replicate only changed columns.

Enterprise Replication passes the following information to an SPL routine as arguments.

Argument	Description
Server name [CHAR(18)]	From the local target row NULL if local target row does not exist
Time stamp (DATETIME YEAR TO SECOND)	From the local target row NULL if local target row does not exist
Local delete-table indicator [CHAR(1)] or Local key delete-row indicator [CHAR(1)]	Y indicates that the origin of the row is the delete table. K indicates that the origin of the row is the replicate-key delete row. If a conflict occurs while a replication key row is being deleted, because the local row with the old key no longer exists, the received key delete row is passed as the local row (using the seventh argument, local row data). The received key insert row is passed to the stored procedure as the replicated row using the eighth argument.
Server name [CHAR(18)]	Of the replicate source
Time stamp (DATETIME YEAR TO SECOND)	From the replicated row
Replicate action type [CHAR(1)]	`I` - insert `D` - delete `U` - update
Local row data that is returned in regular SQL format	Where the regular SQL format is taken from the SELECT clause of the participant list
Replicate row data after-image that is returned in regular SQL format	Where the regular SQL format is taken from the SELECT clause of the participant list

The routine must set the following arguments before the routine can be applied to the replication message.

Argument	Description

Argument	Description
An indicator of the database operation to be performed [CHAR(1)]	Same as the replicate action codes with the following additional codes • A - Accept the replicated row and apply the column values returned by the SPL routine. For example, if Enterprise Replication receives an insert and the row exists locally, the insert is converted to an update • S - Accept the replicated row and apply the column values as received from the other site. For example, if Enterprise Replication receives an insert and the row exists locally, the insert fails at the time Enterprise Replication tries to apply the transaction to the database, and the transaction aborts with an SQL error. • O - Discard the replicated row. • X - Abort the transaction.
A non-zero integer value to request logging of the conflict resolution and the integer value in the spooling files (INTEGER)	Logging value takes effect only if logging is configured for this replicate.
The columns of the row to be applied to the target table replicate action type in regular SQL format	This list of column values is not parsed if the routine returns one of the following replicate action types: S, O, or X.

You can use the arguments to develop application-specific routines. For example, you can create a routine in which a database server always wins a conflict regardless of the time stamp.

The following list includes some items to consider when you use an SPL routine for conflict resolution:

- Any action that a routine takes as a result of replication does not replicate.
- You cannot use an SPL routine to start another transaction.
- Frequent use of routines might affect performance.

In addition, you must determine when the SPL routine runs:

- An optimized SPL routine is called only when a collision is detected and the row to be replicated fails to meet one of the following two conditions:
  - It is from the same database server that last updated the local row on the target table.
  - It has a time stamp greater than or equal to that of the local row.
- A nonoptimized SPL routine runs every time Enterprise Replication detects a collision. By default, SPL routines are nonoptimized.

For information on specifying that the SPL routine is optimized, see Conflict Options.
Tip: Do not assign a routine that is not optimized as a primary conflict resolution rule for applications that usually insert rows successfully.

- SPL Conflict Resolution for Large Objects
  If the replicate is defined with an SPL conflict-resolution rule, the SPL routine must return the desired action for each smart large object (BLOB or CLOB) and simple large object (BYTE or TEXT) column.

SPL Conflict Resolution for Large Objects

If the replicate is defined with an SPL conflict-resolution rule, the SPL routine must return the desired action for each smart large object (BLOB or CLOB) and simple large object (BYTE or TEXT) column.

When the routine is invoked, information about each large object column is passed to the routine as five separate fields. The following table describes the fields.

Argument	Description
Column size (INTEGER)	The size of the column (if data exists for this column). NULL if the column is NULL.
BLOB flag [CHAR(1)]	For the local row, the field is always NULL. For the replicated row: • D indicates that the large object data is sent from the source database server. • U indicates that the large object data is unchanged on the source database server.
Column type [CHAR(1)]	• P indicates tblspace data. • B indicates blobspace data. • S indicates sbspace data.
ID of last update server [CHAR(18)]	The ID of the database server that last updated this column for tblspace data. For blobspace data: NULL For sbspace data: NULL
Last update time (DATETIME YEAR TO SECOND)	For tblspace data: The date and time when the data was last updated. For blobspace data: NULL For sbspace data: NULL

If the routine returns an action code of `A`, `D`, `I`, or `U`, the routine parses the return values of the replicated columns. Each large object column can return a two-character field.

The first character defines the desired option for the large object column, as the following table shows.

Value	Function
`C`	Performs a time-stamp check for this column as used by the time-stamp rule.
`N`	Sets the replicate column to NULL.
`R`	Accepts the replicated data as it is received.
`L`	Retains the local data.

The second character defines the desired option for blobspace or sbspace data if the data is found to be undeliverable, as the following table shows.

Value	Function
`N`	Sets the replicated column to NULL.
`L`	Retains the local data (default).
`O`	Aborts the row.
`X`	Aborts the transaction.

Related concepts:
Replicating Simple Large Objects from Tblspaces
Replication of large objects from blobspaces or sbspaces

Delete wins conflict resolution rule

The delete wins rule ensures that DELETE and INSERT operations win over UPDATE operations and that all other conflicts are resolved by comparing time stamps.

All time stamps and internal computations are performed in Greenwich mean time (GMT). The delete wins conflict-resolution rule assumes time synchronization between cooperating Enterprise Replication servers.

The delete wins rule is similar to the time stamp rule except that it prevents upsert operations and does not use a secondary conflict resolution rule. The delete wins rule prevents upsert operations that results from an UPDATE operation that is converted to an INSERT operation because the row to update was not found on the target server. An upsert operation can occur if a row is deleted from a target server before an UPDATE operation is processed on that target server or if an UPDATE operation was processed by the target server before the INSERT operation for that row. Depending on your business logic, upsert operations might violate referential integrity.

The delete wins rule prevents upsert operations in the following ways:

- If a row is deleted on a replication server, that row is deleted on all other replication servers, regardless of whether an UPDATE operation to that row occurred after the delete.
- If an UPDATE operation to a row is received before its INSERT operation, the UPDATE operation fails and generates and ATS or RIS file. The INSERT operation succeeds, but results in data inconsistency. To repair the inconsistency, run the **cdr check replicate** command with the **--repair** option.

The delete wins rule handles time stamp conflicts differently depending on which scope is in effect:

- Row scope
  Enterprise Replication evaluates one row at a time. The row with the most recent time stamp wins the conflict and is applied to the target database servers.
- Transaction scope
  Enterprise Replication evaluates the most recent row-update time among all the rows in the replicated transaction. This time is compared to the time stamp of the appropriate target row. If the time stamp of the replicated row is more recent than the target, the entire replicated transaction is applied.

If the time stamps are equal, the transaction from the database server with the lower value in the **cdrserver** shadow column wins the conflict.

The following table shows how a conflict is resolved with the delete wins rule with row scope. The time stamp T_{last_update} (the time of the last update) represents the row on the target database server with the last (most recent) update. The time stamp T_{repl} (the time when replication occurs) represents the time stamp on the incoming row.

Enterprise Replication first checks to see if a row with the same replication key exists in either the target table or its corresponding delete table. If the row exists, Enterprise Replication uses the latest time stamp to resolve the conflict.

The following table describes how the delete wins conflict resolution rule handles INSERT, UPDATE, and DELETE operations that are performed on the source server.

Table 1. Conflict Resolution Based on the Time Stamp

Row Exists on Target?	Time Stamp	INSERT	UPDATE	DELETE
No	n/a	Apply row	Discard row and generate and ATS or RIS file	Apply row (INSERT into Enterprise Replication delete table)
Yes	$T_{last\_update} < T_{repl}$	Apply row (Convert INSERT to UPDATE)	Apply row	Apply row
Yes	$T_{last\_update} > T_{repl}$	Discard row	Discard row	Apply row
Yes	$T_{last\_update} = T_{repl}$	The server with the lower value in the **cdrserver** shadow column wins the conflict.	The server with the lower value in the **cdrserver** shadow column wins the conflict.	The server with the lower value in the **cdrserver** shadow column wins the conflict.

Important: Do not remove the delete tables that are created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.

To use delete wins conflict resolution for repairing inconsistencies with the **cdr check replicate** or **cdr check replicateset** command, include the **--timestamp** and **--deletewins** options with the **--repair** option. Also set the CDR_DELAY_PURGE_DTC configuration parameter to the maximum age of modifications to rows that are being actively updated. If you temporarily stop replication on a server whose replicates use the delete wins conflict resolution rule, disable the replication server with the **cdr disable server** command. When you disable a server, information about deleted rows is kept in the delete tables to be used during the time stamp repair after the server is enabled.

Related concepts:
Replicating Simple Large Objects from Tblspaces
Time synchronization
Time stamp conflict resolution rule
Repair inconsistencies by time stamp
Related reference:
cdr disable server
CDR_DELAY_PURGE_DTC configuration parameter

Always-Apply Conflict-Resolution Rule

The always-apply conflict-resolution rule does not attempt to detect or resolve conflicts.

Unlike with the ignore conflict-resolution rule, replicated changes are applied even if the operations are not the same on the source and target servers. If a conflict occurs, the current row on the target is deleted and replaced with the replicated row from the source. Use the always-apply conflict-resolution rule only with a primary-target replication system. If you use always-apply with an update-anywhere replication system, your data might become inconsistent.

The following table describes how the always-apply conflict-resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 1. Always-Apply Conflict-Resolution Rule

Row exists in target?	INSERT	UPDATE	DELETE
No	Apply row	Apply row (convert UPDATE to INSERT)	Apply row (no error returned)
Yes	Apply as an UPDATE (overwrite the existing row)	Apply row	Deletes the row

Conflict Resolution Scope

Each conflict-resolution rule behaves differently depending on the scope.

Enterprise Replication uses the following scopes:

- Row scope
  When you choose a row scope, Enterprise Replication evaluates one row at a time. Only replicated rows that win the conflict resolution with the target rows are applied. If an entire replicated transaction receives row-by-row evaluation, some replicated rows are applied while other replicated rows might not be applied. Row scope can result in fewer failures than transaction scope.
- Transaction scope
  When you choose a transaction scope, Enterprise Replication applies the entire transaction if the replicated transaction wins the conflict resolution. If the target wins the conflict (or other database errors are present), the entire replicated transaction is not applied.

  A transaction scope for conflict resolution guarantees transactional integrity.

Enterprise Replication defers some constraint checking on the target tables until the transaction commits. If a unique constraint or foreign-key constraint violation is found on any row of the transaction at commit time, the entire transaction is rejected (regardless of the scope) and, if you have ATS set up, written to the ATS directory.

Transaction and row scopes are only applicable for apply failure related to conflict resolution, such as missing rows or newer local rows. For other errors, such as lock timeouts, constraint problems, lack of disk space, and so on, the whole incoming transaction is aborted, rolled back, and spooled to ATS or RIS files if so configured, regardless of whether row scope is defined.

Related concepts:
Failed Transaction (ATS and RIS) Files
Time stamp conflict resolution rule
Related tasks:
Specifying Conflict Resolution Rules and Scope
Related reference:
cdr define replicate

Choosing a Replication Network Topology

Enterprise replication *topology* describes connections that replication servers make to interact with each other. This topology is the route of replication data (message) transfer from server to server over the network. The replication topology is not synonymous with the physical network topology. Replication server definitions create the

replication topology, whereas replicate definitions determine data to be replicated and the sources and destinations within the topology.

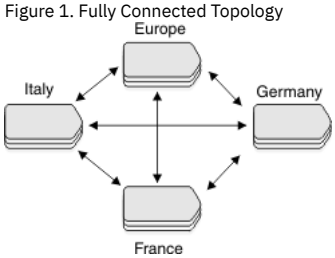
The topology that you choose influences the types of replication that you can use. These topics describe the topologies that Enterprise Replication supports.

- Fully Connected Topology
- Hierarchical Routing Topology Terminology
- Hierarchical Tree Topology
  A *hierarchical tree* consists of a root database server and one or more database servers organized into a tree topology.
- Forest of trees topology

Related concepts:
Flexible Architecture
Related tasks:
Defining Replication Servers
Customizing the Replication Server Definition

Fully Connected Topology

Fully connected replication topology indicates that all database servers connect to each other and that Enterprise Replication establishes and manages the connections. Replication messages are sent directly from one database server to another. No additional routing is necessary to deliver replication messages. Figure 1 shows a fully connected replication topology. Each database server connects directly to every other database server in the replication environment.
Figure 1. Fully Connected Topology



If necessary, you can also add high-availability clusters and a backup server to any server to provide high availability. For more information, see High-availability replication systems.

Related concepts:
Hierarchical Routing Topology Terminology
Hierarchical Tree Topology
Forest of trees topology

Hierarchical Routing Topology Terminology

Enterprise Replication uses the terms in the Table 1 to describe Hierarchical Routing topology.

Table 1. Replication Topology Terms

Term	Definition
Root server	An Enterprise Replication server that is the uppermost level in a hierarchically organized set of information The root is the point from which database servers branch into a logical sequence. All root database servers within Enterprise Replication must be fully interconnected.
Nonroot server	An Enterprise Replication server that is not a root database server but has a complete global catalog and is connected to its parent and to its children
Tree	A data structure that contains database servers that are linked in a hierarchical manner The topmost node is called the root. The root can have zero or more *child* database servers; the root is the *parent* database server to its children.
Parent-child	A relationship between database servers in a tree data structure in which the parent is one step closer to the root than the child.
Leaf server	A database server that has a limited catalog and no children.

A *root* server is fully connected to all other root servers. It has information about all other replication servers in its replication environment. Figure 1 shows an environment with four root servers.

A *nonroot server* is similar to a root server except that it forwards all replicated messages for other root servers (and their children) through its parent. All nonroot servers are known to all root and other nonroot servers. A nonroot server might or might not have children. All root and nonroot servers are aware of all other servers in the replication environment.
Important: In *Hierarchical Routing* topologies, Enterprise Replication specifies the synchronization server as the new server's parent in the current topology. For more information, see Customizing the Replication Server Definition and cdr define server.
Related concepts:
Fully Connected Topology
Hierarchical Tree Topology
Forest of trees topology
Creating sqlhost group entries for replication servers

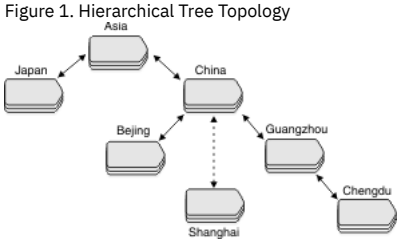
Related reference:
The syscdrs Table

Hierarchical Tree Topology

A *hierarchical tree* consists of a root database server and one or more database servers organized into a tree topology.

The tree contains only one root, which has no parent. Each database server within the tree references its parent. A database server that is not a parent is a leaf. Figure 1 illustrates a replication tree.

Figure 1. Hierarchical Tree Topology



In Figure 1, the parent-child relationship within the tree is as follows:

- **Asia** is the parent of **China** and **Japan**.
- **China** is the child of **Asia** *and* the parent of **Beijing**, **Shanghai**, and **Guangzhou**.
- **Guangzhou** is the child of **China** *and* the parent of **Chengdu**.

Asia is the root database server. **Japan**, **China**, and **Guangzhou** are nonroot database servers. You can define **Beijing**, **Shanghai**, and **Chengdu** as either nonroot database servers or leaf database servers, depending on how you plan to use them. The dashed connection from **China** to **Shanghai** indicates that **Shanghai** is a leaf server.

You can define a replicate that replicates data exclusively between **Shanghai** and **Japan**. However, the transaction data would must go through **China** and **Asia**. If either **China** or **Asia** is offline replication is suspended. Similarly, a replicate defined between **Japan** and **China** would require **Asia** to be functioning, even though both **Japan** and **China**, as nonroot servers, have entries in their sqlhosts files for each other.

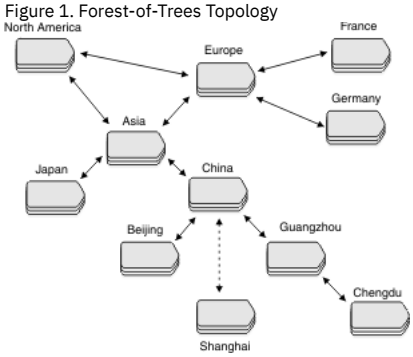
Parent servers are good candidates for using high-availability clusters to provide backup servers.

Related concepts:
Fully Connected Topology
Hierarchical Routing Topology Terminology
Forest of trees topology

Forest of trees topology

A *forest of trees* consists of several hierarchical trees whose root database servers are fully connected. Each hierarchical tree starts with a root database server. The root database servers transfer replication messages to the other root servers for delivery to its child database servers. Figure 1 shows a forest of trees.

Figure 1. Forest-of-Trees Topology



In Figure 1, **North America**, **Asia**, and **Europe** are root database servers. That is, they are fully connected with each other. **France** and **Germany** are in a tree whose root is **Europe**. **Asia** is the root for the six database servers in its tree.

In a forest of trees, all replication messages from one tree to another must pass through their roots. For example, a replication message from **Beijing** to **France** must pass through **China**, **Asia**, and **Europe**.

Organizing the database servers in a hierarchical tree or a forest of trees greatly reduces the number of physical connections that are required to make a replication system. If all the database servers in Figure 1 were fully connected, instead of being organized in trees, 55 connections would be required.

To ensure that all servers retain access to the replication system, use high-availability clusters on parent servers. For more information, see Using high-availability clusters in a forest of trees topology.

Related concepts:
Fully Connected Topology
Hierarchical Routing Topology Terminology

Hierarchical Tree Topology

Setting up and managing Enterprise Replication

After you design your replication system, you define it and start replication.

To set up replication:

1. Select the Enterprise Replication system and network topology to use for your replication environment.
2. Prepare the replication environment.
3. Define database servers for replication.
4. Define a grid and create replicated tables.

After you define and start your replication system, you can monitor and maintain it.

Instead of creating a grid, you can create a replicate set by defining and realizing a template, or you can define replicates and participants and then create a replicate set and start replication.

- Preparing the Replication Environment
  The following topics explain the steps that are required for setting up Enterprise Replication.
- Using High-Availability Clusters with Enterprise Replication
- Defining Replication Servers, Replicates, Participants, and Replicate Sets
  These topics describe the steps defining and starting Enterprise Replication.
- Grid setup and management
  A grid is a set of replication servers that are configured to simplify administration. When you run SQL data definition statements from within a grid context on a grid server, the statements propagate to all servers in the grid. You can run SQL data manipulation statements and routines through grid routines. You can choose to set up replication automatically when you create a table through a grid. You can propagate external files to other servers in the grid.
- Shard cluster setup
  Sharding is a way to horizontally partition a single table across multiple database servers in a shard cluster. Enterprise Replication moves the data from the source server to the appropriate target server as specified by the sharding method. You query a sharded table as if the entire table is on the local server. You do not need to know where the data is. Queries that are performed on one shard server retrieve the relevant data from other servers in a shard cluster. Sharding reduces the index size on each shard server and distributes performance across hardware. You can add shard servers to the shard cluster as your data grows.
- Managing Replication Servers and Replicates
- Monitor and troubleshooting Enterprise Replication
  You can monitor and diagnose problems with the Enterprise Replication system by using several different methods, depending on your needs.

Preparing the Replication Environment

The following topics explain the steps that are required for setting up Enterprise Replication.

The **cdr autoconfig serve** command can auto-configure Enterprise Replication for a database server that has a configured storage pool, and propagate connectivity information between the database servers in an Enterprise Replication domain. Complete the following steps to auto-configure Enterprise Replication:

1. Verify that the CDR_AUTO_DISCOVER configuration parameter is set to `1` on all database servers.
2. Verify that the storage pool is configured on any database server that you are adding to the Enterprise Replication Domain.
3. Choose a database server to be your source server for propagating configuration changes to other servers, and for replicating date to a newly added replication server.
4. On the source server, set trusted-host information for all database servers by running the **admin()** or **task()** function with the **cdr add trustedhost** argument.
5. Verify that all replication servers are active.
6. On the source server, run the **cdr autoconfig serve** command. Alternatively, you can run the **cdr autoconfig serve** command on a different database server, but you must specify the source server's information in the command.

- Preparing the Network Environment
  You must prepare the network environment for each database server in an Enterprise Replication domain.
- Preparing the Disk
  These topics describe how to prepare your disk for Enterprise Replication.
- Preparing the Database Server Environment
  To prepare the database server environment, set database server environment variables and configuration parameters, and synchronize the operating system time on all participating database servers.
- Preparing Data for Replication
- Load and unload data
  You can load data into or unload data out of tables in your replication environment in various ways, depending on your circumstances.
- Data Preparation Example

Related concepts:
Using High-Availability Clusters with Enterprise Replication
Grid setup and management
Shard cluster setup
Managing Replication Servers and Replicates
Monitor and troubleshooting Enterprise Replication
Related tasks:
Defining Replication Servers, Replicates, Participants, and Replicate Sets

Creating a new domain by cloning a server
Related information:
Trusted-host information
cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API)

Preparing the Network Environment

You must prepare the network environment for each database server in an Enterprise Replication domain.

The following files are involved in configuring the replication network:

- sqlhosts: Specifies replication connectivity, including server groups, connection security, and network security.
- hosts: Specifies hosts names if you are not using Domain Name Service (DNS).
- services: Specifies the service name that is associated with a port number.
- The trusted-hosts file. You specify this file by setting the REMOTE_SERVER_CFG configuration parameter. This file specifies the host names for trusted replication servers.
- If you use Connection Managers for managing connectivity, you must create a Connection Manager configuration file.

You can manually specify sqlhost and trusted-host file information to each database server, or you can run the **admin()** or **task()** function with the **cdr add trustedhost** argument to add entries to the trusted host files, and then run the **cdr autoconfig serv** command to propagate sqlhost and trusted-host file entries to other database servers in an Enterprise Replication domain.

To prepare your network environment, configure the following for each replication server:

1. If you are not using DNS, configure replication-server host information in the hosts file.
2. Configure port information in the services files and the sqlhosts files.
3. Create group entries for each replication servers in the sqlhosts file.
4. If necessary, configure secure ports for replication servers in the sqlhosts file.
5. If necessary, configure network security for client/server communications in the sqlhosts files.
6. Create a trusted-host file and add entries for each trusted hosts.

- Configuring hosts information for replication servers
  If you are not using Domain Name Service (DNS) to identify IP addresses and system names, you do need to configure the hosts file on each replication server to add the IP addresses and system names for all other replication servers in the domain.
- Configuring ports and service names for replication servers
  Replication servers must know the port numbers for each of the other replication servers in the domain.
- Creating sqlhost group entries for replication servers
  The sqlhosts file on the host of each replication server must specify a group entry for each replication server in an Enterprise Replication domain. You can manually specify sqlhost file information, or run the **cdr autoconfig serv** command to add entries to a database server's sqlhost file, and then propagate the entries to other database servers in an Enterprise Replication domain. However, if you are configuring secure ports, you cannot use the **cdr autoconfig serv** command.
- Configuring secure ports for connections between replication servers
  If database servers in your Enterprise Replication environment are on a network that is not trusted, you can configure secure ports and an encrypted password file to enable secure connections.
- Configuring network encryption for replication servers
  You encrypt client/server network communication by specifying the ENCCSM module with the communications support module (CSM) option in the sqlhosts file. You encrypt Enterprise Replication communication by setting encryption configuration parameter ENCRYPT_SMX or by configuring ER group option for onsocssl port in sqlhosts file. Unless onsocssl port is used for communicating with peer ER servers, for communicating with older server versions before 14.10xC6, Enterprise Replication requires configuring ENCRYPT_CDR instead of ENCRYPT_SMX.
- Testing the replication network
  After you set up the network environment, test the connections between the replication servers. You cannot test a connection that uses the **s=6** option in the sqlhosts file.
- Testing the password file
  You create and encrypt a password file to allow the CDR utility to access to a secure network environment. Use these steps to test that the encrypted password file is correctly configured.

Related concepts:
Replication Examples
Related reference:
cdr autoconfig serv
CDR_AUTO_DISCOVER configuration parameter
Related information:
The syncsqlhosts utility
Client/server communication
Trusted-host information
cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API)

Configuring hosts information for replication servers

If you are not using Domain Name Service (DNS) to identify IP addresses and system names, you do need to configure the hosts file on each replication server to add the IP addresses and system names for all other replication servers in the domain.

The hosts file is in the following location.

Operating System	File
UNIX	/etc/hosts
Windows	%WINDIR%\system32\drivers\etc\hosts

Important: Leave a blank line at the end of the hosts file on Windows.
For example, your hosts file might look like the following:

```
192.168.0.1 ny.usa.com
192.168.0.2 tokyo.japan.com
192.168.0.3 rome.italy.com
192.168.0.4 perth.australia.com
```


Configuring ports and service names for replication servers

Replication servers must know the port numbers for each of the other replication servers in the domain.

Configure port numbers for replication servers in one of the following ways:

- Specify the port numbers in the sqlhosts file. This method risks conflicting with port numbers being used by other applications.
- Specify the service names in the sqlhosts file and specify the port numbers for each service name in the services file.

The services file is in the following location.

Operating System	File
UNIX	/etc/services
Windows	%WINDIR%\system32\drivers\etc\services

Important: Leave a blank line at the end of the services file on Windows.
For example, your services file might look like the following:

```
sydney 5327/tcp
melbourne 5327/tcp
```

If the database servers reside on the same system, you must provide unique port numbers for each.

Related reference:
cdr start sec2er

Creating sqlhost group entries for replication servers

The sqlhosts file on the host of each replication server must specify a group entry for each replication server in an Enterprise Replication domain. You can manually specify sqlhost file information, or run the **cdr autoconfig serv** command to add entries to a database server's sqlhost file, and then propagate the entries to other database servers in an Enterprise Replication domain. However, if you are configuring secure ports, you cannot use the **cdr autoconfig serv** command.

Typically, a server group includes only one database server. However, if the computer has multiple network protocols or network interface cards, the server group includes all aliases for the database server. Enterprise Replication treats the server group as one object, whether it includes one or several database server names.
The following example shows sqlhosts file entries for four Enterprise Replication servers:

- **serv1**
- **serv2**
- **serv3**
- **serv4**

```
#dbservername  nettype   hostname              servicename  options
 g_serv1       group     -                     -            i=143
 serv1         ontlitcp  ny.usa.com            1230         g=g_serv1

 g_serv2       group     -                     -            i=144
 serv2         ontlitcp  tokyo.japan.com       1231         g=g_serv2

 g_serv3       group     -                     -            i=145
 serv3         ontlitcp  rome.italy.com        1232         g=g_serv3

 g_serv4       group     -                     -            i=146
 serv4         ontlitcp  perth.australia.com   1233         g=g_serv4
```

Each server has two entries of information:

- A group definition, which specifies a group name and unique ID for the replication server
- Connectivity information for the database server

All Enterprise Replication commands and options use the name of the database server group or the more familiar database server name (that is, the name that is specified by the INFORMIXSERVER environment variable) for all references to database servers. The exception is the **--connect** option, which can use either a server name or a group name.

Leaf servers in hierarchical routing topologies do not require connectivity information for all replication servers. A leaf server requires connectivity information for only itself and its parent.

Related concepts:
Connect Option
Hierarchical Routing Topology Terminology
Setting Up Database Server Groups for High-Availability Cluster Servers
Related reference:
cdr autoconfig serv
Related information:
The sqlhosts information
sqlhosts connectivity information
The syncsqlhosts utility

Configuring secure ports for connections between replication servers

If database servers in your Enterprise Replication environment are on a network that is not trusted, you can configure secure ports and an encrypted password file to enable secure connections.

The secure ports that are listed in the sqlhosts files can be used only for communication between database servers. You must configure a separate port for local client/server communications.
To configure a secure port for replication:

1. In the sqlhosts file on each server, create a group entry with two connections for the local server:
   a. Create one connection entry without the **s=6** option to configure local communication with utilities, such as the **cdr** utility and Connection Managers.
   b. Create one connection entry with the **s=6** option to configure communication between servers.
   In the following example, the value of the DBSERVERNAME configuration parameter is **serv1**:

   ```
   #dbservername    nettype         hostname       servicename     options
   serv1            ontlitcp        ny.usa.com     ertest1
   g_serv1          group           -              -               i=143
   serv1_s6         ontlitcp        ny.usa.com     ertest10        g=g_serv1,s=6
   ```

   Note: Do not use the **cdr autoconfig serv** command if you configure secure ports. sqlhosts file entries must be manually added if any entries include the **s=6** option.
2. In the sqlhosts file on each server's host, add entries for each of the other servers in the domain. Use the server names that are associated with the **s=6** options.
3. Create a trusted-host file that includes the host names of the other replication servers in the domain, each on a separate line. You can manually create the trusted-host file in $INFORMIXDIR/etc, and then set the REMOTE_SERVER_CFG configuration parameter to the name of the trusted-host file. Alternatively, you can run the **admin()** or **task()** function with the **cdr add trustedhost** argument to set a replication server's REMOTE_SERVER_CFG configuration parameter and add entries to the server's trusted-host file. If the replication server is part of a high-availability cluster, running the **admin()** or **task()** function with the **cdr add trustedhost** argument propagates trusted-host entries to other database servers in a high-availability cluster.
   Note: You cannot use the hosts.equiv trusted-host file when you configure secure ports.
   The following example trusted-host file has entries for three hosts, and specifies both host names and domain names:

   ```
   #hostname
   tokyo.japan.com
   tokyo

   rome.italy.com
   rome

   perth.australia.com
   perth
   ```

   A database server on a listed host connects to the local database server instance through the sqlhosts file entry with the s=6 option.
4. Set the S6_USE_REMOTE_SERVER_CFG configuration parameter to `1` in the onconfig file.
5. Using a text editor, create and save a password file. The password file includes the host name, alternative server name, user ID, and password for each server and the server group. For example, if the user ID for server **serv1** is `informix`, the alias for the database server that uses a secure port is **serv1_s6**, and the password was `informix_pw`, use the following password file entries:

   ```
   serv1_s6  serv1  informix  informix_pw
   g_serv1   serv1  informix  informix_pw
   ```

6. Encrypt the password file by running the **onpassword** utility. For example, if you named the text file in step 5 $INFORMIXDIR/etc/server_passwords, and you wanted the file encrypted with a key called **access_key**, use the following command:

   ```
   onpassword -k access_key -e $INFORMIXDIR/etc/server_passwords
   ```

   The encrypted file is saved as: `$INFORMIXDIR/etc/passwd_file`.

   Important: To prevent unauthorized access to the server passwords, remove the unencrypted password file, $INFORMIXDIR/etc/server_passwords after you create the encrypted file.

If you do not configure a password file, you must run the **cdr** utility on the local computer, for example:

```
cdr list server --connect=serv1
```

Because secure ports can be used only for replication communication, you cannot test the connections until you start replication.

Related tasks:
Testing the replication network
Configuring secure connections for grid queries
Related information:
S6_USE_REMOTE_SERVER_CFG configuration parameter
The onpassword utility
The sqlhosts file and the SQLHOSTS registry key

DBSERVERALIASES configuration parameter
REMOTE_SERVER_CFG configuration parameter

Configuring network encryption for replication servers

You encrypt client/server network communication by specifying the ENCCSM module with the communications support module (CSM) option in the sqlhosts file. You encrypt Enterprise Replication communication by setting encryption configuration parameter ENCRYPT_SMX or by configuring ER group option for onsocssl port in sqlhosts file. Unless onsocssl port is used for communicating with peer ER servers, for communicating with older server versions before 14.10xC6, Enterprise Replication requires configuring ENCRYPT_CDR instead of ENCRYPT_SMX.

You cannot configure an Enterprise Replication connection with a CSM.

To combine client/server network encryption with Enterprise Replication encryption, configure two network connections for each database server. The configuration in the SQLHOSTS file would look like the following example.

```
#dbservername  nettype   hostname     servicename  options
 gserv1        group     -            -            i=143
 serv1         ontlitcp  ny.usa.com   ertest1      g=gserv1
 c_serv1       ontlitcp  ny.usa.com   ertest10     csm=(ENCCSM)
```

In this example, **serv1** and **c_serv1** are two connection ports on the same database server. Encrypted client/server communication uses the **c_serv1** port, while encrypted Enterprise Replication uses the **serv1** port.

For more information on encrypting client/server network communications, see the *IBM® Informix® Administrator's Guide*.

Related reference:
Set configuration parameters for replication
Enterprise Replication configuration parameter and environment variable reference

Testing the replication network

After you set up the network environment, test the connections between the replication servers. You cannot test a connection that uses the **s=6** option in the sqlhosts file.

To test the network environment:

1. Verify the network connection. Use the **ping** command to test the connection between two systems. For example, from **ny.usa.com**, test the connection to **tokyo.japan.com**:

   ```
   ping tokyo.japan.com
   ```

2. Test the trusted environment:
   a. Run **dbaccess** as user **informix** or as the owner if it is a non-root server.
   b. Select the **Connection** menu option.
   c. Select the **Connect** menu option.
   d. Connect to the server group name and the server name of the other hosts.
      For example, if you are running **dbaccess** on **ny.usa.com**, and you are testing the connection to a database server on **tokyo.japan.com**, select **serv2** and **g_serv2**.
   e. When prompted for the USER NAME, press `Enter`.

   If you can connect to the host database server, the host server trusts the connection from the remote host as user **informix** or as the owner if the remote host is a non-root server.

   For more information, see the *IBM® Informix® DB-Access User's Guide*.

Related tasks:
Configuring secure ports for connections between replication servers

Testing the password file

You create and encrypt a password file to allow the CDR utility to access to a secure network environment. Use these steps to test that the encrypted password file is correctly configured.

To test the password file configuration:

Use the **cdr view state -c *remote_server_group_name*** command to verify that the password file supplies the correct password to the CDR command. For example, if your remote server group was named g_serv2, specify the following command:

```
cdr view state -c g_serv2
```

The state of all configured enterprise replication servers is returned. If enterprise replication is not defined, but the password file is set up correctly, the following message is returned:

```
ERROR:ER not defined on g_serv2
```

If the CDR utility is unable to connect to the server or if the following error is returned then verify that $INFORMIXDIR/etc/passwd_file is correctly configured.

```
25539: Invalid connection-type
```

The following is an example of command output returned when Enterprise Replication and the password file are correctly configured:

```
$  cdr view state -c g_serv2
STATE
Source   ER              Capture          Network          Apply
         State           State            State            State
------------------------------------------------------------------------
g_serv2 Active          Running          Running          Running
g_serv1 Active          Running          Running          Running
```


Preparing the Disk

These topics describe how to prepare your disk for Enterprise Replication.

- Logical Log Configuration Disk Space
- Logical Log Configuration Guidelines
  Logical logs must be configured correctly for Enterprise Replication.
- Disk Space for Delete Tables
  If you use the time stamp, time stamp and SPL routine, or delete wins conflict resolution rules, you must provide enough disk space for the *delete tables* that Enterprise Replication creates to keep track of modified rows for conflict resolution.
- Shadow column disk space
  If you plan to use shadow columns, make sure to allow additional disk space for their values.
- Setting Up Send and Receive Queue Spool Areas
- Setting Up the Grouper Paging File
- Creating ATS and RIS directories
  You can create directories for Aborted Transactions Spooling (ATS) and Row Information Spooling (RIS) files instead of using the default directories.

Logical Log Configuration Disk Space

The database server uses the logical log to store a record of changes to the data since the last archive. Enterprise Replication requires the logical log to contain entire row images for updated rows, including deleted rows.

The database server normally logs only columns that have changed. This behavior is called the logical-log record reduction option. Enterprise Replication deactivates this option for tables that participate in replication. (The logical-log record reduction option remains enabled for tables that do *not* participate in Enterprise Replication.) Enterprise Replication logs all columns, not only the columns that have changed, which increases the size of your logical log.

To determine the size of your logical log, examine your data activity for normal operations and for the replication system you defined. Keep in mind that defining replication on a table causes Enterprise Replication to deactivate log reduction for that table, and that your transactions might log more data.
Important: Enterprise Replication performs internal cleanup tasks based on how often the log files switch. If the log files switch too frequently, Enterprise Replication might perform excessive cleanup work.

Logical Log Configuration Guidelines

Logical logs must be configured correctly for Enterprise Replication.

Use the following guidelines when configuring your logical log files:

- Make sure that all logical log files are approximately the same size.
- Make the size of the logical log files large enough so that the database server switches log files no more than once every 15 minutes during normal processing.
- Plan to have sufficient logical-log space to hold at least four times the maximum transaction size.
- Set LTXEHWM (long-transaction, exclusive-access, high-watermark) 30 percent larger than LTXHWM (long-transaction high-watermark).

Important: If you specify that the database server allocate logical log files dynamically (DYNAMIC_LOGS), it is recommended that you set LTXEHWM to no higher than 70 when using Enterprise Replication.
For more information about logical logs and these configuration parameters, see *IBM® Informix® Administrator's Reference* and *IBM Informix Administrator's Guide*.

The database server can add logs dynamically when Enterprise Replication approaches a potential log wrap situation if the CDR_MAX_DYNAMIC_LOGS configuration parameter is set to a non-zero integer.

Related concepts:
Handle potential log wrapping
Related tasks:
Preventing Memory Queues from Overflowing

Disk Space for Delete Tables

If you use the time stamp, time stamp and SPL routine, or delete wins conflict resolution rules, you must provide enough disk space for the *delete tables* that Enterprise Replication creates to keep track of modified rows for conflict resolution.

Delete tables handle conflicts such as when a DELETE or UPDATE operation finds no corresponding row on the target. The DTCleaner thread removes a row from the delete tables after all the servers have progressed beyond that row. Enterprise Replication does not create delete tables for tables that have replicates defined with a conflict resolution rule of ignore or always-apply.

Delete tables are created on the database server where the data originates and on all the database servers to which data gets replicated. Delete tables are stored in the same dbspaces, using the same fragmentation strategy, as their base tables.

To determine the disk space requirements to accommodate delete tables, estimate how many rows will be deleted or modified. For example, if the base table has 100 megabytes of data, but only half the rows might be deleted or modified, then 50 megabytes is a reasonable estimate for the size of the delete table.
Important: Do not remove the delete tables created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.
Related concepts:
Update-Anywhere Replication System
Related reference:
Replicate only changed columns

Shadow column disk space

If you plan to use shadow columns, make sure to allow additional disk space for their values.

If you plan to use any conflict-resolution rule except ignore or always-apply, you must allow for an additional 8 bytes for the CRCOLS shadow columns, **cdrserver** and **cdrtime**, which store the server and time stamp information that Enterprise Replication uses for conflict resolution.

If you want to speed consistency checking by indexing the REPLCHECK shadow column, you must allow for an additional 8 bytes for the **ifx_replcheck** shadow column.

If you want to use ERKEY shadow columns as the replication key, or you create your replicated tables through a grid, you must allow of an additional 10 bytes for the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**. When you create replicated tables through a grid, these ERKEY columns are automatically added.ERKEY columns also require disk space for the index that is created on them. In addition to the standard partition and page overhead, for each row in the table the ERKEY index uses 14 bytes for non-fragmented tables and 18 bytes for fragmented tables for each row in the table.

The following table shows the amount of space used by each shadow column.

Table 1. Shadow column size

Shadow column name	Data type	Size
cdrserver	INTEGER	4 bytes
cdrtime	INTEGER	4 bytes
ifx_replcheck	BIGINT	8 bytes
ifx_erkey_1	INTEGER	4 bytes
ifx_erkey_2	INTEGER	4 bytes
ifx_erkey_3	SMALLINT	2 bytes

The shadow columns claim disk space immediately, except when CRCOLS and ERKEY columns are added to an existing table.

Related concepts:
Update-Anywhere Replication System
Shadow columns
Preparing Tables for Conflict Resolution
Preparing Tables for a Consistency Check Index

Setting Up Send and Receive Queue Spool Areas

The term *data queue* refers to both the *send queue* and the *receive queue*. Enterprise Replication collects information from the logical logs and places the data to be transferred in the send queue. Then Enterprise Replication transfers the contents of the send queue to the receive queue on the target server. Enterprise Replication on the target reads the data from the receive queue and applies the changes to the tables on the target server.

The send and receive queues reside in memory and are managed by the Reliable Queue Manager (RQM). The CDR_QUEUEMEM configuration parameter (CDR_QUEUEMEM Configuration Parameter) specifies the amount of memory space that is available for the data queues.

When a queue in memory fills (for the receive queue, this only occurs with large transactions), the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of *transaction records* (headers that contain internal information for Enterprise Replication), *replicate information* (summaries of the replication information for

each transaction), and *row data* (the actual replicated data). Spooled transaction records and replication records are stored in transaction tables and replication tables in a single dbspace. Spooled row data is stored in one or more sbspaces.
Important: To prevent the send and receive queues from spooling to disk, see Preventing Memory Queues from Overflowing.

- Row Data sbspaces
  Replicated data might include UDT and CLOB or BLOB data types. Therefore, the spooled row data is stored as smart large objects in one or more sbspaces.

Related concepts:
Send queues and receive queues
Related tasks:
Preventing Memory Queues from Overflowing

Row Data sbspaces

Replicated data might include UDT and CLOB or BLOB data types. Therefore, the spooled row data is stored as smart large objects in one or more sbspaces.

The CDR_QDATA_SBSPACE configuration parameter accepts multiple sbspaces, up to a maximum of 32 sbspaces. Enterprise Replication can support a combination of logging and non-logging sbspaces for storing spooled row data. If CDR_QDATA_SBSPACE is configured for multiple sbspaces, then Enterprise Replication uses all appropriate sbspaces in round-robin order.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the CDR_QDATA_SBSPACE configuration parameter when defining a replication server. If the CDR_QDATA_SBSPACE configuration parameter is not set and the database server has a storage pool with sufficient space, the **cdr define** command performs the following tasks:

- Creates a new sbspace using one or more new chunks from the storage pool
- Sets the CDR_QDATA_SBSPACE configuration parameter both in memory and in the onconfig file to the newly defined sbspace.

For clusters, the **cdr define** command creates new sbspaces and sets the CDR_QDATA_SBSPACE configuration parameters in all secondary database servers, as well.
Note: A database server's storage pool must have 500 MB of free space for sbspaces, and chunk sizes of 100 MB or greater for the database server to use automatic storage provisioning.

- Creating sbspaces for Spooled Row Data
  You must create dedicated sbspaces for spooled row data.
- Logging Mode for sbspaces
- Dropping a Spooled Row Data sbspace

Related tasks:
Defining Replication Servers
Related reference:
Set configuration parameters for replication

Creating sbspaces for Spooled Row Data

You must create dedicated sbspaces for spooled row data.

Follow these guidelines when creating sbspaces for spooled row data:

- Create all the sbspaces of same default log mode type with the same size.
- Do not use Enterprise Replication row data sbspaces for non-Enterprise Replication activity.
- Ensure that the sbspaces are sufficiently large.
  To determine the size of your spooled row data sbspaces, determine your log usage and then consider how much data you can collect if your network goes down. For example, assume that you usually log 40 megabytes of data each day, but only 10 percent of that is replicated data. If your network is down for 24 hours and you estimate that 4 MB of replicated data are logged each day, the size of the sbspaces you identify for the spooled row data must be a total of at least 4 MB.

  Windows Only
  On Windows, increase the resulting size of the sbspace by approximately a factor of two. (The default page size, the way that data maps onto a page, and the number of pages written to disk differs on Windows.)

Important: When the row data sbspaces fill, Enterprise Replication hangs until you either resolve the problem that is causing Enterprise Replication to spool or allocate additional disk space to the sbspaces. For more information, see Preventing Memory Queues from Overflowing.
To create row data sbspaces, use the **onspaces -c** utility. For example, to create a 4-megabyte sbspace, called **er_sbspace**, using raw disk space on UNIX with an offset of 0, enter:

```
onspaces -c -S er_sbspace -p /dev/rdsk/c0t1d0s4 -o 0 -s 4000\
    -m /dev/rdsk2/c0t1d0s4 0 \
    -Df "AVG_LO_SIZE=2,LOGGING=OFF"
```

The path name for an sbspace cannot be longer than 256 bytes.

The **-m** option specifies the location and offset of the sbspace mirror. The **-Df** option specifies default behavior of the smart large objects stored in the sbspace:

- AVG_LO_SIZE (average large object size)
  Set this parameter to the expected average transaction size (in KB). The database server uses this value to calculate the metadata size. The minimum value for AVG_LO_SIZE is 2 KB, which is appropriate for Enterprise Replication in most cases. (The default value of AVG_LO_SIZE is 32 KB.) If you set AVG_LO_SIZE to larger

than the expected transaction size, you might run out of metadata space. If you set AVG_LO_SIZE too small, you might waste space on metadata.

- LOGGING
  Set this parameter to OFF to create an sbspace without logging. Set this parameter to ON to create an sbspace with logging. Use a combination of logging and non-logging sbspaces for Enterprise Replication. For more information, see Logging Mode for sbspaces.

Set the CDR_QDATA_SBSPACE configuration parameter in the ONCONFIG file to the location of the row data sbspace (**er_sbspace**, in this example). For more information, see CDR_QDATA_SBSPACE Configuration Parameter.

Logging Mode for sbspaces

Enterprise Replication uses the default log mode that the sbspace was created with for spooling row data.

Create sbspaces with a default logging mode of ON or OFF according to the types of transactions Enterprise Replication replicates:

- **LOGGING=ON**
  Create sbspaces with LOGGING set to ON to support these situations:
  - Replicated systems with high-availability clusters
    Enterprise Replication must use logging sbspaces for transactions involved in high-availability clusters.
  - Small transactions
    Enterprise Replication uses logging sbspaces for transactions that are less than a page size (2K or 4K) of replicated data.

  For logging sbspaces, performance might be enhanced because logging mode enables asynchronous IO. However, a logging sbspace consumes additional logical-log space.
- **LOGGING=OFF**
  Create sbspaces with LOGGING set to OFF to support replication of large transactions (greater than a page size of replicated data).

  It is recommended that you mirror non-logging sbspaces. For more information, see the chapter on managing disk space in the *IBM® Informix® Administrator's Guide* and the *IBM Informix Administrator's Reference*.

  For non-logging sbspaces, performance is enhanced on the database server when Enterprise Replication spools to disk because Enterprise Replication writes less data to disk.

Important: Do not change the Enterprise Replication sbspace default log mode while Enterprise Replication is running. To change the default log mode, follow the procedure below.
You can change the default logging mode of the row data sbspace if you have more than one sbspace specified by the CDR_QDATA_SBSPACE configuration parameter.

To change the default logging mode of a row data sbspace:

1. Shut down the database server.
2. Remove the sbspace from the CDR_QDATA_SBSPACE configuration parameter value list.
3. Start the database server in recovery mode.
4. Wait for all the smart large objects to get deleted from the sbspace. Use the **onstat -g smb lod** command to check for smart large objects stored in an sbspace.
5. Change the default logging mode for the sbspace.
6. Add the sbspace name to the CDR_QDATA_SBSPACE configuration parameter value list.
7. Shut down and restart the database server using the **onmode -ky** and **oninit** commands.

Dropping a Spooled Row Data sbspace

Important: Do not drop an Enterprise Replication row data sbspace using the **onspaces -d -f** (force) command.
You can drop a row data sbspace if you have more than one sbspace specified by the CDR_QDATA_SBSPACE configuration parameter.

To drop a row data sbspace

1. Shutdown the database server.
2. Remove the sbspace from the CDR_QDATA_SBSPACE configuration parameter value list.
3. Start the database server in recovery mode.
4. Wait for all the smart large objects to get deleted from the sbspace. Use the **onstat -g smb lod** command to check for smart large objects stored in a sbspace.
5. If the sbspace was added from the storage pool, use the **drop sbspace to storagepool** argument with the **admin()** or **task()** function to return the empty sbspace to the storage pool.

Related information:
drop sbspace to storagepool argument: Return space from an empty sbspace to the storage pool (SQL administration API)

Setting Up the Grouper Paging File

Enterprise Replication uses a grouper paging mechanism for evaluating large transactions. A transaction is large if the portion to be replicated meets at least one of the following conditions:

- It has greater than 5,000 log records.
- It exceeds one fifth the size of the value of the CDR_QUEUEMEM ONCONFIG variable.
- It exceeds one tenth the size of the value of the SHMVIRTSIZE configuration variable.

The location of the sbspace used for the paging file is determined by the first of the following ONCONFIG configuration parameters that is set:

- SBSPACETEMP
- SBSPACENAME
- CDR_QDATA_SBSPACE

The best solution is to set up an unlogged sbspace, as specified by the SBSPACETEMP configuration parameter. All updates to the paging files are unlogged.

The size of the paging sbspace should be at least three times the size of the largest transaction to be processed. This sbspace is also used by the database server for other tasks; consider its overall usage when determining size requirements.
Important: If the paging sbspace fills, Enterprise Replication hangs until you allocate additional disk space to the sbspace. If additional space is unavailable, use the **cdr stop** command to stop replication.

Creating ATS and RIS directories

You can create directories for Aborted Transactions Spooling (ATS) and Row Information Spooling (RIS) files instead of using the default directories.

ATS and RIS files contain information about failed transactions and aborted rows. You can repair data after a replicated transaction fails by applying ATS and RIS files. Enterprise Replication examines the specified ATS or RIS file and attempts to reconcile the rows that failed to be applied. ATS and RIS files are relevant only if you specify a conflict resolution role other than ignore or always-apply.

The default location for ATS and RIS directories is /tmp (UNIX) or \tmp (Windows).

If you want to use non-default directories, create the ATS or RIS directories before you define the server for replication. The path names for the ATS and RIS directories cannot be longer than 256 characters.

- Create RIS directories on all replication servers in the domain.
- Create ATS directories on all replication servers in the domain, if you are using update-anywhere replication.
- Create the ATS directory on the target system, if you are using primary-target replication.

Related concepts:
Failed Transaction (ATS and RIS) Files
Related tasks:
Enabling ATS and RIS File Generation
Customizing the Replication Server Definition
Setting Up Failed Transaction Logging

Preparing the Database Server Environment

To prepare the database server environment, set database server environment variables and configuration parameters, and synchronize the operating system time on all participating database servers.

If you are using high-availability clusters with Enterprise Replication, set up your servers according to the instructions in Setting Up Database Server Groups for High-Availability Cluster Servers.

- Setting Database Server Environment Variables
  Certain environment variables must be set in a replication environment.
- Set configuration parameters for replication
  You must set certain configuration parameters before you start Enterprise Replication. You can set other configuration parameters to customize the behavior of Enterprise Replication.
- Time synchronization
  Whenever you use replication that requires time stamp, time stamp with a stored procedure, or delete wins conflict resolution, you must synchronize the operating system times of the database servers that participate in the replicate.

Setting Database Server Environment Variables

Certain environment variables must be set in a replication environment.

To configure the database server environment, verify that the following environment variables are set correctly:

- INFORMIXDIR is set to the full path of the IBM® Informix® directory.
- INFORMIXSERVER is set to the name of the default database server.

- INFORMIXSQLHOSTS is set to the full path to the SQLHOSTS file.
- DELIMIDENT is not set or set to `n`. Enterprise Replication does not allow delimited identifiers.

Set configuration parameters for replication

You must set certain configuration parameters before you start Enterprise Replication. You can set other configuration parameters to customize the behavior of Enterprise Replication.

Parameters to set before you start replication

Set the following configuration parameters in the onconfig file on each database server that you want to include in the replication domain before you start replication:

- DBSERVERNAME specifies the name of the database server. If you use both the DBSERVERNAME and DBSERVERALIASES configuration parameters, set the DBSERVERNAME configuration parameter to the TCP connection and not to a shared-memory connection.
- CDR_QUEUEMEM specifies the maximum amount of memory to be used for the send and receive queues.
- CDR_SERIAL specifies how to generate non-overlapping (unique) values for serial columns across all database servers in the replication domain.
- CDR_TSINSTANCEID specifies how to generate unique identifiers for time series instances across all database servers in the replication domain.

Logging parameters

By default, if Enterprise Replication detects the potential for a log wrap situation when replication log processing lags behind the current log position, user transactions are blocked. You can configure Enterprise Replication to prevent the blocking of user transactions. Depending on the solutions you need, you might set the following configuration parameters in the onconfig file for each database server:

- CDR_LOG_LAG_ACTION specifies the actions that Enterprise Replication during a potential log wrap situation.
- LOG_STAGING_DIR specifies a directory in which compressed log files are staged.
- CDR_LOG_STAGING_MAXSIZE specifies the maximum size that Enterprise Replication can use to stage log files.
- CDR_MAX_DYNAMIC_LOGS specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.
- DYNAMIC_LOGS specifies that logical logs can be added dynamically.

Encryption parameters

If you want to encrypt network communications, set the following configuration parameters in the onconfig file for each database server:

- ENCRYPT_CDR specifies whether to enable encryption. The default value is `0`, which prevents encryption.
- ENCRYPT_CIPHERS specifies which ciphers and cipher modes are used for encryption.
- ENCRYPT_MAC controls the level of Message Authentication Code (MAC) used to ensure message integrity.
- ENCRYPT_MACFILE specifies the full path and file names of the MAC files.
- ENCRYPT_SWITCH specifies the number of minutes between automatic renegotiations of ciphers and keys. (The cipher is the encryption methodology. The secret key is the key that is used to build the encrypted data using the cipher.)

Other parameters

Set the following optional configuration parameters to customize your replication environment:

- CDR_DSLOCKWAIT specifies the number of seconds the data sync component waits for the database locks to be released. When replication is active on an instance, you can increase the amount of time to wait for lock resources to accommodate transactions on replicated tables.
- CDR_SUPPRESS_ATSRISWARN suppresses certain data sync error and warning codes from appearing in ATS and RIS files.
- CDR_DELAY_PURGE_DTC specifies how long to retain rows in delete tables to support the delete wins conflict resolution rule.
- GRIDCOPY_DIR specifies the default directory that is used by the **ifx_grid_copy** procedure.
- CDR_MAX_FLUSH_SIZE specifies the number of replicated transactions that are applied before the logs are flushed to disk.

Related concepts:
Row Data sbspaces
Serial data types and replication keys
Configuring network encryption for replication servers
Related tasks:
Managing Replication Servers
Adding a server to the domain by cloning a server
Related reference:
Enterprise Replication configuration parameter and environment variable reference
Related information:
DBSERVERNAME configuration parameter
DBSERVERALIASES configuration parameter

Time synchronization

Whenever you use replication that requires time stamp, time stamp with a stored procedure, or delete wins conflict resolution, you must synchronize the operating system times of the database servers that participate in the replicate.

All timestamps and internal computations are performed in Greenwich Mean Time (GMT) and have an accuracy of plus or minus one second.

Important: Enterprise Replication does not manage clock synchronization between database servers that participate in a replicate. You should use a product that supplies a network time protocol to ensure that times remain synchronized. For information on tools for synchronizing the times, refer to your operating system documentation. To synchronize the time on one database server with the time on another database server, use one of the following commands, where *hostname* or *servername* is the name of the remote database server computer.

UNIX

```
rdate hostname
```

Windows

```
net time \\servername /set
net time /domain:servername /set
```

Important: These commands do not guarantee the times will remain synchronized. If the operating system times of the database servers do become out of sync or if their times move backward, time stamp or stored procedure conflict resolution might produce failures caused by incorrect time stamps.

Related concepts:
Conflict Resolution
Delete wins conflict resolution rule
Time stamp conflict resolution rule

Related tasks:
Adding a server to the domain by cloning a server

Preparing Data for Replication

The goal of data replication is to provide identical, or at least consistent, data on multiple database servers. This section describes how to prepare the information in your databases for replication.

When you define a new replicate on tables with existing data on different database servers, the data might not be consistent. Similarly, if you add a participant to an existing replicate, you must ensure that all the databases in the replicate have consistent values.

For more information, see Data Preparation Example.

- Preparing Consistent Data
- Blocking Replication
  You might need to block replication so that you can put data into a database that you do not want replicated, perhaps for a new server or because you had to drop and re-create a table.
- Preparing to Replicate User-Defined Types
- Preparing to Replicate User-Defined Routines
- Preparing Tables for Conflict Resolution
  To use any conflict-resolution rule other than ignore or always-apply, you must define the shadow columns, **cdrserver** and **cdrtime** in the tables on both the source and target servers involved in replication.
- Preparing Tables for a Consistency Check Index
  To improve the speed of consistency checking with an index, you must define the **ifx_replcheck** shadow column in the tables on both the source and target servers involved in replication.
- Preparing tables without primary keys
  The data columns in your table might not need a primary key. To replicate tables that do not have primary keys, you can specify a unique index or add the ERKEY shadow columns.
- Preparing Logging Databases
- Preparing for Role Separation (UNIX)
  You can use role separation to allow members of the DBSA group to run Enterprise Replication commands, in addition to the user **informix**. For some Enterprise Replication commands, you must grant the DBSA user additional permissions on tables or files. For non-root servers, role separation is not supported. Only the owner of a non-root server is allowed to run the Enterprise Replication commands that require additional permissions for a DBSA.

Related concepts:
Update-Anywhere Replication System

Preparing Consistent Data

In most cases, preparing consistent data simply requires that you decide which of your databases has the most accurate data and then that you copy that data onto the target database. If the target database already has data, for data consistency, you must remove that data before adding the copied data. For information on loading the data, see Load and unload data.

Blocking Replication

You might need to block replication so that you can put data into a database that you do not want replicated, perhaps for a new server or because you had to drop and re-create a table.

To block replication while you prepare a table, use the BEGIN WORK WITHOUT REPLICATION statement. This starts a transaction that does not replicate to other database servers.

The following code fragment shows how you might use this statement:

```
BEGIN WORK WITHOUT REPLICATION
LOCK TABLE office
DELETE FROM office WHERE description = 'portlandR_D'
COMMIT WORK
```

- Using DB-Access to Begin Work Without Replication
- Using ESQL/C to Begin Work Without Replication

Related concepts:
Load and unload data

Using DB-Access to Begin Work Without Replication

The following example shows how to use DB-Access to begin work without replication:

```
DATABASE adatabase;
BEGIN WORK WITHOUT REPLICATION
insert into mytable (col1, col2, ....)
   values (value1, value2, ....);
COMMIT WORK
```


Using ESQL/C to Begin Work Without Replication

The following example shows how to use Informix® ESQL/C to begin work without replication as well as update the Enterprise Replication shadow columns **cdrserver** and **cdrtime**:

```
MAIN (argc, argv)
   INT      argc;
   CHAR      *argv[];
{
   EXEC SQL CHAR      stmt[256];
   EXEC SQL database mydatabase;

   sprintf(stmt, "BEGIN WORK WITHOUT REPLICATION");
   EXEC SQL execute immediate :stmt;

   EXEC SQL insert into mytable (col1, col2, ...)
      values (value1, value2, ...);
   EXEC SQL commit work;
}
```

Important: You must use the following syntax when you issue the BEGIN WORK WITHOUT REPLICATION statement from Informix ESQL/C programs. Do not use the '$' syntax.

```
sprintf(stmt, "BEGIN WORK WITHOUT REPLICATION");
EXEC SQL execute immediate :stmt;
```


Preparing to Replicate User-Defined Types

You must install and register user-defined types on all database servers prior to starting replication.

For Enterprise Replication to be able to replicate opaque user-defined types (UDTs), the UDT designer must provide two support functions, **streamwrite()** and **streamread()**. For more information, see Replication of opaque user-defined data types.

Preparing to Replicate User-Defined Routines

You must install and register user-defined routines on all database servers prior to starting replication.

Preparing Tables for Conflict Resolution

To use any conflict-resolution rule other than ignore or always-apply, you must define the shadow columns, **cdrserver** and **cdrtime** in the tables on both the source and target servers involved in replication.

To define the **cdrserver** and **cdrtime** shadow columns when you create a new table, use the WITH CRCOLS clause. For example, the following statement creates a new table named **customer** with a data column named **id** and the two shadow columns:

```
CREATE TABLE customer(id int) WITH CRCOLS;
```

To add the **cdrserver** and **cdrtime** shadow columns to an existing replicated table:

1. Set alter mode on the table by running the **cdr alter --on** command.
2. Alter the table using the ADD CRCOLS clause.
3. Unset alter mode on the table by running the **cdr alter --off** command.

Adding CRCOLS columns to an existing table can result in a slow alter operation if any of the table columns have data types that require a slow alter. If a slow alter operation is necessary, make sure you have disk space at least twice the size of the original table, plus extra log space.

For example, the following statement adds the shadow columns to an existing table named **customer**:

```
ALTER TABLE customer ADD CRCOLS;
```

You cannot drop conflict resolution shadow columns while replication is active. To drop the **cdrserver** and **cdrtime** shadow columns, stop replication and then use the DROP CRCOLS clause with the ALTER TABLE statement. For example, the following statement drops the two shadow columns from a table named **customer**:

```
ALTER TABLE customer DROP CRCOLS;
```

Related concepts:
Shadow columns
Shadow column disk space
SQL statements and replication
Related information:
Enterprise Replication shadow columns
Using the WITH CRCOLS Option

Preparing Tables for a Consistency Check Index

To improve the speed of consistency checking with an index, you must define the **ifx_replcheck** shadow column in the tables on both the source and target servers involved in replication.

To define the **ifx_replcheck** shadow column when you create a new table, use the WITH REPLCHECK clause. For example, the following statement creates a new table named **customer** with a data column named **id** and the **ifx_replcheck** shadow column:

```
CREATE TABLE customer(id int) WITH REPLCHECK;
```

To add the **ifx_replcheck** shadow column to an existing replicated table:

1. Set alter mode on the table by running the **cdr alter --on** command.
2. Alter the table using the ADD REPLCHECK clause.
3. Unset alter mode on the table by running the **cdr alter --off** command.

Because altering a table to add the **ifx_replcheck** shadow column is a slow alter operation, make sure you have disk space at least twice the size of the original table plus log space.

For example, the following statements add the **ifx_replcheck** shadow column to an existing table named **customer**:

```
ALTER TABLE customer ADD REPLCHECK;
```

To drop the **ifx_replcheck** shadow column, use the DROP REPLCHECK clause with the ALTER TABLE statement. For example, the following statements drop the **ifx_replcheck** shadow column from a table named **customer**:

```
ALTER TABLE customer DROP REPLCHECK;
```

For more information on the CREATE TABLE and ALTER TABLE statements, see the sections in the *IBM® Informix® Guide to SQL: Syntax*.

Related concepts:
Shadow column disk space
Shadow columns
SQL statements and replication
Related tasks:
Indexing the ifx_replcheck Column
Related information:
Enterprise Replication shadow columns
Using the WITH REPLCHECK Keywords

Preparing tables without primary keys

The data columns in your table might not need a primary key. To replicate tables that do not have primary keys, you can specify a unique index or add the ERKEY shadow columns.

You can specify an existing unique index or unique constraint as the replication key when you define the replicate. Use the **--key** or **--anyUniqueKey** option with the **cdr define replicate** or **cdr define template** commands.

If you create a replicated table through a grid, the ERKEY shadow columns are automatically created and included in the replicate definition.

To add ERKEY shadow columns:

1. Add the ERKEY shadow columns when you create at table by using the WITH ERKEY keywords with the CREATE TABLE statement. For example, the following statement adds the ERKEY shadow columns to a table named **customer**:

   ```
   CREATE TABLE customer (id int) WITH ERKEY;
   ```

   The ERKEY shadow columns are named **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**.
2. Define the replicate. If you define a replicate by using the **cdr define replicate** command, include the **--erkey** option. If you define a template by using the **cdr define template** command, the ERKEY columns are included in the replicate definition automatically.

To add the ERKEY shadow columns to an existing table that you want to start replicating:

1. Run the ALTER TABLE statement with the ADD ERKEY clause. For example, the following statement adds the ERKEY shadow columns to an existing table named **customer**:

```
ALTER TABLE customer ADD ERKEY;
```

Occasionally, you might need to drop the ERKEY shadow columns; for example, if you are reverting to an earlier version of the database server.

To drop the ERKEY shadow columns from a replicated table:

1. Run the **cdr remaster** command without the **--erkey** option.
2. Run the DROP ERKEY clause with the ALTER TABLE statement.

For example, the following statement drops the ERKEY shadow columns from a table named **customer**:

```
ALTER TABLE customer DROP ERKEY;
```

Related concepts:
Unique key for replication
SQL statements and replication
Related tasks:
Creating replicated tables through a grid
Attaching a New Fragment to a Replicated Table
Related reference:
cdr define replicate
cdr remaster
cdr change replicate
cdr define template
Related information:
Using the WITH ERKEY Keywords
Enterprise Replication shadow columns

Preparing Logging Databases

Databases on all server instances involved in replication must be created with logging. For best results, use unbuffered logging. For more information, see Unbuffered Logging.

Related reference:
cdr start sec2er

Preparing for Role Separation (UNIX)

You can use role separation to allow members of the DBSA group to run Enterprise Replication commands, in addition to the user **informix**. For some Enterprise Replication commands, you must grant the DBSA user additional permissions on tables or files. For non-root servers, role separation is not supported. Only the owner of a non-root server is allowed to run the Enterprise Replication commands that require additional permissions for a DBSA.

The DBSA user who runs Enterprise Replication commands must be a member of the DBSA group on all of the replication servers in the domain.

The following table describes the permissions that are needed for each command.

Table 1. Permissions for the DBSA user

Command	Type of Permission	Tables, Files, or Database
cdr check replicate **cdr check replicateset** **cdr define replicate** **cdr define replicateset** **cdr define template** **cdr realize template** **cdr sync replicate** **cdr sync replicateset**	INSERT UPDATE DELETE	The tables that participate in replication. Must be granted on all replication servers in the domain.
The following commands with the **--background** option: • **cdr check replicate** • **cdr check replicateset** • **cdr sync replicate** • **cdr sync replicateset**	CONNECT or INSERT, depending on the object	**sysadmin** database: CONNECT **ph_task** table in the **sysadmin** database: INSERT Must be granted on the database server from which the command is run.
cdr define repair **cdr start repair** **cdr stop repair** **cdr delete repair** The following commands with the **--syncdatasource** option: • **cdr realize template** • **cdr start replicate** • **cdr start replicateset**	INSERT, UPDATE, or DELETE, depending on the table	The following **syscdr** tables: • **rsncjobdef_tab**: INSERT, UPDATE, DELETE • **rsncjobdef**: UPDATE • **rsncprocnames_tab**: INSERT • **rsncjobdeps**: INSERT Must be granted on all replication servers in the domain.
cdr repair **cdr view atsdir** **cdr view risdir**	read	ATS and RIS files Must be granted on the database server on which the files are located.

To update the permissions on a table or database, use the GRANT statement. For example, the following statement grants INSERT and UPDATE permissions on the **rsncjobdef_tab** table to the DBSA member with the user name of **carlo**:

```
GRANT INSERT, UPDATE ON rsncjobdef_tab TO carlo;
```

For more information about the GRANT statement, see the *IBM® Informix® Guide to SQL: Syntax*.

To update the permissions on ATS and RIS files, use an operating system command, such as the **chown** UNIX command.

Related reference:
cdr check replicate
cdr check replicateset
cdr sync replicate
cdr sync replicateset
cdr repair
cdr view
cdr realize template
cdr define replicate
cdr define replicateset
cdr start replicate
cdr start replicateset
cdr define template

Load and unload data

You can load data into or unload data out of tables in your replication environment in various ways, depending on your circumstances.

If you have not yet set up your replication environment, for loading data, you can use the following tools:

- High-Performance Loader
- onunload and onload Utilities
- dbexport and dbimport utilities
- UNLOAD and LOAD statements
- External tables

When you unload and load data, you must use the same type of utility for both the unload and load operations. For example, you cannot unload data with the **onunload** utility and then load the data with a LOAD statement.

Existing replication environment

If you are adding a table to your already existing replication environment, Enterprise Replication provides an initial synchronization feature that allows you to easily bring a new table up-to-date with replication. You can synchronize the new table with data on the source server you specify when you start the new replicate, or when you add a new participant to an existing replicate. You do not need to suspend any servers that are replicating data while you add the new replicate and synchronize it.

If you want to use load and unload tools on tables that are already being replicated, you should block replication while you prepare the table. Unlogged changes to a table, such as data added by a light append, can be replicated to other tables.

If a table that you plan to replicate includes the CRCOLS or REPLCHECK shadow columns, the statements that you use for unloading the data must explicitly name the shadow columns. If you use the SELECT statement with * FROM *table_name* to the data to unload, the data from the shadow columns is not unloaded. To include the shadow columns in the unloaded data, explicitly name them. For example, use a statement like the following:

```
SELECT cdrserver, cdrtime, ifx_replcheck, * FROM table_name
```

If a table that you plan to replicate includes ERKEY shadow columns, you cannot unload and then load the data from these columns and preserve the original values. If you need to preserve the values of the ERKEY shadow columns, use synchronization to propagate the values.

- High-Performance Loader
  The High-Performance Loader (HPL) provides a high-speed tool for moving data between databases.
- onunload and onload Utilities
  You can use the **onunload** and **onload** utilities to unload and load an entire table.
- dbexport and dbimport Utilities
- UNLOAD and LOAD Statements
  The UNLOAD and LOAD statements allow you to move data within the context of an SQL program.

Related concepts:
Blocking Replication
Setting Up Database Server Groups for High-Availability Cluster Servers
Shadow columns
Related tasks:
Initially Synchronizing Data Among Database Servers
Related information:
Moving data with external tables

High-Performance Loader

The High-Performance Loader (HPL) provides a high-speed tool for moving data between databases.

How you use the HPL depends on how you defined the tables to replicate.

If the table contains shadow columns, you must:

- Include all the shadow column names in your map when you load the data.
- Use express mode to load data that contains shadow columns. You must perform a level-0 archive after completion.

You can also use deluxe mode without replication to load data. After a deluxe mode load, you do not need to perform a level-0 archive. Deluxe mode also allows you to load TEXT and BYTE data and opaque user-defined types.

For information about HPL, refer to the *IBM® Informix® High-Performance Loader User's Guide*.

onunload and onload Utilities

You can use the **onunload** and **onload** utilities to unload and load an entire table.

If you want to unload selected columns of a table, you must use either the UNLOAD statement or the HPL.

Restriction: You can only use the **onunload** and **onload** utilities in identical (homogeneous) environments.
If you use the **onload** utility while replication is active, you must synchronize the data after you finish loading the data.

Related information:
The onunload and onload utilities

dbexport and dbimport Utilities

If you need to copy an entire database for replication, you can use the **dbexport** and **dbimport** utilities. These utilities unload an entire database, including its schema, and then re-create the database. If you want to move selected tables or selected columns of a table, you must use some other utility.

Related information:
The dbexport and dbimport utilities

UNLOAD and LOAD Statements

The UNLOAD and LOAD statements allow you to move data within the context of an SQL program.

If the table contains shadow columns, you must:

- Include all shadow columns in your map when you unload the data.
- List the columns that you want to load in the INSERT statement and explicitly include the shadow columns in the list when you load your data.

For more information about the UNLOAD and LOAD statements, see the *IBM® Informix® Guide to SQL: Syntax*.

Data Preparation Example

The following examples show how to add a new participant (**delta**) to an existing replicate by two different methods:

- Using the **cdr start replicate** command
  This method is simple and can be done while replication is online.
- Using the LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION statements.
  If you use HPL, this method can be faster for a large table.

Replicate **zebra** replicates data from table **table1** for the following database servers: **alpha**, **beta**, and **gamma**.

The servers **alpha**, **beta**, and **gamma** belong to the server groups **g_alpha**, **g_beta**, and **g_gamma**, respectively. Assume that **alpha** is the database server from which you want to get the initial copy of the data.

- Using the cdr start replicate Command
- Using LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION
  When you add a new participant to an existing replicate, you can unload and load data without replication.

Using the cdr start replicate Command

To add a new participant to an existing replicate

1. Declare server **delta** to Enterprise Replication. For example:

   ```
   cdr def ser -c delta -I -S g_alpha g_delta
   ```

   At the end of this step, all servers in the replication environment include information in the **syscdr** database about **delta**, and **delta** has information about all other servers.

2. Add **delta** as a participant to replicate **zebra**. For example:

   ```
   cdr cha rep -a zebra "dbname@g_delta:owner.table1"
   ```

   This step updates the replication catalog. The servers **alpha**, **beta**, and **gamma** do not queue any qualifying replication data for **delta** because the replicate on **delta**, although defined, has not been started.

3. Start replication for replicate **zebra** on **delta**.

   ```
   cdr sta rep zebra g_delta -S g_alpha -e delete
   ```

   The **-S g_alpha** option specifies that the server **alpha** be used as the source for data synchronization.

   The **-e delete** option indicates that if there are rows on the target server, **delta**, that are not present on the synchronization data server (**alpha**) then those rows are deleted

   Do not run any transactions on **delta** that affect table **table1** until you finish the synchronization process.

Using LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION

When you add a new participant to an existing replicate, you can unload and load data without replication.

To add a new participant to an existing replicate

1. Add the server **delta** to the Enterprise Replication domain. For example:

   ```
   cdr def ser -c delta -I -S g_alpha g_delta
   ```

   At the end of this step, all servers in the replication environment include information in the **syscdr** database about **delta**, and **delta** has information about all other servers.

2. Add **delta** as a participant to replicate **zebra**. For example:

   ```
   cdr cha rep -a zebra "P dbname@g_delta:owner.table1" \
   "select * from table1"
   ```

   This step updates the replication catalog. The servers **alpha**, **beta**, and **gamma** do not queue any qualifying replication data for **delta** because the replicate on **delta**, although defined, has not been started.

3. Suspend server **delta** on **alpha**, **beta**, and **gamma**.

   ```
   cdr sus ser g_delta g_alpha g_beta g_gamma
   ```

   As a result of this step, replication data is queued for **delta**, but no data is delivered.

4. Start replication for replicate **zebra** on **delta**.

   ```
   cdr sta rep zebra g_delta
   ```

   This step causes servers **alpha**, **beta**, and **gamma** to start queuing data for **delta**. No data is delivered to **delta** because **delta** is suspended. Then, **delta** queues and delivers qualifying data (if any) to the other servers.

   Do not run any transactions on **delta** that affect table **table1** until you finish the synchronization process.

5. Unload data from table **table1** using the UNLOAD statement or the **unload** utility on HPL.
6. Copy the unloaded data to **delta**.
7. Start transactions with BEGIN WORK WITHOUT REPLICATION, load the data using the LOAD statement, and commit the transactions. If you used the HPL to unload the data in step 5, then use the HPL Deluxe load without replication to load the data into **table1** on **delta**.
8. Resume server **delta** on **alpha**, **beta**, and **gamma**.

   ```
   cdr res ser g_delta g_alpha g_beta g_gamma
   ```

   This step starts the flow of data from **alpha**, **beta**, and **gamma** to **delta**.
   At this point, you might see some transactions aborted because of conflict. Transactions can abort because **alpha**, **beta**, and **gamma** started queuing data from **delta** in step 4. However, those same transactions might have been moved in steps 5 and 7.

You must declare replication on server **delta** and then immediately suspend replication because, while you are preparing the replicates and unloading and loading files, the other servers in the replicate (**alpha**, **beta**, and **gamma**) might be collecting information that needs to be replicated. After you finish loading the initial data to **delta** and resume replication, the information that was generated during the loading process can be replicated.

Using High-Availability Clusters with Enterprise Replication

In This Chapter

This chapter covers how to include other data replication solutions, such as high-availability data replication, in your Enterprise Replication system. The following topics are covered:

- The design of a high-availability cluster replication system
- Preparing a high-availability cluster database server
- Managing Enterprise Replication with a high-availability cluster

For a complete description of data replication, see the *IBM Informix Administrator's Guide*.

- High-availability replication systems
  You can combine IBM® Informix® Enterprise Replication and high-availability clusters to create a high-availability replication system.
- Managing Enterprise Replication with High-Availability Clusters

Related concepts:
Preparing the Replication Environment
Grid setup and management
Shard cluster setup
Managing Replication Servers and Replicates
Monitor and troubleshooting Enterprise Replication
Related tasks:
Defining Replication Servers, Replicates, Participants, and Replicate Sets

High-availability replication systems

You can combine IBM® Informix® Enterprise Replication and high-availability clusters to create a high-availability replication system.

A high-availability cluster consists of two types of database servers:

- A primary database server, which receives updates, and can participate in Enterprise Replication.
- Secondary servers, which mirror the primary server and are perpetually applying logical-log records from the primary server, and cannot participate in Enterprise Replication.

A minimal high-availability cluster consists of a primary server and a HDR secondary server that are tightly coupled. Transactions on the primary server are not committed until the log records containing the transactions are sent to the HDR secondary server.

High-availability clusters can also contain shared-disk (SD) secondary servers and remote standalone (RS) secondary servers. A SD secondary server does not maintain a copy of the physical database on its own disk space; it shares disks with the primary server. An RS secondary servers maintains a copy of the physical database on its own disk space.

If the primary server in a high-availability cluster becomes unavailable, one of the secondary servers takes over the role of the primary server. In a high-availability replication system, if the primary server fails, a secondary database is promoted to primary server, and Enterprise Replication can continue with the new primary server.

You can configure Connection Managers to direct client requests to replication servers, and to control which secondary server takes over if the primary server becomes unavailable.

A high-availability replication system is effective when you use a hierarchical or a forest of trees topology.

- High-Availability Clusters in a Hierarchical Tree Topology
  With a hierarchical tree topology, parent servers are good candidates for using high-availability clusters to provide backup servers.
- Using high-availability clusters in a forest of trees topology
- Setting Up Database Server Groups for High-Availability Cluster Servers
  When defining a high-availability cluster within Enterprise Replication, the cluster must appear to be a single logical entity within the replication domain. Define the servers within the same database server group in the sqlhosts file.

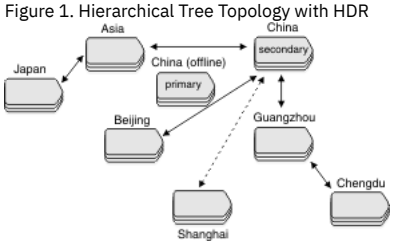
Related concepts:
Update-Anywhere Replication System
Related information:
The sqlhosts information

High-Availability Clusters in a Hierarchical Tree Topology

With a hierarchical tree topology, parent servers are good candidates for using high-availability clusters to provide backup servers.

The following example is based on the example in Figure 1.

If **China** fails, then **Beijing** and **Shanghai** can no longer replicate with other servers in the replication system; **Guangzhou** and **Chengdu** can replicate only with each other. However, if **China** was part of a high-availability cluster, when it failed, the secondary server would replace it and replication would continue, as illustrated in Figure 1.
Figure 1. Hierarchical Tree Topology with HDR

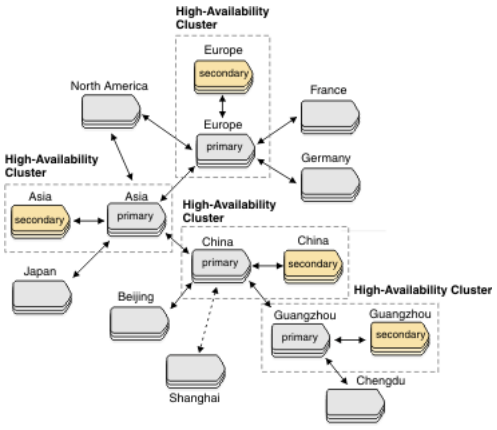


In this example, **Asia** and **Guangzhou**, which are also parent servers, might also benefit from using a high-availability cluster to ensure high availability.

Using high-availability clusters in a forest of trees topology

Use a high-availability cluster to ensure that all servers retain access to the replication system in a forest of trees topology.

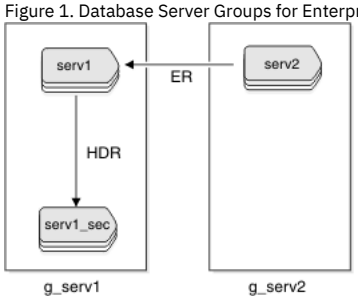
For example, in Figure 1, **Asia**, **Europe**, **China**, and **Guangzhou** should use high-availability clusters to provide backup servers, as illustrated in Figure 1.
Figure 1. High-Availability Clusters in a Forest-of-Trees Topology



Setting Up Database Server Groups for High-Availability Cluster Servers

When defining a high-availability cluster within Enterprise Replication, the cluster must appear to be a single logical entity within the replication domain. Define the servers within the same database server group in the sqlhosts file.

For example, Figure 1 illustrates two Enterprise Replication nodes, one of which is an HDR pair.
Figure 1. Database Server Groups for Enterprise Replication with HDR



In this example, the HDR pair consists of the primary server, **serv1**, and the secondary server, **serv1_sec**. These two servers belong to the same database server group, **g_serv1**. The non-HDR server, **serv2**, belongs to the database server group **g_serv2**. The following example displays the sqlhosts file for this configuration:

```
#dbservername    nettype    hostname      servicename   options
 g_serv1         group      -             -             i=1
 serv1           ontlitcp   machine1pri   port1         g=g_serv1
 serv1_sec       ontlitcp   machine1sec   port1         g=g_serv1
 g_serv2         group      -             -             i=2
 serv2           ontlitcp   machine2      port1         g=g_serv2
```

Important: If you use the g=server option in the group member definition, you can put the definition anywhere in the sqlhosts file.
Either HDR or Enterprise Replication can be set up first on the HDR pair **serv1** and **serv1_sec**, but Enterprise Replication **cdr** commands must be run only on the primary server. If any **cdr** commands are attempted on the secondary server, a –117 error is returned: `Attempting to process a cdr command on an HDR secondary server`.

Related concepts:
Load and unload data
Creating sqlhost group entries for replication servers
Related information:
sqlhosts connectivity information

Managing Enterprise Replication with High-Availability Clusters

This section describes how to manage Enterprise Replication with HDR in the following areas:

- Failure of the primary server in a high-availability cluster
- Performance considerations

- Failover for High-availability clusters in an Enterprise Replication environment
- Replication latency for secondary servers
  When you combine Enterprise Replication with high-availability clusters, replication latency can increase.

Failover for High-availability clusters in an Enterprise Replication environment

If you configure connection management for failover, Connection Managers can promote a secondary server to the primary-server if the primary server fails. If connection management is not configured to control failover, the **onmode -d make primary** command can promote a secondary server to the primary-server role. In either of these cases, Enterprise Replication automatically connects to the new primary server.

If the primary server fails, and you manually change a secondary server to a standard server, you must complete the following steps to prevent Enterprise Replication from starting on all servers in cluster.

Run the following commands on the secondary server:

1. **onmode -s**
2. **onmode -d standard**
3. **cdr start**

If Enterprise Replication is running on the secondary server, and you want to restart the server that was the primary server, but without Enterprise Replication and high-availability cluster replication, run the **oninit -D** command. You can then stop Enterprise Replication on the standard server and reestablish the primary server.

First, run the following commands on the standard server:

1. **cdr stop**
2. **onmode -d secondary** *primary_ha_alias*

Second, run the following commands on the primary server:

1. **oninit**
2. **cdr start**

To split an active cluster into two standalone servers, you must restart the database servers with the **oninit -D** command to prevent Enterprise Replication from starting on either server after they are split.

To remove a server from a cluster, run the **cdr delete server –force** *ha_alias* command, where *ha_alias* is an Enterprise Replication group name, to remove Enterprise Replication from that server. For example, the two HDR servers are being split and the secondary server is to be used for reporting purposes. After the report processing is complete, HDR can be reestablished. cdr delete server shows how to remove a secondary server from a high-availability cluster and Enterprise Replication.

Table 1. Removing the Secondary Server from a cluster and ER

Step	On the Primary	On the Secondary
1.	**onmode -d standard** *secondary_ha_alias*	
2.		Run **onmode -d standard**
3.		Run **cdr delete server -f** *ha_alias*

If the HDR primary server has problems communicating to its secondary server, Enterprise Replication is in a suspended state until one of the following actions is taken:

- Resolve the connection problem between HDR pairs.
- Convert the primary server to standard mode.

Related reference:
cdr delete server
Related information:
Connection management through the Connection Manager

Replication latency for secondary servers

When you combine Enterprise Replication with high-availability clusters, replication latency can increase.

When Enterprise Replication is running on a high-availability cluster, some operations cannot be performed until the logs are shipped to the secondary server. By default, the logs are shipped to secondary servers after 50 replicated transactions are applied, or 5 seconds elapse. This delay prevents possible inconsistency within the Enterprise Replication domain during a failover to a secondary server.

You can control replication latency for high-availability data replication (HDR) servers in one of the following ways

- Set HDR replication to fully synchronous, nearly synchronous, or asynchronous mode.
- Set HDR replication to HDR SYNC.
- Adjust the DRINTERVAL configuration parameter to specify a different interval between flushing the high-availability data-replication buffer.

If you combine Enterprise Replication with shared-disk secondary servers, you can reduce replication latency by setting the CDR_MAX_FLUSH_SIZE configuration parameter to 1 to flush the logs after each replicated transaction.

Related reference:
CDR_MAX_FLUSH_SIZE configuration parameter
Related information:
DRINTERVAL configuration parameter
HDR_TXN_SCOPE configuration parameter
Replication of primary-server data to secondary servers
Fully synchronous mode for HDR replication
Nearly synchronous mode for HDR replication

Defining Replication Servers, Replicates, Participants, and Replicate Sets

These topics describe the steps defining and starting Enterprise Replication.

To define and start replication:

1. Initialize the database server.
2. Create a replication domain by defining replication servers.
3. Configure replication by defining replicates, and optionally grouping replicates into a replicate set. The replicate definition includes information about the participants, replication options, frequency, and conflict-resolution rules and scope.
4. Specify the data to replicate by defining participants. A participant definition specifies the data (database, table, and columns) that should be replicated.
5. Synchronize the data among the replicates.

- Starting Database Servers
  The database server must be online before you can define it as a replication server.
- Defining Replication Servers
  You must define a replication server to create a replication domain or to add a server to an existing domain.
- Define a replicate
  To define a replicate, use the **cdr define replicate** command.
- Define replicate sets
  When you define a replicate set, you specify the type of replicate set, the replicates that belong to the replicate set, and the frequency of replication for the member replicates.
- Initially Synchronizing Data Among Database Servers
- Set up replication through templates
  Enterprise Replication provides templates to allow easy setup and deployment of replication for clients with large numbers of tables to replicate. A template uses schema information about a database, a group of tables, columns, and replication keys to define a group of master replicates and a replicate set.

Related concepts:
Preparing the Replication Environment
Using High-Availability Clusters with Enterprise Replication
Grid setup and management
Shard cluster setup
Managing Replication Servers and Replicates
Monitor and troubleshooting Enterprise Replication

Starting Database Servers

The database server must be online before you can define it as a replication server.

To bring the server from offline to online, issue the following command for your operating system.

Operating System	Command
UNIX	**oninit**
Windows	**start** *dbservername*

To bring the server from quiescent mode to online on either UNIX or Windows, enter **onmode -m**.

For more information on initializing the database server, see the chapter on database server operating modes in the *IBM® Informix® Administrator's Guide*.

Defining Replication Servers

You must define a replication server to create a replication domain or to add a server to an existing domain.

The database server must be online.

You must be the Enterprise Replication server administrator to define the replication server.

You can define replication servers using two different methods:

- The **cdr** utility
- Cloning

To define the replication server in a new domain by using the **cdr** utility, use the **cdr define server** command to connect to the database server and specify the database server group name. For example, the following command connects to a server called **stan** and creates a domain containing the database server group **g_stan**:

```
cdr define server  --connect=stan --init g_stan
```

The **--init** option specifies the database server group to add to the replication domain. If the INFORMIXSERVER environment variable is not set to the server that you are defining, specify the **--connect=***server_name* option. You can also configure replication attributes for the server.

To define a replication server in an existing domain by using the **cdr** utility, include the **--sync=***sync_server* option with the **cdr define server** command to synchronize the global catalog with an existing server. For example, the following command adds a server group named **g_oliver** to the domain created in the previous command, using **g_stan** as the synchronization server:

```
cdr define server  --connect=oliver --init g_oliver --sync=g_stan
```

You can specify any existing server in the domain, however, if you define a server as a nonroot or a leaf server, then the synchronization server becomes the parent of the new server. For example, if you add a server **kauai** as a leaf server and want its parent to be **hawaii**, then specify **hawaii** with the **--sync** option.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the appropriate configuration parameters when defining a replication server. If the CDR_QDATA_SBSPACE or the CDR_DBSPACE configuration parameter is not set and the server has a storage pool with sufficient space, the **cdr define server** command automatically creates the necessary disk space and sets the configuration parameters to appropriate values.

The maximum number of replication servers that you can define is 32767.

- Creating a new domain by cloning a server
  You can create a new replication domain by cloning a server and then converting the two database servers to replication servers. Use cloning and conversion if you want to set up replication based on the data on a source server that is not yet running Enterprise Replication.
- Adding a server to the domain by cloning a server
  You can add a replication server to an existing replication domain by using the **ifxclone** utility to clone an existing replication server onto a target database server.
- Customizing the Replication Server Definition
  You can specify replication attributes of a server when you define it.

Related concepts:
Row Data sbspaces
Choosing a Replication Network Topology
Enterprise Replication Server administrator
Modify server attributes
Related tasks:
Setting Up Failed Transaction Logging
Related reference:
cdr define server
cdr define replicate

Creating a new domain by cloning a server

You can create a new replication domain by cloning a server and then converting the two database servers to replication servers. Use cloning and conversion if you want to set up replication based on the data on a source server that is not yet running Enterprise Replication.

Because the source server does not have Enterprise Replication defined, you use the **ifxclone** utility to create a cluster containing a primary server and remote stand-alone (RS) secondary server. The conversion process converts the cluster to a pair of replication servers in a new domain.
To create a new domain with two replication servers:

1. On the source server, prepare the server environment for Enterprise Replication, such as configuring sqlhosts information and setting the necessary configuration parameters.
2. On both servers, complete the **ifxclone** prerequisites for all servers, such as setting the required configuration parameters and environment variables.
3. On the target server, complete the **ifxclone** prerequisites for an RS secondary server, such as creating all of the chunks that exist on the source server. You can use the **--createchunkfile** option (**-k**) of the **ifxclone** utility to automatically create cooked chunks on the target server.
4. On the target server, run the **ifxclone** command with the **--disposition=RSS** option to clone the data and the configuration of the source server onto the target server. Do not include the **--useLocal** option.
5. On the source server, run the **cdr check sec2er** command to determine if conversion to replication servers is possible.
6. Solve any error conditions identified by the **cdr check sec2er** command and rerun it until its output indicates that conversion will be successful. You can also solve warning conditions.
7. On the source server, run the **cdr start sec2er** command to convert both servers to replication servers and create a new replication domain.

To add other servers to the domain, you can clone a replication server.

- Example of creating a new replication domain by cloning
  This is an example of creating a new replication domain based on the data and configuration on a source database server that does not have replication defined. The three additional replication servers in the domain are added by cloning the source server.

Related concepts:
Preparing the Replication Environment
Related tasks:
Adding a server to the domain by cloning a server
Related information:
The ifxclone utility

Example of creating a new replication domain by cloning

This is an example of creating a new replication domain based on the data and configuration on a source database server that does not have replication defined. The three additional replication servers in the domain are added by cloning the source server.

This example creates a replication domain and grid that contain four replication servers: **serv1**, **serv2**, **serv3**, **serv4**. Each server computer has the Informix® database server installed. The source server contains the **stores_demo** database.

1. On the **serv1** server, set the CDR_QDATA_SBSPACE configuration parameter.
2. On the **serv1** server, set the value of the ENABLE_SNAPSHOT_CLONE configuration parameter to `1` in the onconfig file.
3. On the **serv1** server, add the following sqlhosts information about **serv1** and **serv2**:

```
gserv1     group    -                      -     i=143
serv1      ontlitcp ny.usa.com             1230  g=gserv1
gserv2     group    -                      -     i=144
serv2      ontlitcp tokyo.japan.com        1231  g=gserv2
```

4. On both the **serv1** and **serv2** servers, complete the **ifxclone** prerequisites for all servers, such as setting the required configuration parameters and environment variables.
   Set these environment variables:
   - INFORMIXDIR
   - INFORMIXSERVER
   - INFORMIXSQLHOSTS
   - ONCONFIG

   Set these configuration parameters to the same values on both servers:
   - DRAUTO
   - DRINTERVAL
   - DRTIMEOUT
   - LOGBUFF
   - LOGFILES
   - LOGSIZE
   - LTAPEBLK
   - LTAPESIZE
   - ROOTNAME
   - ROOTSIZE
   - PHYSBUFF
   - PHYSFILE
   - STACKSIZE
   - TAPEBLK
   - TAPESIZE
5. On the **serv2** server, create all of the chunks that exist on the **serv1** server. You can use the **--createchunkfile** option (**-k**) of the **ifxclone** utility to automatically create cooked chunks on the target server.
6. On the **serv2** server, run the **ifxclone** command with the **--disposition=RSS** option to clone the data and the configuration of the **serv1** server onto the **serv2** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv2 --targetIP=192.168.0.2
--targetPort=1231 --disposition=RSS --createchunkfile
```

7. On the **serv1** server, run the **cdr check sec2er** command to determine if conversion to replication servers is possible:

```
$cdr check sec2er -c gserv1 gserv2
Secondary conversion to ER is possible.
```

8. On the **serv1** server, run the **cdr start sec2er** command to convert both servers to replication servers, create a new replication domain, create and start replicates based on all the tables on the **serv1** server:

```
cdr start sec2er -c gserv1 gserv2
```

9. On the **serv3** and **serv4** servers, provision chunk paths and other storage to the same paths and at least the same sizes as on the **serv1** server.
10. On the **serv3** server, run the **ifxclone** command with the **--disposition=ER** option to clone the data and the configuration of the **serv1** server onto the **serv3** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv3 --targetIP=192.168.0.3
--targetPort=1232 --disposition=ER
```

11. On the **serv4** server, run the **ifxclone** command with the **--disposition=ER** option to clone the data and the configuration of the **serv1** server onto the **serv4** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv4 --targetIP=192.168.0.4
--targetPort=1233 --disposition=ER
```

12. Edit the sqlhosts files on all four servers so that they each have the following information:

```
gserv1     group    -                      -     i=143
serv1      ontlitcp ny.usa.com             1230  g=gserv1
gserv2     group    -                      -     i=144
serv2      ontlitcp tokyo.japan.com        1231  g=gserv2
gserv3     group    -                      -     i=145
serv3      ontlitcp rome.italy.com         1232  g=gserv3
gserv4     group    -                      -     i=146
serv4      ontlitcp perth.australia.com    1233  g=gserv4
```

Related reference:
cdr start sec2er
cdr check sec2er
Related information:
The ifxclone utility

Adding a server to the domain by cloning a server

You can add a replication server to an existing replication domain by using the **ifxclone** utility to clone an existing replication server onto a target database server.

Enterprise Replication must be active on the source server. The source server should not have any stopped or suspended replicates or any shadow replicates defined. You must be user **informix** or member of the **informix** group to run the **ifxclone** utility.

IBM® Informix® database software must be installed on the target server.

Cloning a server defines replication on the target server, copies the data, and adds the target server to all replicates in which the source server participates. The onconfig file and the **sqlhosts** file are copied from the source server to the target server and updated with the target server information.
To clone a replication server by using the **ifxclone** utility:

1. On the source server, set the value of the ENABLE_SNAPSHOT_COPY configuration parameter to `1` in the onconfig file.
2. On the target server, create the following directories, if they exist on the source server. The directories must be the same on both servers:
   - ATS and RIS directories
   - Log staging directory
3. On the target server, synchronize the system clock with the source server.
4. On the target server, provision chunk paths and other storage to the same paths and at least the same sizes as on the source server. Ensure that the target server has at least as much memory and disk space resources as the source server. You can use the **--createchunkfile** option (**-k**) of the **ifxclone** utility to automatically create cooked chunks on the target server.
5. On the target server, run the **ifxclone** command. You must provide the following information to the **ifxclone** utility:
   - Source server name
   - Source IP address
   - Source port
   - Target server name
   - Target IP address
   - Target port

   Include the **--disposition=ER** option.

   Optional: Include the **--createchunkfile** option.

   If the source server replicates serial columns, use the **--configParam** option to set the value of the CDR_SERIAL configuration parameter to ensure that serial values do not conflict between replication servers.

   The **ifxclone** utility has the following format for cloning a replication server:

   ```
   ifxclone --source=source_name --sourceIP=source_IP
   --sourcePort=source_port --target=target_name
    --targetIP=target_IP --targetPort=target_port
   --disposition=ER --createchunkfile
   ```

6. On all other replication servers in the domain, edit the **sqlhosts** file to add entries for the new replication server.

Related concepts:
Time synchronization
Related tasks:
Creating a new domain by cloning a server
Adding a replication server to a grid by cloning
Related reference:
Set configuration parameters for replication
Related information:
The ifxclone utility

Customizing the Replication Server Definition

You can specify replication attributes of a server when you define it.

When you define a replication server, you can specify the following attributes in the **cdr define server** command:

- Set the idle timeout.
  To specify the time (in minutes) that you want to allow the connection between two Enterprise Replication servers to remain idle before disconnecting, use the **--idle=***timeout* option.

  You can later change the values of this attribute with the **cdr modify server** command.

- Specify the location of the ATS and RIS directories.
  To use ATS, specify the directory for the Aborted Transaction Spooling (ATS) files for the server using **--ats=***dir* or**--ris=***dir* . To prevent either ATS or RIS file generation, set the directory to **/dev/null** (UNIX) or **NUL** (Windows).

  You can later change the values of these attributes with the **cdr modify server** command.

- Specify the format of the ATS and RIS files.
  Use the **–atsrisformat=***type* option to specify whether the ATS and RIS files are generated in text format, XML format, or both formats.

  You can later change the values of this attribute with the **cdr modify server** command.

- Specify the type of server if you are using hierarchical replication:

- To specify the server as a nonroot server, use the **--nonroot** option.
- To specify the server as a leaf server, use the **--leaf** option.

If neither **--leaf** nor **--nonroot** is specified, the server is defined as a root server. The parent server is the server specified by the **--sync=***sync_server* option.

Related concepts:
Choosing a Replication Network Topology
Enterprise Replication Terminology
Related tasks:
Creating ATS and RIS directories
Related reference:
cdr define server
cdr modify server

Define a replicate

To define a replicate, use the **cdr define replicate** command.

You can provide the following information in the replicate definition:

- Participants
- Create as a master replicate
- Conflict resolution rules and scope
- Replication frequency
- Error logging
- Replicate full rows or only changed columns
- IEEE or canonical message formats
- Database triggers
- Code set conversion between replicates
- Replication key
- Serial or parallel processing

After you define the replicate and participants, you must manually start the replicate by running the **cdr start replicate** command.

The maximum number of replicates that you can define as participants on a particular replication server is 32767.

- Participant definitions
  You must define a participant for each server that is involved in the replicate definition by running the **cdr define replicate** command. Each participant in a replicate must specify a different database server.
- Replicate types
  You can choose a replicate type depending on whether you want the schema definitions on all participants to be the same. A master replicate enforces consistency between the schema definitions of the participants and the schema definition on a designated server. A classic replicate does not check the schema definitions of the participants.
- Defining Shadow Replicates
- Specifying Conflict Resolution Rules and Scope
  You specify the conflict resolution rule in the replicate definition.
- Specifying Replication Frequency
- Setting Up Failed Transaction Logging
  The Aborted Transaction Spooling (ATS) files and Row Information Spooling (RIS) files contain information about failed transactions and aborted rows. You can use this information to help you diagnose problems that arise during replication.
- Replicate only changed columns
  You can choose to replicate only those columns that have changes instead of entire rows.
- Using the IEEE Floating Point or Canonical Format
  You can specify how the FLOAT and SMALLFLOAT data types are handled, depending on your platform.
- Enabling Triggers
  By default, when a replicate causes an insert, update, or delete on a target table, triggers associated with the table are not executed. However, you can specify that triggers are executed when the replicate data is applied by enabling triggers in the replicate definition.
- Enabling code set conversion between replicates
  You can enable code set conversion to allow replication of data between servers that use different code sets.
- Replication to SPL routine

Participant definitions

You must define a participant for each server that is involved in the replicate definition by running the **cdr define replicate** command. Each participant in a replicate must specify a different database server.

Each participant definition includes the following information:

- Database server group name
- Database in which the table to be replicated resides
- Table name
- Table owner
- Participant type

For a primary-target replication system, you can specify the participant type as primary, receive-only, or send-only. If you do not specify the participant type, Enterprise Replication defines the participant as update-anywhere, by default.

- SELECT statement and optional WHERE clause

Restriction: Do not create more than one participant definition for each row and column to replicate. If the participant is the same, Enterprise Replication attempts to insert or update duplicate values during replication. For example, if one participant modifier includes WHERE x < 50 and another includes WHERE x < 100, Enterprise Replication sends the data for when x is between 50 and 100 twice.

- Defining Replicates on Table Hierarchies

Related concepts:
Primary-Target Replication System
Related reference:
Participant and participant modifier

Defining Replicates on Table Hierarchies

When you define replicates on inherited table hierarchies, use the following guidelines to replicate operations:

- For both the parent and child tables, define a replicate on each table.
- For only the parent table (not the child table), define a replicate on the parent table only.
- For only the child table (not the parent table), define a replicate on the child table only.

Replicate types

You can choose a replicate type depending on whether you want the schema definitions on all participants to be the same. A master replicate enforces consistency between the schema definitions of the participants and the schema definition on a designated server. A classic replicate does not check the schema definitions of the participants.

By default, replicates are master replicates. If you do not specify a master server, the master replicate is based on the first participant. Dictionary information is then stored about replicated column attributes for the participant you specify. Enterprise Replication checks for consistency between the master definition and local participant definitions. Checks are run when the replicate is defined and each time a new participant is added to the replicate, thus avoiding runtime errors. Verification also occurs each time that the master replicate is started on a server.

If you do not want to verify the schema, create a classic replicate. For example, if you want to create a data consolidation system in which one server only receives data from other servers that only send data, create a classic replicate.

Defining a replicate as a master replicate provides several advantages:

- Ensures data integrity by verifying that all participants in the replicate have table and replicated column attributes that match the master replicate definition.
- Provides automatic table generation on participants that do not already contain the table that is specified in the master replicate. However, Enterprise Replication cannot create tables with user-defined data types.
- Allows alter operations on the replicated tables.

When you define a master replicate, you can specify a participant that is on the server for which you are running the command. By default, the first participant that you list in the **cdr define replicate** command is the used to create the dictionary information for the master replicate. The additional participants in the **cdr define replicate** command are verified against the master definition and added to the replicate if they pass validation. If any participant fails validation, the **cdr define replicate** command fails and that participant is disabled.

- Master Replicate Verification
- Creating Strict Master Replicates
- Creating Empty Master Replicates

Related reference:
cdr define template
cdr define replicate

Master Replicate Verification

Enterprise Replication verifies the following information about a participant when the participant is added to the master replicate:

- The participant contains all replicated columns.
- The replicated columns in the participant have the correct data types. For columns that are user-defined data types, only the names of the data types are verified.
- Optionally, the replicated columns in the participant have the same column names as the master replicate.

Related tasks:
Creating Strict Master Replicates
Creating Empty Master Replicates

Creating Strict Master Replicates

You can create a strict master replicate in which all participants have the same replicated column names by using the **--name=y** option. This option specifies that when the master replicate verification is done for a new participant, that the column names on the participant must be identical to the column names of the master replicate. Strict master replicates allow you to perform the following tasks:

- Alter operations on replicated tables. For more information, see Alter, rename, or truncate operations during replication.
- Remastering by using the **cdr remaster** command. For more information, see Remastering a Replicate.

You can modify an existing master replicate to remove name verification by using the **--name=n** option of the **cdr modify replicate** command.

Related tasks:
Master Replicate Verification
Creating Empty Master Replicates
Related reference:
cdr modify replicate

Creating Empty Master Replicates

You can create an empty master replicate by using the **--empty** option. This option allows you to specify a participant as the basis of the master replicate but not include that participant in the replicate. Creating an empty replicate can be convenient in large environments in which you later add all participants using scripts.

When you define an empty master replicate, you must specify only one participant in the **cdr define replicate** command. This participant is used to create the master dictionary information but is not added to the replicate.

The **--empty** option is only supported for master replicates, you cannot use it without the **--master** option.

Related tasks:
Master Replicate Verification
Creating Strict Master Replicates

Defining Shadow Replicates

A *shadow replicate* is a copy of an existing, or primary, replicate. Enterprise Replication uses shadow replicates to manage alter and repair operations on replicated tables. You must create a shadow replicate to perform a manual remastering of a replicate that was defined with the **-n** option. See Resynchronize data manually for information about how you can repair, or remaster, your replicated data. After creating the shadow replicate, the next step in manual remastering is to switch the primary replicate and the shadow replicate using the **cdr swap shadow** command.

You create a shadow replicate using the **cdr define replicate** command with the **--mirrors** option, as described in cdr define replicate.

When you define a shadow replicate, its state is always set to the same state as the primary replicate. If you change the state of the primary replicate, all its shadow replicates' states are also changed to the same state.

You cannot delete a primary replicate if it has any shadow replicates defined. You must first delete the shadow replicates, and then the primary replicate.

You cannot modify a primary replicate (using the **cdr modify replicate** command) if it has any shadow replicates defined. Also, you cannot modify shadow replicates directly.

You cannot suspend or resume a primary replicate (using the **cdr suspend replicate** or **cdr resume replicate** command) if it has any shadow replicates defined. Also, you cannot suspend or resume shadow replicates directly. If the primary replicate and its shadow replicates are part of an exclusive replicate set, you can suspend or resume the entire replicate set using the **cdr suspend replicate** or **cdr resume replicate** command.

You cannot add a participant to a shadow replicate:

- If the participant is not part of the primary replicate's definition
- After remastering the replicate

If the primary replicate is part of an exclusive replicate set, any shadow replicates you define are automatically added to that replicate set.

If you add a primary replicate to an exclusive replicate set, all its shadow replicates are also automatically added. If you delete a primary replicate from an exclusive replicate set, all its shadow replicates are also automatically deleted.

Specifying Conflict Resolution Rules and Scope

You specify the conflict resolution rule in the replicate definition.

For update-anywhere replication systems, you must specify the conflict-resolution rules in the replicate definition using the **--conflict=***rule* option to the **cdr define replicate** command. The conflict resolution rule option names are:

- **always**
- **deletewins**
- **ignore**
- **timestamp**
- ***routine_name***
  If you use an SPL routine for your conflict-resolution rule, you can also use the **--optimize** option to specify that the routine is optimized.

You can also specify the scope using the **--scope=*scope*** option:

- **transaction** (default)
- **row**

Related concepts:
Update-Anywhere Replication System
Conflict resolution rule
Conflict Resolution Scope
Related reference:
cdr define replicate

Specifying Replication Frequency

The replication frequency options allow you to specify the interval between replications, or the time of day when an action should occur. If you do not specify the frequency, the default action is that replication always occurs immediately when data arrives.

The frequency options are:

- **--immed**
- **--every=*interval***
- **--at=*time***

For more information, see Frequency Options.
Important: If you use time-based replication and two tables have referential constraints, the replicates must belong to the same exclusive replicate set. For more information, see Exclusive Replicate Sets.

Setting Up Failed Transaction Logging

The Aborted Transaction Spooling (ATS) files and Row Information Spooling (RIS) files contain information about failed transactions and aborted rows. You can use this information to help you diagnose problems that arise during replication.

To configure your replicate to use ATS and RIS

1. Set up the ATS and RIS directories.
2. Specify the location of the ATS and RIS directories when you define your server.
3. Specify that the replicate use ATS and RIS when you define the replicate by including the **--ats** and **--ris** options in the replicate definition.

Tip: Until you become thoroughly familiar with the behavior of the replication system, select both ATS and RIS options.
Related concepts:
Monitor and troubleshooting Enterprise Replication
Related tasks:
Creating ATS and RIS directories
Defining Replication Servers
Related reference:
Replicate only changed columns
cdr define replicate

Replicate only changed columns

You can choose to replicate only those columns that have changes instead of entire rows.

By default, even if only one column changes, Enterprise Replication replicates the entire row, except columns that contain unchanged large objects.

You can change the default behavior to replicate only the columns that changed. To replicate only changed columns, include the **--fullrow=n** option in the replicate definition. Enterprise Replication always sends the replication key columns, even if you specify to replicate only changed columns.

Replicating only the columns that changed has the following advantages:

- Sends less data, because only the modified data is sent

- Uses less Enterprise Replication resources, such as memory

If Enterprise Replication replicates an entire row from the source, and the corresponding row does not exist on the target, Enterprise Replication applies the update as an insert, also known as an *upsert*, on the target (unless you are using the delete wins conflict resolution rule). By replicating the entire row, Enterprise Replication corrects any errors during replication. If any errors occur in an update of the target database server (for example, a large object is deleted before Enterprise Replication can send the data), the next update from the source database server (a complete row image) corrects the data on the target server.

Replicating only the columns that changed has the following disadvantages:

- Enterprise Replication does not apply upserts.
  If the row to replicate does not exist on the target, Enterprise Replication does not apply it. If you set up error logging, Enterprise Replication logs this information as a failed operation.
- You cannot use the SPL routine or time stamp with SPL routine conflict-resolution rules.
- You cannot use update-anywhere replication; doing so can result in inconsistent conflict resolution.

Enterprise Replication logs bitmap information about the updated columns in the logical-log file. For more information, see the CDR record type in the logical-logs chapter in the *IBM® Informix® Administrator's Reference*.

Related concepts:
Controlling the replication of large objects
Conflict Resolution
Disk Space for Delete Tables
Related tasks:
Setting Up Failed Transaction Logging
Related reference:
cdr define replicate

Using the IEEE Floating Point or Canonical Format

You can specify how the FLOAT and SMALLFLOAT data types are handled, depending on your platform.

You can specify sending this data in either IEEE floating point format or machine-independent decimal representation:

- Enable IEEE floating point format to send all floating point values in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating point format.
  To use IEEE floating point format, include the **--floatieee** option in your replicate definition.

  It is recommended that you define all new replicates with the **--floatieee** option.
- Enable canonical format to send floating-point values in a machine-independent decimal representation when you replicate data between dissimilar hardware platforms.
  To use canonical format, include the **--floatcanon** option in your replicate definition. The **--floatcanon** option is provided for backward compatibility only; it is recommended that you use the **--floatieee** option when defining new replicates.
- If you specify neither IEEE or canonical formats, Enterprise Replication sends FLOAT and SMALLFLOAT data types as a straight copy of machine representation. If you are replicating across different platforms, replicated floating-point numbers will be incorrect.

For more information, see Special Options.
Important: You cannot modify the replicate to change the **--floatieee** or **--floatcanon** options.
Related reference:
cdr define replicate

Enabling Triggers

By default, when a replicate causes an insert, update, or delete on a target table, triggers associated with the table are not executed. However, you can specify that triggers are executed when the replicate data is applied by enabling triggers in the replicate definition.

To enable triggers, include the **--firetrigger** option in your replicate definition.

When you design your triggers, you can use the **'cdrsession'** option of the **DBINFO()** function to determine if the transaction is a replicated transaction.

For information, refer to Triggers and Special Options.

Related reference:
cdr define replicate

Enabling code set conversion between replicates

You can enable code set conversion to allow replication of data between servers that use different code sets.

Prerequisites:
The table and column names must contain ASCII characters to convert a non-master replicate to a master replicate.

The servers must have UTF-8 code set transaction support enabled to replicate between server versions.

The target schema must allow for expansion due to code set conversion. For example, a CHAR(10) column in one code set might require 40 bytes in the converted code set.

When code set conversion is enabled, character columns of the following data types are converted to UTF-8 (Unicode) when the row is copied into the transmission queue.

- CHAR
- VARCHAR
- NCHAR
- NVARCHAR
- LVARCHAR
- TEXT
- CLOB

When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set that is used on the target server. No attempt is made to convert character data contained within opaque data types, such as **TimeSeries** data types, user-defined data types, or DataBlade module data types.

To enable code set conversion between replicates, include the **--UTF8=y** option in your replicate definition.

To use the latest version of the Unicode library, set the GL_USEGLU environment variable in your server environment. The GL_USEGLU environment variable must be set to a value of 1 (one) in the database server environment before the server is started, and before the database is created.

If your table names or column names contain non-ASCII characters, you must manually create a shadow replicate and then swap the shadow replicate with the primary replicate using the **cdr swap shadow** command.

The autocreate option is not supported for replicates defined with **--UTF8=y** option when using the **cdr realize template** or **cdr change replicate** commands.

Code set conversion with the GLS library requires only those code set conversion files found in the INFORMIXDIR/gls/cv9 directory.

- For US English, locales are handled automatically by the IBM® Informix® Client Software Development Kit installation and setup.
- For other locales, you might need to explicitly provide the locale and conversion files.

- Configuring code set conversion between replicates
  The examples in this topic show how to create replicate and template definitions while replicating data between databases that use different code sets.
- Code set conversion errors
  You can use the ATS and RIS files to identify problems that occur during code set conversion.
- Controlling the replication of large objects
  You can control whether columns that contain unchanged large objects are always included in replicated rows.

Related concepts:
Global language support for replication
Related reference:
cdr swap shadow
Related information:
GL_USEGLU environment variable

Configuring code set conversion between replicates

The examples in this topic show how to create replicate and template definitions while replicating data between databases that use different code sets.

When non-English characters are used for database, table, column, or owner names, each server must be added to the UTF-8 realize template definition by connecting to the server locally. Only one server at a time should be added to the replicate definition using the **change replicate** command. You cannot add multiple servers to a replication definition using the **define repl** command unless the database code set number is the same for all servers. The CLIENT_LOCALE environment variable must be set unless the database locale is en_us.819. Replicate and template names must be in English.

This example shows how to create and realize a template on two servers, named *node_1* and *node_2*. For this example, assume that *node_1* uses **de_de.819** locale and *node_2* uses **de_de.utf8** locale:

1. On node_1, run the following commands:

```
export DB_LOCALE=de_de.819
export CLIENT_LOCALE=de_de.819
cdr define template set1 -C always -M g_node1 -S row -d testdb -a -A -R --UTF8=y
cdr realize template set1 g_node1
```

2. On node_1 run the following command and wait for the **Txns in queue** count to go to zero.

```
onstat -g rqm cntrlq
```

3. On node_2, run the following commands:

```
export DB_LOCALE=de_de.utf8
export CLIENT_LOCALE=de_de.utf8
cdr realize template set1 g_node2
```

The following steps show how to define a replicate when non-ASCII characters are used for table, column, owner, or database names. Before starting, ensure that the replicate name uses English ASCII characters and that the DB_LOCALE environment variable on the server is set to the same value as the locale of the participant being added.

1. Define the replicate with the first participant and then connect to the participant.
2. Add and connect to each additional participant, one participant at a time.
3. When all of the participants have been added, ensure that the control queue is empty and start the replicate definition.
   You can check the control queue message count using the **onstat -g rqm cntrlq**. Wait for the **Txns in queue** count to go zero.

The following example shows how to create a replicate definition between two servers to replicate data between de_de.819 and de_de.utf8 databases:

1. On server node_1, run the following commands:

```
export DB_LOCALE=de_de.819
export CLIENT_LOCALE=de_de.819
cdr define repl german_repl  -M g_node1 -C always -S transaction
         -A -R -I --UTF8=y "testdb@g_node1:user1.table1" "select * from table1"
```

2. On node_1 run the following command and wait for the **Txns in queue** count to go to zero.

```
onstat -g rqm cntrlq
```

3. On node_2, run the following commands:

```
export DB_LOCALE=de_de.utf8
export CLIENT_LOCALE=de_de.utf8
cdr change repl -c node2 -a german_repl
           "testdb@g_node2:user1.table1" "select * from table1"
```

4. On node_2 run the following command and wait for the **Txns in queue** count to go to zero.

```
onstat -g rqm cntrlq
```

5. Run the following command on either server:

```
cdr start repl german_repl
```


Code set conversion errors

You can use the ATS and RIS files to identify problems that occur during code set conversion.

To specify which warnings and errors to suppress, use the CDR_SUPPRESS_ATSRISWARN configuration parameter. For more information, see CDR_SUPPRESS_ATSRISWARN Configuration Parameter

Each column in the RIS file begins with (W) if substitute characters were added to the column data or (E) if data was rejected because of a UTF-8 conversion failure.

Examples of conversion errors:

On the source server, a row of data fails conversion to UTF-8 code set.

Data sync error 63 is stored in an RIS file on the source server. The RIS file contains the row that failed to convert; the failed row is not converted and is not replicated on the target server. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:0 / Name:*UNKNOWN* / CommitTime:
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:35:22
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:63 (Error while converting data from local database codeset to
    UTF8.) / SQL:0 / ISAM:0
LRH  Failed column list: charcol (W), ncharcol (E)
LRD 3|Lkqy|jvdHj@ifcjuWg|biLs|uk|RwvCZOpfpqruLAA|JloY|<27, TEXT,
    PB 1 (utm_group_1) 1305051204 (11/05/10 13:13:24)>|<4, TEXT, BB>|
    <18, CLOB, SB 1305051204 (11/05/10 13:13:24)>
RRD ||||||||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

On the source server, conversion from the local code set to UTF-8 resulted in the substitution of one or more characters in the row.

Data sync error 65 is stored in an RIS file on the source server, and the row is replicated. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:0 / Name:*UNKNOWN* / CommitTime:
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:32:14
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:65 (Substitute characters added  while converting data from local
    database codeset to UTF8.) / SQL:0 / ISAM:0
LRH Failed column list: charcol (W), ncharcol (W), vchar (W), nvchar (W),
    lvchar (W)
LRD 2|iU\VoJMZ|axhGRxKmDW|e@Xv|biLs|pyqasjUpAc{wCu|efM@}Vd|<22, TEXT,
    PB 1 (utm_group_1) 1305051204 (11/05/10 13:13:24)>|<36, TEXT, BB>
    |<15, CLOB, SB 1305051204 (11/05/10 13:13:24)>
RRD ||||||||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

On the target server, a row of data failed to convert from UTF-8 format to the local database code set.

Data sync error 64 is stored in an ATS/RIS file on the target server, and the row or transaction is aborted depending on the replicate scope. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:1 / Name:utm_group_1 / CommitTime:11-05-10 13:40:19
TXH Target ID:3 / Name:utm_group_3 / ReceiveTime:11-05-10 13:40:19
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:64 (Error while converting data from UTF8 to local database
    codeset.) / SQL:0 / ISAM:0
RRH  Failed column list: vchar (E)
RRD 3||jdicW|?|?|?|?|?|?
==========
TXH Transaction aborted
TXH ATS file:/usr4/nagaraju/utm/tmp/ats.utm_group_3.utm_group_1.D_3
    .110510_13:40:19.2 has also been created for this transaction
```

On the target server, conversion from UTF-8 to the local server code set resulted in the substitution of one or more characters in the row.

Data sync error 66 is stored in a warning RIS file on the target server, and the row is applied. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:3 / Name:utm_group_3 / CommitTime:11-05-10 13:13:58
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:13:58
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Insert
RRH CDR:66 (Substitute characters added while converting data
    from UTF8 to local database codeset.) / SQL:0 / ISAM:0
RRH  Failed column list: charcol (W), ncharcol (W), vchar (W),
    nvchar (W), lvchar (W), textcol (W), textbcol (W), clobcol (W)
RRD 99||keI||m||<46, TEXT, PB 3 (utm_group_3) 1305051238
    (11/05/10 13:13:58)>|<68, TEXT, BB>|<13, CLOB, SB>
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

Text and CLOB data conversion failures

If the conversion of text or CLOB data to UTF-8 fails on the source server then the blob buffer is marked with the appropriate error and the target servers create ATS/RIS files for these blob data conversion failures. Example text column conversion error:

```
TXH Source ID:1 / Name:utm_group_1 / CommitTime:11-05-10 12:26:30
TXH Target ID:3 / Name:utm_group_3 / ReceiveTime:11-05-10 12:28:15
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:65 (Substitute characters added  while converting data from local
    database codeset to UTF8.) / SQL:0 / ISAM:0
RRH  Failed column list: textcol (W)
LRD 2|<46, TEXT, PB 1 (utm_group_1) 1305048215 (11/05/10 12:23:35)
    >|<40, CLOB, SB 1305048215 (11/05/10 12:23:35)>
RRD 2|<44, TEXT, PB 1 (utm_group_1) 1305048390 (11/05/10
    12:26:30)>|<0(NoChange), CLOB, SB>
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```


Controlling the replication of large objects

You can control whether columns that contain unchanged large objects are always included in replicated rows.

By default, columns that contain unchanged large objects are not included in replicated rows. Large object columns are transmitted only when the data is changed.

You can specify to replicate columns that contain unchanged large objects by including the **--alwaysRepLOBS=y** option in the replicate definition. For example, if your replication system is designed for as a workflow, you must replicate all columns when you move data to the next site in the workflow.

If you want to change how large objects are replicated for an existing replicate, you must delete the replicate and then re-create the replicate.

Related concepts:
Workflow Replication
Related reference:
Replicate only changed columns

Replication to SPL routine

At target participant, ‘replication to SPL routine’ type replicate definition causes SPL routine to be executed instead of applying data to target table. Target participant for “replication to SPL routine” replicate definition can be configured to be same as source database, different database on the same server, or remote peer Enterprise Replication server. “Replication to SPL routine” replicate definition does not enforce the requirement to have primary key, unique index or ER key on the replicated table. Note: Even though data is applied to stored procedure routine, target table definition must exist.

```
|--+--splname=spl_routine_name----+-------+---------------------+--------->
   |                              |       |               .-y-. |
'--'--jsonsplname=spl_routine_name-'       '- -- cascaderepl=-+-n-+-'
```

Long Form	Meaning
--splname	Stored procedure routine name to apply data to. SPL routine must exist at all participants. Column list for SPL routine extracted from replicate participant select statement column projection list.
--jsonsplname	Stored procedure routine name to apply data to. SPL routine and table definition must exist at all participants. Input argument for SPL routine must be a JSON document. --jsonsplname option is mutually exclusive to --splname option.
--cascaderepl	Enable cascade replication. Required if replication to SPL needs to be executed for the data applied through Enterprise Replication.

--splname option stored procedure argument list:

- Optype char(1) – operation type. Values include
  - I – Insert
  - U – Update
  - D – Delete
- Soucre_id integer – Source server id. Same as group id.
- Committime integer – Transaction commit time.
- Txnid bigint – Transaction id.
- userid - Userid of the user executing the IUD operation.
- session_id - Session id of the session executing the IUD operation.
- Before value column list.
- After value column list.

Note: Column list for SPL routine extracted from select statement projection list

--jsonsplname option SPL routine json argument

Attribute Name	Description
operation	Operation type: Insert/Delete/Update
table	Table name
owner	Table owner
database	Database name
txnid	8 byte unique id. Higher order 4 bytes: commit work log id, lower order 4 bytes: commit work log position.
operation_owner_id	User id of the user executing the IUD operation
operation_session_id	Session id of the session executing the IUD operation
commit_time	Transaction commit time for the event data.
rowdata	Row data in JSON document format. Data is returned in column name as key and column data as value.
before_rowdata	Before row data for “update” operation.

Example: JSON Document format

```
{“operation”:"insert",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”txnid”:2250573177224,
”operation_owner_id”:201,”operation_session_id”:200,”commit_time”:1488243530,rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",
”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}

{“opertion”:"update",table:"creditcardtxns",”owner”:"informix",”database”:"creditdb”,”txnid”:2250573308360,
”operation_owner_id”:201,”operation_session_id”:202,”commit_time”:1488243832,”rowdata”:{uid:21,cardid:"7777-7777-7777-7777",
”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":"25-Jan-2017 16:15"} },
”before_rowdata”:{“uid”:21,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-
01T10:35:10.000Z  } }}

{“opertion”:"delete",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”txnid”:2250573287760,
”operation_owner_id”:201,”operation_session_id”:203,”commit_time”:1488243797,rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",
”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T13:35:06.000Z } }}
```

Asynchronous post commit trigger support

- Define loopback replication server. For more information, see Loopback Replication
- Create ‘replication to SPL’ type replicate with same “database and table” information for both source and target participants. Loopback server group name shall be specified with target participant definition

Example

```
cdr define replicate txn_repl -C always -S row -M group_1 -A -R --splname=logger4repl2spl
"stores@group_1:informix.transactions" "select * from transactions" "test@lb_group_1:informix.transactions" "select * from
transactions”
```

Note: group_1 is the local server ER group, and lb_group_1 is the pseudo ER server group for loop back replication.

Prerequisites

- Enable Login for the tables.
- Ensure not to combine participant definitions that include table as the target, and SPL routine as the target.
- Define target table.
- Out-of-row data datatypes, TEXT, BYTE, BLOB, CLOB are not supported.

Example 1: build staging table for data changes using --splname replicate attribute

```
create database storesdb with log;

create table transaction (uid int, bill_amount int);

create table changelog (optype int, srcdir int, committime int, txnid bigint, userid int, session_id int, uid_bef int,
 bill_amount_bef int, uid int, bill_amount int);

create procedure logger4repl2spl (opType char(1), srcid int, committime int, txnid bigint, userid int, session_id int,
uid_bef int, bill_amount_bef int, uid int, bill_amount_bef int)

insert into changelog values (opType, srcid, committime, txnid, userid, session_id, uid_bef, bill_amount_bef, uid,
bill_amount);

end procedure;
```

Note: Require new code section for these shell commands:

```
$ cdr define replicate txn_repl -C always -S row -M group_1 -A -R --splname=logger4repl2spl
"storesdb@group_1:informix.transaction" "select * from transaction"
"storesdb@lb_group_1:informix.transaction" "select * from transaction”

$ cdr start replicate txn_repl
```

Example 2: build staging table for data changes using --jsonsplname replicate attribute

```
create database storesdb with log;

create table transaction (uid int, bill_amount int);

create table inventory (inv_id int, inv_count int);

create table staging (data json);

create procedure logger4repl2spl (data json)
   insert into staging values (data);
end procedure;

$ cdr define replicate inv_repl -C always -S row -M group_1 -A -R --jsonsplname=logger4repl2spl
"storesdb@group1:informix.inventory" "select * from inventory" "storesdb@lb_group_1:informix.inventory" "select * from
inventory”

$ cdr start replicate inv_repl

$ cdr define replicate txn_repl -C always -S row -M group_1 -A -R --jsonsplname=logger4repl2spl
"storesdb@group_1:informix.transaction" "select * from transaction"
"storesdb@lb_group_1:informix.transaction" "select * from transaction”

$ cdr start replicate txn_repl
```

Example 3: Realtime aggregation framework

```
create database retaildb with log;

create table sales (customerid int, storeid int, bill_amount float);

create table sales_summary(storeid int , s_count int, s_sum float, s_avg float, s_min float, s_max float);

create procedure store_agg(opType char(1), srcid int, committime int, txnid bigint, userid int, session_id int,
customerid_bef int, storeid_bef int, bill_amount_bef float, customerid int, storeid_aft int, bill_amount float)
---- -----
----  ----
end procedure;

$ cdr define replicate sale_repl -C always -S row -M group_1 -A -R --serial --splname=store_agg
"retaildb@group_1:informix.sales" "select * from sales"
"retaildb@lb_group_1:informix.sales" "select * from sales"

$ cdr start replicate sale_repl
```

Note: For streaming aggregation, make sure to define replicate with –serial option to avoid executing multiple instances of the same SPL routine in parallel.

Example 4: Calculate leader board on the fly based on changes to scores table

```
create table scores (playerid int, score int);
create table leaderboard(playerid int, score int);

create procedure leaderboard_spl(opType char(1), srcid int, committime int, txnid bigint, userid int, session_id int,
playerid_bef int, score_bef int, playerid_aft int, score_aft int)
… ….
end procedure;

$ cdr define replicate game_repl -C always -S row -M group_1 -A -R --serial --splname=leaderboard_spl
```

```
"gamedb@group_1:informix.scores" "select * from scores" "gamedb@lb_group_1:informix.scores" "select * from scores"

$ cdr start replicate game_repl
```

Define smart trigger on leaderboard table(mentioned in the example given above) and push out changes to application layer.

Note: For more information, see Smart Trigger.

Define replicate sets

When you define a replicate set, you specify the type of replicate set, the replicates that belong to the replicate set, and the frequency of replication for the member replicates.

To create a replicate set, use the **cdr define replicateset** command.

Enterprise Replication supports these types of replicate sets:

exclusive
: Replicates can belong to only one replicate set. Include the **--exclusive** option in the **cdr define replicateset** command.

non-exclusive
: Default. Replicates can belong to one or more non-exclusive replicate sets.

derived
: A replicate set that is derived from an existing replicate set. For example, you can create a derived replicate set that contains replicates that must be remastered.

- Exclusive Replicate Sets
  If your replicated tables use referential integrity and are defined with time-based replication, you must create an *exclusive replicate set*. If your replicates use referential integrity and you plan to stop and start the replicate set, use an exclusive replicate set.
- Non-Exclusive Replicate Sets
- Customizing the Replicate Set Definition

Related reference:
cdr define replicateset

Exclusive Replicate Sets

If your replicated tables use referential integrity and are defined with time-based replication, you must create an *exclusive replicate set*. If your replicates use referential integrity and you plan to stop and start the replicate set, use an exclusive replicate set.

An exclusive replicate set has the following characteristics:

- All replicates in an exclusive replicate set have the same state and frequency settings. For more information, see cdr list replicateset.
- When you create the replicate set, Enterprise Replication sets the initial state of the replicate set to active.
- You can manage the replicates in an exclusive replicate set only as part of the set. Enterprise Replication does not support the following actions for the individual replicates in an exclusive replicate set:
  - Starting a Replicate
  - Stopping a Replicate
  - Suspending a Replicate
  - Resuming a Suspended Replicate
- Replicates that belong to an exclusive replicate set cannot belong to any other replicate sets.

To create an exclusive replicate set, use the **--exclusive** option with **cdr define replicateset**.
Important: You cannot change an exclusive replicate set to non-exclusive.
Related tasks:
Non-Exclusive Replicate Sets
Customizing the Replicate Set Definition
Related reference:
cdr define replicateset
cdr define template
cdr resume replicate

Non-Exclusive Replicate Sets

By default, the **cdr define replicateset** command creates *non-exclusive* replicate sets.

A non-exclusive replicate set has the following characteristics:

- You can manage replicates that belong to a non-exclusive replicate set both individually and as part of the set.
- Because individual replicates in a non-exclusive replicate set can have different states, the non-exclusive replicate set itself has no state.
- You should not use non-exclusive replicate sets for replicates that include tables that have referential constraints placed on columns.
- A replicate can belong to more than one non-exclusive replicate set.

Important: You cannot change a non-exclusive replicate set to exclusive.
Use non-exclusive replicate sets if you want to add a replicate to more than one replicate set. For example, you might want to create replicate sets to manage replicates on the target server, table, or entire database. To do this, create three non-exclusive replicate sets:

- A set that contains the replicates that replicate to the target server
- A set that contains the replicates on a particular table
- A set that contains all the replicates

In this scenario, each replicate belongs to three non-exclusive replicate sets.

Related tasks:
Exclusive Replicate Sets
Customizing the Replicate Set Definition

Customizing the Replicate Set Definition

You can specify the replication frequency (Specifying Replication Frequency) for all the replicates when you define the replicate set. For example, to define the non-exclusive replicate set **sales_set** with the replicates **sales_fiji** and **sales_tahiti** and specify that the members of **sales_set** replicate at 4:00 a.m. every day, enter:

```
cdr define replicateset --at 4:00 sales_set sales_fiji \
   sales_tahiti
```

To define the exclusive replicate set **dev_set** with the replicates **dev_pdx** and **dev_lenexa** and specify that the members of **dev_set** replicate at 5:00 p.m. every day, enter:

```
cdr define replicateset -X --at 17:00 dev_set dev_pdx\
   dev_lenexa
```

Important: For replicates that belong to an exclusive replicate set, you cannot specify the frequency individually for replicates in the set.
For more information, see cdr define replicateset.

Related tasks:
Exclusive Replicate Sets
Non-Exclusive Replicate Sets

Initially Synchronizing Data Among Database Servers

Enterprise Replication provides an initial synchronization feature that allows you to easily bring a new table up-to-date with replication when you start a new replicate, or when you add a new participant to an existing replicate.

You do not need to suspend any servers that are replicating data while you add the new replicate and synchronize it.

The **cdr start replicate** and **cdr start replicateset** commands provide options to perform an initial synchronization for the replicates you are starting. All of the rows that match the replication criteria will be transferred from the source server to the target servers. If you are starting a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables).

Use the **--syncdatasource** (**-S**) option of the **cdr start replicate** or **cdr start replicateset** command to specify the source server for synchronization. Any existing rows in the specified replicates are deleted from the remote tables and replaced by the data from the node you specify using **-S**.

The **--extratargetrows** option of the **cdr start replicate** or **cdr start replicateset** commands specifies how to handle rows found on the target servers that are not present on the source server. You can specify to remove rows from the target, keep extra rows on the target, or replicate extra rows from the target to other participants.

If you use the **cdr start replicate** or **cdr start replicateset** command to specify a subset of servers on which to start the replicate (or replicate set), that replicate (or replicate set) must already be active on the source server. The source server is the server you specify with the **-S** option. For example, for the following command, **repl1** must already be active on **serv1**:

```
cdr start repl repl1 ... -S serv1 serv2 serv3
```

When you start a replicate (or replicate set) for participants on all servers, the replicate does not need to be active on the source server. So, for the following command, **repl1** does not need to be active:

```
cdr start repl1 ... -S serv1
```

When Enterprise Replication performs initial data synchronization, it keeps track of discrepancies between the constraints set up on source and target server tables. Rows that fail to be repaired due to these discrepancies are recorded in the ATS and RIS files.

If replication fails for some reason and data becomes inconsistent, there are different ways to correct data mismatches between replicated tables while replication is active. The recommended method is direct synchronization. You can also repair data based on an ATS or RIS file. Both of these methods are described in Resynchronizing Data among Replication Servers.

Related concepts:
Repair and Initial Data Synchronization
Load and unload data

Set up replication through templates

Enterprise Replication provides templates to allow easy setup and deployment of replication for clients with large numbers of tables to replicate. A template uses schema information about a database, a group of tables, columns, and replication keys to define a group of master replicates and a replicate set.

Do not use a template if you want to use time-based replication.

You create a template by running the **cdr define template** command and then you instantiate the template on the servers where you want to replicate data by running the **cdr realize template** command.

Templates set up replication for all the columns in the table. Templates are useful for setting up large-scale replication environments. If you want a participant to contain a partial row (just some columns in the table), you can either set up replication manually, or, after you realize a template you can run the **cdr remaster** command to restrict the query.

- Defining Templates
- Realizing Templates

Defining Templates

You define a template using the **cdr define template** command, with which you can specify which tables to use, the database and server they are located in, and whether to create an exclusive or non-exclusive replicate set. Table names can be listed on the command line or accessed from a file using the **--file** option, or all tables in a database can be selected.

Important: A template cannot define tables from more than one database.
Specify that the replicate set is exclusive if you have referential constraints on the replicated columns. Also, if you create an exclusive replicate set using a template, you do not need to stop the replicate set to add replicates. For more information about exclusive replicate sets, see Define replicate sets.

A template defines a group of master replicates and a replicate set.

You can use the **cdr list template** command from a non-leaf node to view details about the template, including the internally generated names of the master replicates. These are unique names based on the template, the server, and table names.

Realizing Templates

After you define a template using the **cdr define template** command, use the **cdr realize template** command to instantiate the template on your Enterprise Replication database servers. The **cdr realize template** command first verifies that the tables on each node match the master definition used to create the template. Then, on each node, it adds the tables defined in the template as participants to master replicates created by the template.

If a table on a server has additional columns to those defined in the template, those columns are not considered part of the replicate.

If a table does not already exist on a server where you realize the template, you can choose to create it, and it is also added to the replicate.

Also, at realization time, you can also choose to synchronize data among all servers.

- Verifying Participants without Applying the Template
- Synchronizing Data Among Database Servers
- Create tables automatically
  You automatically create tables in the template definition if they do not exist on a server.
- Other synchronization options
  Several other options to the **cdr realize template** command can affect how synchronization occurs.
- Changing Templates
- Template Example
  This example illustrates a scenario in which one template is created and realized on two servers, then realized on a third server.

Verifying Participants without Applying the Template

The **--verify** option allows you to check that a template's schema information is correct on all servers before actually instantiating the template.

Synchronizing Data Among Database Servers

Use the **--syncdatasource** option to specify a server to act as the source for data synchronization on all servers where you are realizing the template. The server listed with this option must either be listed as one of the servers on which to realize the template, or it must already have the template.

- Improve Performance During Synchronization
  You can speed up a synchronization operation by temporarily increasing the size of the send queue.

Improve Performance During Synchronization

You can speed up a synchronization operation by temporarily increasing the size of the send queue.

Enterprise Replication uses the value of the CDR_QUEUEMEM configuration parameter as the size of the send queue during a synchronization operation. To increase the size of the send queue during a particular synchronization operation, use the **--memadjust** option.

In addition to controlling memory during initial synchronization, you can also control memory consumption when you realize a template and perform a direct synchronization.

Create tables automatically

You automatically create tables in the template definition if they do not exist on a server.

Include the **--autocreate** option in the **cdr realize template** command to automatically create tables. You cannot use the **--autocreate** option for tables that contain user-defined data types.

Use the **--dbspace** option to specify a dbspace for table creation.

Note: Tables that are created by **--autocreate** option do not automatically include non-replicate key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. You must manually create these objects.

Other synchronization options

Several other options to the **cdr realize template** command can affect how synchronization occurs.

You can use the **--applyasowner** option to realize a table by its owner rather than the user **informix**.

The **--extratargetrows** option specifies whether to delete, keep, or merge rows found on target servers that are not present on the source server during the synchronization operation.

The **--mode** option defines whether servers only receive or only send data.

Changing Templates

You cannot update a template. To adjust a template, you must delete it with the **cdr delete template** command and then re-create it with the **cdr define template** command.

Template Example

This example illustrates a scenario in which one template is created and realized on two servers, then realized on a third server.

The template **Replicateset1** is defined on three tables in the **college** database: **staff**, **students**, and **schedule**. The template is first realized on the servers **g_cdr_ol_1** and **g_cdr_ol_2**.

This procedure is performed as follows:

1. Define the template **Replicateset1** on the **staff**, **students**, and **schedule** tables of the **college** database:

```
cdr define template -c g_cdr_ol_1 Replicateset1 -M g_cdr_ol_1\
 -C "timestamp" -A -R -d college testadm.staff testadm.students\
 testadm.schedule
```

This command also creates the replicate set **Replicateset1**.

2. Realize the template on the server **g_cdr_ol_1**:

```
cdr realize template -c g_cdr_ol_1 Replicateset1 "college@g_cdr_ol_1"
```

3. Realize the template on server **g_cdr_ol_2** and synchronize the data with server **g_cdr_ol_1**:

```
cdr realize template -c g_cdr_ol_2 -u -S g_cdr_ol_1 \
Replicateset1 "college@g_cdr_ol_2"
```

4. Realize the **Replicateset1** template on a new server **g_cdr_ol_3** and synchronize the data with server **g_cdr_ol_1**. The **g_cdr_ol_3** server participant is automatically added to all replicates within the **Replicateset1** template:

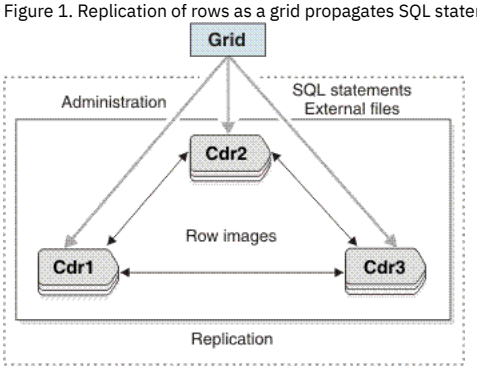
```
cdr realize template -c g_cdr_ol_1 -u -S g_cdr_ol_1\
  Replicateset1 "g_cdr_ol_3"
```


Grid setup and management

A grid is a set of replication servers that are configured to simplify administration. When you run SQL data definition statements from within a grid context on a grid server, the statements propagate to all servers in the grid. You can run SQL data manipulation statements and routines through grid routines. You can choose to set up replication automatically when you create a table through a grid. You can propagate external files to other servers in the grid.

SQL statements are not replicated by Enterprise Replication. Enterprise Replication replicates the row images that are the results from SQL statements. The grid propagates SQL statements, but does not, by default, propagate the results of propagated SQL statements. The following illustration shows three replication servers, named **Cdr1**, **Cdr2**, and **Cdr3**, that replicate row images between each other, while the grid propagates SQL statements and administration commands.

Figure 1. Replication of rows as a grid propagates SQL statements to each server.



A grid can be useful if you have multiple replication servers and you often perform the same tasks on every replication server. The following types of tasks can be run through the grid:

- Creating replicated tables. When you create a replicated table through a grid, the other tasks for setting up replication are completed automatically: a replicate is created for the table, participants are defined for each replication server, and the replicate is added to the grid replicate set.
- Administering servers, for example, adding chunks, removing logical logs, or changing configuration parameter settings
- Updating the database schema, for example, altering, adding, or removing tables
- Running or creating stored procedures or user-defined routines
- Updating data, for example, purging old data or updating values that are based on conditions
- Altering a replicate definition when you alter a replicated table
- Copying external files to grid servers

For example, suppose that you have 100 replication servers and must create a table. You must fragment the table into two new dbspaces. You also must create a new stored procedure to run on the table. With a grid, you would run four commands to perform these tasks on all 100 replication servers, instead of running 400 commands. The command to create the table can also specify that the data in that table is replicated.

You can control the security of the grid by authorizing which users can run grid routines on which servers. You can monitor the results of grid routines and rerun any failed routines on the appropriate servers.

You can configure Connection Managers to route client connection requests to the replication servers of a grid, based on one of the following redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.
- ROUNDROBIN: Connection requests are directed in a repeating, ordered fashion (round-robin) to a group of replication servers.
- WORKLOAD: Connection requests are directed to the replication server that has the lowest workload.

- Example of setting up a replication system with a grid
  This comprehensive example sets up a replication domain, creating a grid, creating a database, creating a replicated table, and loading data.
- Example of rolling out schema changes in a grid
  You can roll out schema changes to replicated tables through a grid without shutting down your applications.
- Creating a grid
  You can create a grid based on an existing replication domain. You must authorize users who can run grid routines, and designate a server from which to run grid routines.
- Grid maintenance
  You can adjust grid membership, change user or server authorization to run grid routines, and delete grid-routine history from the **syscdr** database.
- Adding replication servers to a grid
  There are multiple ways to add a replication server to a grid.

- Adding an externally created replicate into a grid replicate set
  If a replicate is created external to a grid, it can still be added to a grid replicate set.
- Creating replicated tables through a grid
  You can automatically create a replicate and start replication when you create a table through the grid.
- Enabling replication within a grid transaction
  You can enable replication within a transaction that is run in the context of the grid.
- Propagating updates to data
  You can change your data through a grid routine and propagate the changes to all the servers in the grid.
- Administering servers in the grid with the SQL administration API
  You can run SQL administration API commands in grid routines to perform administrative tasks on all servers in the grid.
- Propagating database object changes
  You can create or alter database objects by running DDL statements while connected to the grid and propagate the changes to all the servers in the grid.
- Propagating external files through a grid
  You can copy non-database, external files to the servers within a grid.
- Rerunning failed grid routines
  You can rerun a grid routine that failed on one or more servers in the grid.
- Connection management for client connections to participants in a grid
  You can configure Connection Managers to route connection requests from clients to the replication servers of a grid.
- Grid queries
  If you have a table that is the same on multiple servers in a grid, but whose data is not replicated, you can run a grid query to return the consolidated data from the multiple servers.

Related concepts:
Preparing the Replication Environment
Using High-Availability Clusters with Enterprise Replication
Shard cluster setup
Managing Replication Servers and Replicates
Monitor and troubleshooting Enterprise Replication
Related tasks:
Defining Replication Servers, Replicates, Participants, and Replicate Sets
Related information:
Connection management through the Connection Manager

Example of setting up a replication system with a grid

This comprehensive example sets up a replication domain, creating a grid, creating a database, creating a replicated table, and loading data.

This example creates a replication domain and grid that contain four replication servers: **serv1**, **serv2**, **serv3**, **serv4**. Each server computer has the Informix® database server installed, but no databases defined.

1. On all servers, set the CDR_QDATA_SBSPACE configuration parameter.
2. Edit the sqlhosts files on all four servers so that they each have the following information:

```
#dbservername   nettype   hostname              servicename   options
gserv1          group     -                     -             i=143
serv1           ontlitcp  ny.usa.com            1230          g=gserv1
gserv2          group     -                     -             i=144
serv2           ontlitcp  tokyo.japan.com       1231          g=gserv2
gserv3          group     -                     -             i=145
serv3           ontlitcp  rome.italy.com        1232          g=gserv3
gserv4          group     -                     -             i=146
serv4           ontlitcp  perth.australia.com   1233          g=gserv4
```

3. Define each server as a replication server by running the **cdr define server** command:

```
cdr define server -c gserv1 -I gserv1
cdr define server -c gserv2 -S gserv1 -I gserv2
cdr define server -c gserv3 -S gserv1 -I gserv3
cdr define server -c gserv4 -S gserv1 -I gserv4
```

4. Create a grid that includes all replication servers in the domain as members of the grid:

```
cdr define grid grid1 --all
```

5. Authorize the user **bill** to run commands on the grid and designate the server **gserv1** as the source server from which grid commands can be run:

```
cdr enable grid --grid=grid1 --user=bill --node=gserv1
```

   Tip: User **informix** does not have permission to run grid operations unless you include it in the user list.
6. Run **cdr list grid** to see the grid configuration:

```
Grid                Node                User
------------------- ------------------- -----------------
grid1               gserv1*             bill
                    gserv2
                    gserv3
                    gserv4
```

   The asterisk indicates that **gserv1** is the source server for the grid.
7. Run the **cdr list replicateset** command to see the grid replicate set information:

```
Ex T REPLSET                   PARTICIPANTS
------------------------------------------------
Y  Y  grid1
```

The replicate set has the same name as the grid. It does not yet contain any participants.

8. Create two dbspaces named **dbsp2** and **dbsp3** in which to fragment a table:

```
database sysmaster;

EXECUTE FUNCTION ifx_grid_function('grid1',
                 'task("create dbspace","dbsp2",
           "/db/chunks/dbsp2","2G","0")');

EXECUTE FUNCTION ifx_grid_function('grid1',
                 'task("create dbspace","dbsp3",
           "/db/chunks/dbsp3","8G","0")');
```

The dbspaces are created on all four servers.

9. Create database named **retail** and a table named **special_offers** with replication enabled:

```
database sysmaster;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE DATABASE retail WITH LOG;

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar,
    offer_type        char(16))
    WITH CRCOLS
FRAGMENT BY EXPRESSION
    offer_type = "GOLD" IN dbsp2,
    REMAINDER IN dbsp3;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

10. Run the **cdr list grid --verbose grid1** command to see information about the statements on each server:

```
Grid                Node                User
------------------- ------------------- -----------------
grid1               gserv1*             bill
                    gserv2
                    gserv3
                    gserv4
Details for grid grid1

Node:gserv1 Stmtid:1 User:bill Database:retail 2010-05-27 15:21:57
CREATE DATABASE retail WITH LOG;
ACK gserv1 2010-05-27 15:21:57
ACK gserv2 2010-05-27 15:21:58
ACK gserv3 2010-05-27 15:21:59
ACK gserv4 2010-05-27 15:21:59

Node:gserv1 Stmtid:1 User:bill Database:retail 2010-05-27 15:21:57
CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar
    offer_type        char(16))
    WITH CRCOLS
FRAGMENT BY EXPRESSION
    offer_type = "GOLD" IN dbsp2
    REMAINDER IN dbsp3;
ACK gserv1 2010-05-27 15:21:57
ACK gserv2 2010-05-27 15:21:58
ACK gserv3 2010-05-27 15:21:59
ACK gserv4 2010-05-27 15:21:59
```

Both statements succeeded on all four servers.

11. Run **cdr list replicate** to see the replicate information:

```
CURRENTLY DEFINED REPLICATES
------------------------------------------
REPLICATE:        gserv1_1
STATE:            Active
CONFLICT:         Timestamp
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      retail:bill.special_offers
OPTIONS:
REPLTYPE:         Master,Grid
```

The replicate was created and is active.

12. Run the **cdr list replicate brief gserv1_1** command to see the participants:

```
REPLICATE  TABLE                              SELECT
-----------------------------------------------------------------
gserv1_1   retail@gserv1:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
```

```
gserv1_1    retail@gserv2:bill.special_offers  select * from
                                                bill.special_offers
gserv1_1    retail@gserv2:bill.special_offers  select * from
                                                bill.special_offers
```

13. Load data onto one of the replication servers and Enterprise Replication replicates the data to the other servers. For more information, see Load and unload data.

Related concepts:
Connection management for client connections to participants in a grid
Related tasks:
Adding a replication server to a grid by cloning
Related reference:
cdr enable grid
cdr list grid
cdr list replicateset
Related information:
sqlhosts connectivity information

Example of rolling out schema changes in a grid

You can roll out schema changes to replicated tables through a grid without shutting down your applications.

Suppose that you have a grid replicate set named **gridset** that contains 12 replicates, each of which represents a different table. You want to alter the data types of columns in five tables. The grid contains four servers.

To roll out schema changes without application downtime:

1. Change any connections from the original application to the replication server named **cdr1** to connect to the replication server named **cdr2**.
2. On the **cdr1** server, connect to the **stores_demo** database, connect to the grid, and alter the five tables:

```
dbaccess stores_demo -
EXECUTE PROCEDURE ifx_grid_connect('grid1', 'gridset', 4);
SET LOCK MODE TO WAIT 120;
ALTER TABLE customer ADD prefix (char15);
ALTER TABLE items MODIFY order_num (bigint);
ALTER TABLE stock MODIFY description (lvarchar);
ALTER TABLE cust_calls ADD call_descr2 (lvarchar);
ALTER TABLE manufact MODIFY manu_name (char32);
```

   The **ifx_grid_connect()** procedure changes the tables on **cdr1** but delays the propagation of the changes to the other replication servers.

3. Update the application to reflect the new schema for the five tables and connect to the server **cdr1**.
4. Close the connections from the original application.
5. On the server **cdr1**, propagate schema changes to the other replication servers by running the following statement:

```
EXECUTE FUNCTION ifx_grid_release('grid1', 'gridset');
```

6. On the server **cdr1**, create a derived replicate set named **alterSet** that contains the altered tables by running the following command:

```
cdr define replicateset --needRemaster=gridset alterSet
```

7. From the server **cdr1**, remaster the altered tables on all replication servers by running the following command:

```
cdr remaster replicateset --master=cdr1 alterSet
```

8. From the server **cdr1**, synchronize the data on all replication servers by running the following command:

```
cdr check replicateset --replset=alterSet --repair --master=cdr1 --all
```

9. On the server **cdr1**, drop the derived replicate set by running the following command:

```
cdr delete replicateset alterSet
```

Related tasks:
Altering multiple tables in a replicate set
Propagating database object changes

Creating a grid

You can create a grid based on an existing replication domain. You must authorize users who can run grid routines, and designate a server from which to run grid routines.

You must be connected to a replication server in the domain that contains the servers that you want to include in the grid.
To create a grid:

1. Specify a name for the grid and the servers to include in the grid by running the **cdr define grid** command. For example, the following command creates a grid named **grid1** and adds all replication servers in the domain as members of the grid:

```
cdr define grid grid1 --all
```

2. Authorize users to run commands on the grid and designate a server from which grid commands can be run by running the **cdr enable grid** command. For example, the following command authorizes the user **bill** to run commands on the server **gserv1**:

   ```
   cdr enable grid --grid=grid1 --user=bill --node=gserv1
   ```

   Only authorized users can run grid routines on authorized servers. User **informix** does not have permission to perform grid operations unless you include it in the user list.

Related reference:
cdr define grid
cdr enable grid

Grid maintenance

You can adjust grid membership, change user or server authorization to run grid routines, and delete grid-routine history from the **syscdr** database.

To see information about the grid, such as, which servers can run grid routines and the status of routines that are run on the grid servers, run the **cdr list grid** command.

If you remove a server from your replication domain, remove the server from your grid. The following example removes a replication server named **gserv1** from the grid **grid_1**:

```
cdr change grid grid_1 --delete gserv1
```

You cannot drop a replicated column through a grid. To drop a replicated column, you must manually remaster the replicate and then drop the column.

You cannot rename a replicated database. You must manually rename the database on each participant server by using the **cdr remaster** command.

To change which users can run routines on the grid or which servers are authorized to run grid routines, run the **cdr enable grid** and **cdr disable grid** commands. For example, to change the authorized server from **gserv1** to **gserv2** and authorize the user **srini**, run the following commands:

```
cdr disable grid --grid=grid1 --node=gserv1
cdr enable grid --grid=grid1 --node=gserv2 --user=srini
```

To delete the history of grid routines, run the **ifx_grid_purge()** procedure. You must occasionally purge information about completed grid routines to prevent the **syscdr** database from growing too large.

- Viewing grid information
  You can view information about a grid and whether a replicate or replicate set belongs to a grid.

Related reference:
cdr change grid
cdr disable grid
cdr enable grid
cdr list grid
ifx_grid_purge() procedure

Viewing grid information

You can view information about a grid and whether a replicate or replicate set belongs to a grid.

To view information about a grid:

Run the **cdr list grid** command. For example, the following command shows the servers and authorized users for a grid named **grid1**:

```
cdr list grid grid1
```

The output for this command might be:

```
Grid                 Node                 User
-------------------- -------------------- ----------------
grid1                gserv1*              bill
                     gserv2
                     gserv3
                     gserv4
```

The user **bill** is authorized to run grid commands on the server **gserv1**.

You can see whether a replicate is a member of a grid replicate set by running the **cdr list replicate** command or the **onstat -g cat repls** command. You can also query the **syscdrrepl** SMI table. The following example output of the **cdr list replicate** command shows that the replicate is a master replicate and a member of a grid replicate set:

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------
REPLICATE:         grid_6553604_100_3
STATE:             Active ON:g_delhi
CONFLICT:          Always Apply
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       tdb:nagaraju.t1
OPTIONS:           row,ris,fullrow
```

```
REPLID:            6553605 / 0x640005
REPLMODE:          PRIMARY  ON:gserv1
APPLY-AS:          INFORMIX ON:gserv1
REPLTYPE:          Master,Grid
```

Related reference:
cdr list replicateset
cdr list replicate
onstat -g cat: Print ER global catalog information
The syscdrrepl Table

Adding replication servers to a grid

There are multiple ways to add a replication server to a grid.

You can add a replication server to a grid in the following ways:

- Run the **cdr change grid** command.
- Clone an existing replication server in the grid.

- Adding a replication server to a grid by running cdr change grid
  You can add a replication server to a grid by running the **cdr change grid** command.
- Adding a replication server to a grid by cloning
  You can add a new server to a grid by cloning an existing replication server in the grid.

Adding a replication server to a grid by running cdr change grid

You can add a replication server to a grid by running the **cdr change grid** command.

To add a replication server to a grid:

Run the **cdr change grid** command. For example, to add a replication server named **gserv3** to the grid **grid1**, run the following command:

```
cdr change grid grid1 --add=gserv3
```

To see information about the grid, such as, which servers can run grid routines and the status of routines that are run on the grid servers, run the **cdr list grid** command.
Related reference:
cdr change grid

Adding a replication server to a grid by cloning

You can add a new server to a grid by cloning an existing replication server in the grid.

The server you are adding to the grid must have the same hardware and operating system as the source server that you are cloning.
To add a server to a grid:

Clone an existing replication server in the grid by using the **ifxclone** utility with the **--disposition=ER** option. This process is described in Adding a server to the domain by cloning a server.
The following example adds a fifth server, named **serv5**, to an existing replication domain and to a grid named **grid1**. The server **serv1** is used as the source server.

1. On the **serv1** server, set the value of the ENABLE_SNAPSHOT_COPY configuration parameter to `1` in the onconfig file.
2. On the **serv5** servers, complete the **ifxclone** prerequisites for all servers, such as setting the required configuration parameters and environment variables.
   Set these environment variables:

   - INFORMIXDIR
   - INFORMIXSERVER
   - INFORMIXSQLHOSTS
   - ONCONFIG

   Set these configuration parameters to the same values on the **serv5** server as on the **serv1** server:

   - DRAUTO
   - DRINTERVAL
   - DRTIMEOUT
   - LOGBUFF
   - LOGFILES
   - LOGSIZE
   - LTAPEBLK
   - LTAPESIZE
   - ROOTNAME
   - ROOTSIZE

- PHYSBUFF
- PHYSFILE
- STACKSIZE
- TAPEBLK
- TAPESIZE

3. On the **serv5** server, run the **ifxclone** command with the **--disposition=ER** option to clone the data and the configuration of the **serv1** server onto the **serv5** server and the **--createchunkfile** command to create the necessary chunks:

```
ifxclone --trusted --source=serv1 --sourceIP=111.222.333.444
--sourcePort=1230 --target=serv5 --targetIP=111.222.333.777
--targetPort=1234 --disposition=ER --createchunkfile
```

4. Edit the sqlhosts files on all five servers in the domain so that they each have the following information:

```
#dbservername   nettype    hostname              servicename   options
gserv1          group      -                     -             i=143
serv1           ontlitcp   ny.usa.com            1230          g=gserv1
gserv2          group      -                     -             i=144
serv2           ontlitcp   tokyo.japan.com       1231          g=gserv2
gserv3          group      -                     -             i=145
serv3           ontlitcp   rome.italy.com        1232          g=gserv3
gserv4          group      -                     -             i=146
serv4           ontlitcp   perth.australia.com   1233          g=gserv4
gserv5          group      -                     -             i=147
serv5           ontlitcp   helsinki.finland.com  1234          g=gserv5
```

The server **serv5** is automatically added to the grid **grid1**.

Related concepts:
Example of setting up a replication system with a grid
Related tasks:
Adding a server to the domain by cloning a server
Related reference:
cdr change grid
Related information:
onconfig Portal: Configuration parameters by functional category
ENABLE_SNAPSHOT_COPY configuration parameter
The ifxclone utility

Adding an externally created replicate into a grid replicate set

If a replicate is created external to a grid, it can still be added to a grid replicate set.

You can add an existing replicate to a grid replicate set in the following ways:

- Run the **cdr change replicateset** command.
- Alter a replicate through a grid.

- Adding an existing replicate to a grid replicate set by using cdr change replicateset
  You can use the **cdr change replicateset** command to add replicates created outside of a grid environment to a grid replicate set.
- Adding an existing replicate to a grid replicate set by altering a table
  You can alter replicated tables through a grid even if the replicate was not created through a grid. Altering a replicated table through a grid adds the replicate to the grid replicate set.

Adding an existing replicate to a grid replicate set by using cdr change replicateset

You can use the **cdr change replicateset** command to add replicates created outside of a grid environment to a grid replicate set.

Before you begin, you must verify the following items:

- All the replicate participants are members of the grid. Replicate participants must include every member node of the grid, and no additional participants.
- Each replicate participant's information refers to the same database, owner, table name and SELECT statement.
- The replicated table schema is the same among all participants.
- The replicate does not belong to an exclusive replicate set.

To add a replicate to a grid replicate set by using the **cdr remaster** command:

1. Use the **cdr remaster** command to convert the replicate to a mastered replicate.
2. Run the **cdr change replicateset** command with the **--add** option and specifying the grid replicate set. For example, the following command adds a replicate named **vendors** to the **grid1** grid replicate set:

```
cdr change replicateset --add grid1 vendors
```

When you run the **cdr list replicate** command, the `REPLTYPE` field shows `Grid`.
Related tasks:
Adding an existing replicate to a grid replicate set by altering a table

Related reference:
cdr change replicateset
cdr list replicate
Related information:
sqlhosts connectivity information

Adding an existing replicate to a grid replicate set by altering a table

You can alter replicated tables through a grid even if the replicate was not created through a grid. Altering a replicated table through a grid adds the replicate to the grid replicate set.

Before you begin, you must verify the following items:

- All the replicate participants are members of the grid. Replicate participants must include every member node of the grid, and no additional participants.
- Each replicate participant's information refers to the same database, owner, table name and SELECT statement.
- The replicated table schema is the same among all participants.
- The replicate does not belong to an exclusive replicate set.

To alter a replicated table through a grid:

1. Connect to the grid by running the **ifx_grid_connect()** procedure with the *ER_enable* argument set to `1`.
2. Run an ALTER TABLE statement.
3. Disconnect from the grid by running the **ifx_grid_disconnect()** procedure.

The replicate is automatically remastered.
The following example adds a new column to the **special_offers** table and remasters the replicate on all participants that are members of the grid:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

ALTER TABLE special_offers ADD (
          offer_exceptions      varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related tasks:
Removing replicated columns
Adding an existing replicate to a grid replicate set by using cdr change replicateset
Related reference:
ifx_grid_connect() procedure

Creating replicated tables through a grid

You can automatically create a replicate and start replication when you create a table through the grid.

If the table you are creating is a typed table, you must define a primary key.
If you plan to create a table with a **TimeSeries** column, all grid servers must be running Informix® version 12.10 or later.

When you enable replication while creating a table through a grid, replication is set up in the following way:

- A replicate is created for the table. The replicate name is based on the name of the source server. Use the **cdr list replicate** command to see the name.
- All servers that are members of the grid are included as participants in the replicate.
- The replicate is included in a replicate set that has the same name as the grid.
- The conflict resolution rule for the replicate is time stamp if you include the WITH CRCOLS clause. Otherwise, the conflict resolution rule is always apply.
- The ERKEY shadow columns are automatically added to the table.
- All other replicate properties are the same as the default properties of a replicate created through a template.

To set up replication:

1. Connect to the grid by running the **ifx_grid_connect()** procedure with the *ER_enable* argument set to `1`.
2. Run a CREATE TABLE statement. Include the WITH CRCOLS clause if you want time stamp conflict resolution.
3. Disconnect from the grid by running the **ifx_grid_disconnect()** procedure.

The following example creates a table with replication enabled that uses the time stamp conflict resolution rule:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar)
    WITH CRCOLS;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

If you need to alter or delete a database object that you created through a grid, perform those operation from within a grid context. For example, do not create a table from within a grid and then delete the table on one of the replication servers outside of a grid context. Instead, delete the table through the grid.

Related concepts:
Conflict resolution rule
Related tasks:
Preparing tables without primary keys
Related reference:
ifx_grid_connect() procedure
cdr define template

Enabling replication within a grid transaction

You can enable replication within a transaction that is run in the context of the grid.

By default, the results of transactions run in the context of the grid are not also replicated by Enterprise Replication. In certain situations you might want to both propagate a transaction to the servers in the grid and replicate the results of the transaction.
To enable replication within a transaction:

1. Connect to the grid with the **ifx_grid_connect()** procedure.
2. Create a procedure that performs the following tasks:
   a. Defines a data variable for the Enterprise Replication state information.
   b. Runs the **ifx_get_erstate()** function and save its result in the data variable.
   c. Enables replication by running the **ifx_set_erstate()** procedure with an argument of `1`.
   d. Runs the statements that you want to replicate.
   e. Resets the replication state to the previous value by running the **ifx_set_erstate()** procedure with the name of the data variable.
3. Disconnect from the grid with the **ifx_grid_disconnect()** procedure.
4. Run the newly-defined procedure by using the **ifx_grid_procedure()** procedure.

Example

Suppose that a retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory and then replicates that summary information to a central server. A stored procedure named **low_inventory()** that creates a low inventory report exists on all the servers in the grid named **grid1**. The following example creates a new procedure named **xqt_low_inventory()** that enables replication for the **low_inventory()** procedure, and then runs the **low_inventory()** procedure:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');
CREATE PROCEDURE xqt_low_inventory()
        DEFINE curstate integer;
        EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
        EXECUTE PROCEDURE ifx_set_erstate(1);
        EXECUTE PROCEDURE low_inventory();
        EXECUTE PROCEDURE ifx_set_erstate(curstate);
END PROCEDURE;
EXECUTE PROCEDURE ifx_grid_disconnect();
EXECUTE PROCEDURE ifx_grid_procedure('grid1', 'xqt_low_inventory()');
```

The following events occur in this example:

1. The **ifx_grid_connect()** procedure connects to the **grid1** grid so that the **xqt_low_inventory()** procedure is propagated to all the servers in the **grid1** grid.
2. The **xqt_low_inventory()** procedure defines a data variable called curstate to hold the Enterprise Replication state information.
3. The **ifx_get_erstate()** function obtains the Enterprise Replication state and stores it in the curstate variable. The **ifx_set_state()** procedure enables replication.
4. The **low_inventory()** procedure is run.
5. The replication state is reset back to its original value.
6. The connection to the grid is closed by the **ifx_grid_disconnect()** procedure.
7. The **ifx_grid_procedure()** procedure runs the **xqt_low_inventory()** procedure on all the servers in the grid and the result of the **low_inventory()** procedure is replicated like any normal updating activity.

Related reference:
ifx_set_erstate() procedure
ifx_get_erstate() function

Propagating updates to data

You can change your data through a grid routine and propagate the changes to all the servers in the grid.

You can propagate updates to data on servers in the grid. By default, changes to data that are propagated through the grid are treated the same as changes to data that are made by Enterprise Replication apply threads: they are not replicated again. For example, if you propagate a DELETE statement through the grid to remove old data, you would not want the resulting deleted rows to be replicated as well. Although you can use the grid to run a DML statement, in general, use Enterprise Replication to replicate changes to replicated data.

The grid must exist and you must run the grid routines as an authorized user from an authorized server.

To propagate an SQL statement or a stored procedure that updates data, run the **ifx_grid_execute()** procedure with the DML statements or the stored procedure as the second argument.

Examples

Example 1: Reduce the price of products with low sales

In the following example, the **ifx_grid_execute()** procedure runs SQL statements that reduce the price of wool overcoats in stores that did not sell an overcoat in the last week:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1',
                 'UPDATE price_table SET price = price * 0.75
                        WHERE item =
                        (SELECT item FROM inventory i, sales s
                          WHERE i.description = "Wool Overcoat"
                             AND  i.item = s.item
                             AND  s.recent_sale_date <
                                extend (current - Interval(7) DAY))');
```

Example 2: Purge old data

The following example purges all sales records before 2010:

```
Database retail_db;
        EXECUTE PROCEDURE ifx_grid_execute('grid1',
                                 'DELETE FROM sales WHERE sales_year < 2010');
```

Example 3: Run a low inventory report

The following example runs an existing stored procedure named **low_inventory()**:

```
EXECUTE PROCEDURE ifx_grid_procedure('grid1', 'low_inventory()');
```

Related reference:
ifx_grid_execute() procedure

Administering servers in the grid with the SQL administration API

You can run SQL administration API commands in grid routines to perform administrative tasks on all servers in the grid.

The grid must exist and you must run the grid routines as an authorized user from an authorized server and while connected to the **sysadmin** database.

To propagate an SQL administration API command:

1. Run the **ifx_grid_function()** function with the SQL administration API command as the second argument.
2. Check the return code of the SQL administration API command to determine if it succeeded by running the **cdr list grid** command. The **cdr list grid** command shows the return code. The status of the **ifx_grid_function()** function can be ACK, which indicates success, even if the SQL administration API command failed.

Examples

The following examples must be run in the **sysadmin** database.

Example 1: Change a configuration parameter setting

The following example sets the maximum size of the log staging directory to 100 KB on all the servers in the grid:

```
EXECUTE FUNCTION ifx_grid_function('grid1',
                 'admin("set onconfig permanent",
          "CDR_LOG_STAGING_MAXSIZE","100")');
```

The output of the **cdr list grid** command shows that the **admin()** function succeeded because the return codes are positive numbers:

```
Grid                Node                User
------------------- ------------------- -----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
admin("set onconfig permanent",
          "CDR_LOG_STAGING_MAXSIZE","100")
ACK cdr1 2010-05-27 15:21:57
    '110'
ACK cdr2 2010-05-27 15:21:58
    '111'
ACK cdr3 2010-05-27 15:21:58
    '112'
```

Example 2: Create a new dbspace

The following example creates a new dbspace on all the servers in the **grid1** grid:

```
EXECUTE FUNCTION ifx_grid_function('grid1',
                 'task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0")');
```

The output of the **cdr list grid** command shows that the **task()** function failed:

```
Grid                 Node                 User
-------------------- -------------------- ----------------
grid1                cdr1*                bill
                     cdr2
                     cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0"
ACK cdr1 2010-05-27 15:21:57
    'Unable to create file /db/chunks/dbsp2'
ACK cdr2 2010-05-27 15:21:58
    'Unable to create file /db/chunks/dbsp2'
ACK cdr3 2010-05-27 15:21:58
    'Unable to create file /db/chunks/dbsp2'
```

Related reference:
ifx_grid_function() function
ifx_grid_execute() procedure

Propagating database object changes

You can create or alter database objects by running DDL statements while connected to the grid and propagate the changes to all the servers in the grid.

You can propagate creating, altering, and dropping database objects to servers in the grid. For example, you can create a database or table or alter an existing database or table. You can also create stored procedures and user-defined routines.

You can choose to run the DDL statements on the local server and defer the propagation of the DDL statements to the other grid servers. Deferred propagation of DDL statements can be useful when you are rolling out schema changes or performing a rolling upgrade.

The grid must exist and you must run the grid routines as an authorized user from an authorized server.

To propagate DDL statements:

1. Connect to the grid by running the **ifx_grid_connect()** procedure.
2. Run one or more SQL DDL statements.
3. Disconnect from the grid by running the **ifx_grid_disconnect()** procedure.

If you deferred the propagation of DDL statements, you can propagate them by running the **ifx_grid_release()** function, or remove them by running the **ifx_grid_remove()** function.

Example

Suppose that you have a retail shop with a website. You replicate your data to several other locations for web applications. You want to be able to quickly and easily create, drop, and update tables. You create a grid named **grid1**, from which you can update the database schema for all servers in one step. The following example creates a table for special offers in the **prod_db** database:

```
Database  prod_db;

        EXECUTE PROCEDURE ifx_grid_connect('grid1');

  CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules        lvarchar);
        EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related reference:
Example of rolling out schema changes in a grid
ifx_grid_connect() procedure
ifx_grid_disconnect() procedure

Propagating external files through a grid

You can copy non-database, external files to the servers within a grid.

The **ifx_grid_copy()** procedure copies files from a directory on the source server to a specified destination on all servers in a grid. You specify the source directory on the source server by setting the GRIDCOPY_DIR configuration parameter to the location of the file to copy. You also set the GRIDCOPY_DIR configuration parameters on each of the destination servers to specify the directory to which the file is copied. The source directory can be different than the destination directory.

The file is copied to all of the servers within the grid with the same permissions, owner, and group. The names of the group and owner are transmitted along with the file rather than the group ID and User ID because user and group names might have different group ID and User ID values on different servers.

The grid must exist and you must run the grid routines as an authorized user from an authorized server. Wildcard characters in file names are not supported.

1. On the source server, set the GRIDCOPY_DIR configuration parameter to the location of the file to copy.
2. On the destination servers, set the GRIDCOPY_DIR configuration parameter to the location of the destination of the file to copy.
3. Run the **ifx_grid_copy()** procedure specifying the grid name, the name of the file to send, and, optionally, the file destination.

Examples

Example 1: Copy a file to servers in a grid

The following example copies the file $INFORMIXDIR/tmp/myfile to the other nodes within grid **grid1**.

```
EXECUTE PROCEDURE ifx_grid_copy("grid1", "tmp/myfile")
```

Example 2: Copy a file to servers in a grid and change the name on the destination servers

In the following example, assume that the GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR/tmp on the source server and on the destination server. The following example copies the file $INFORMIXDIR/tmp/bin/sales-010512.exe on the source server to $INFORMIXDIR/tmp/bin/sales.exe on all servers within the grid mygrid.

```
EXECUTE PROCEDURE ifx_grid_copy ("mygrid", "bin/sales-010512.exe", "bin/sales.exe");
```

Related reference:
ifx_grid_copy() procedure
GRIDCOPY_DIR Configuration Parameter

Rerunning failed grid routines

You can rerun a grid routine that failed on one or more servers in the grid.

If a grid routine failed on one or more servers in the grid, you can run the **cdr list grid** command with the **--nacks** option to see the details of why it failed. If a server in the grid is offline or is not connected to the network, then a grid routine will be pending on that server and will be run when the server is reconnected to the grid.
In some cases, you should not rerun a failed routine, because the failure is expected. For example, if a server already has the database object that a grid routine is creating, then that routine fails on that server. If a command failed on all grid servers, you can run the original command again instead of running the **ifx_grid_redo()** procedure.

The grid must exist and you must run the grid routine as an authorized user from an authorized server.

To rerun a grid routine, run the **ifx_grid_redo()** procedure.
If you run the **ifx_grid_redo()** procedure without additional arguments besides the grid name, all routines that failed are re-attempted on all the servers on which they failed. You can specify on which server to rerun routines and which routines to rerun.

Example

Suppose you have a grid, named **grid1**, that contains the servers **gserv_1** and **gserv_2**, which have a database named **db1**.

You create a dbspace named **dbsp2** on the server **gserv_1** and then create a table in that dbspace in a grid context with the following commands:

```
$ dbaccess db1 -
execute procedure ifx_grid_connect('grid1');
create table t100 (c1 int primary key) in dbsp2;
execute procedure ifx_grid_disconnect();
```

The **cdr list grid** command shows that the command failed on the server **gserv_2**:

```
$ cdr list grid grid1 --nack
Grid                    Node                    User
----------------------- ----------------------- -----------------------
grid1                   gserv_1*                user1
                        gserv_2
Details for grid grid1

Node:gserv_1 Stmtid:4 User:user1 Database:db1 2011-02-24 09:27:44
create table t100 (c1 int primary key) in dbsp2
NACK gserv_2 2011-02-24 09:27:45 SQLERR:-261 ISAMERR:-130
     Grid Apply Transaction Failure
```

The error indicates that the table could not be created because the specified dbspace does not exist.

You create a dbspace named **dbsp2** on the server **gserv_2** and run the **ifx_grid_redo()** procedure to rerun the original command on **gserv_2**:

```
$ dbaccess db1 -
execute procedure ifx_grid_redo('grid1');
```

The output of the **cdr list grid** command shows that the command succeeded on both servers:

```
$ cdr list grid grid1 -v
Grid                    Node                    User
----------------------- ----------------------- -----------------------
grid1                   gserv_1*                user1
                        gserv_2
Details for grid grid1
...
```

```
Node:gserv_1 Stmtid:4 User:user1 Database:db1 2011-02-24 09:27:44
create table t100 (c1 int primary key) in dbsp2
ACK gserv_1 2011-02-24 09:27:44
ACK gserv_2 2011-02-24 09:31:09
```

Related reference:
ifx_grid_redo() procedure
cdr list grid

Connection management for client connections to participants in a grid

You can configure Connection Managers to route connection requests from clients to the replication servers of a grid.

Connection requests can be directed to replication servers based on Connection Manager service-level agreements (SLAs). You can configure Connection Manager SLAs to redirect connection requests based on various redirection policies. Connection Managers support the following redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.
- ROUNDROBIN: Connection requests are directed in a repeating, ordered fashion (round-robin) to a group of replication servers.
- WORKLOAD: Connection requests are directed to the replication server that has the lowest workload.

Related concepts:
Example of setting up a replication system with a grid
Related information:
Connection management through the Connection Manager
Example of configuring connection management for a grid or replicate set

Grid queries

If you have a table that is the same on multiple servers in a grid, but whose data is not replicated, you can run a grid query to return the consolidated data from the multiple servers.

For example, suppose that you have a chain of retail stores. Each store has a database with the same schema. The database contains tables for inventory, customer data, and sales transactions. You set up a grid because you want to replicate the inventory tables to a central server. You want the tables for sales transactions to be the same on every server, but you do not want to replicate all the sales transactions to the central server. You do, however, want a monthly report that shows the total sales per store. You run a grid query on the central server that aggregates the sales data for the last month for each store and returns results that are grouped by store.

To run a grid query, you include the GRID clause in the SELECT statement. The GRID clause specifies the grid, or subset of the grid, on which to run the query. The GRID clause has requirements and restrictions for the tables and other SQL constructs that you can include in the query.

Before you can run a grid query, you must define the table that you want to query as a grid table. If you use secure connections between your grid servers, you must configure secure connections on the grid server from which you want to run grid queries.

Planning for grid queries

Consider the following options when you plan grid queries.

Before you run a grid query, you can configure the following options for the queries:

- Whether to run the grid query on all the servers in the grid or a subset of grid servers. To define subsets of grid servers, create regions by running the **cdr define region** command. You can create as many grid regions as you need. Grid regions can overlap or be divided into smaller grid regions. A grid server can be a member of multiple grid regions.
- Whether to make all SELECT statements that are run in the current session run as grid queries by default. Run the SET ENVIRONMENT SELECT_GRID or the SET ENVIRONMENT SELECT_GRID_ALL statements to specify the grid or region name for every query. Leave the GRID clause out of SELECT statements.
- Whether to skip grid servers that are not available when you run the grid query. By default, the grid query runs only if all servers are available. Run the SET ENVIRONMENT GRID_NODE_SKIP ON statement to run the query on the available servers and skip the unavailable servers.

While you run a grid query, besides choosing the tables and the grid or region to include in the query, you can include the following options:

- Whether to return all qualifying rows, including duplicate rows. By default, grid queries return only unique rows. Include the ALL keyword in the GRID clause to return all rows.
- Whether to return information about which server the results are from. Include the **ifx_node_id()** or **ifx_node_name()** function to return a column that identifies the grid server from which each row originates. You can use the server ID or name to group the results.

After you run a grid query, you can find out which servers were skipped for a grid query, if the GRID_NODE_SKIP option was set to ON. Run the **ifx_gridquery_skipped_node_count()** and **ifx_gridquery_skipped_nodes()** functions to return the grid servers that were unavailable during the grid query.

- Defining tables for grid queries
  Define the tables that you want to include in grid queries as grid tables.
- Configuring secure connections for grid queries
  If the sqlhosts files on the grid servers include the s=6 option, you must define alternate connections for grid queries. On the grid server from which you want to run grid queries, create a grid.servers file that lists the server group names and aliases for the other grid servers.
- Examples of grid queries
  These examples show some of the options that you have when you run grid queries.

Related reference:
ifx_grid_connect() procedure
cdr define region
cdr delete region
cdr change gridtable
cdr remaster gridtable
ifx_node_id() function
ifx_node_name() function
ifx_gridquery_skipped_nodes() function
ifx_gridquery_skipped_node_count() function
ifx_grid_release() function
ifx_grid_remove() function
Related information:
SELECT_GRID session environment option
SELECT_GRID_ALL session environment option
GRID_NODE_SKIP session environment option
GRID clause

Defining tables for grid queries

Define the tables that you want to include in grid queries as grid tables.

The only prerequisite for defining a table as a grid table is that the table must have the same name, column names, and data types on multiple grid servers. However, the GRID clause has other restrictions and requirements for running grid queries.

You can include system catalog and **sysmaster** databases tables in grid queries without defining them as grid tables.
To define a table as a grid table, run the **cdr change gridtable** command. The **cdr change gridtable** command verifies that the tables have matching column names and data types across the grid.
For example, the following command defines the **items**, **orders**, and **customer** tables in the **stores_demo** database for the grid named **grid1**:

```
cdr change gridtable --grid=grid1 --database=stores_demo --add items orders customer
```

If you want to alter a grid table, you must run the alter operation through the grid. You cannot run a grid query on the table during an alter operation. After the alter operation is complete, the database server verifies that the table is consistent across grid servers.

Related reference:
cdr change gridtable
cdr remaster gridtable
Related information:
GRID clause

Configuring secure connections for grid queries

If the sqlhosts files on the grid servers include the s=6 option, you must define alternate connections for grid queries. On the grid server from which you want to run grid queries, create a grid.servers file that lists the server group names and aliases for the other grid servers.

You do not need to encrypt the file. Authentication is done through normal authentication methods.
To configure secure connections for grid queries:

On the grid server from which you want to run grid queries, create a text file named grid.servers in the INFORMIXDIR/etc directory. List each grid server group name and alias on a separate line.
For example, the following sqlhosts file for a grid uses the `s=6` option for secure connections:

```
#dbservers     nettype    hostname    servicename    options

g_ca_sf        group      -           -              i=100
san_francisco  ontlitcp   computer1   sf_alt         g=g_ca_sf,s=6

g_ca_sj        group      -           -              i=200
san_jose       ontlitcp   computer2   sj_alt         g=g_ca_sj,s=6

g_ca_okl       group      -           -              i=300
oakland        ontlitcp   computer3   okl_alt        g=g_ca_okl,s=6

g_ca_yk        group      -           -              i=400
yreka          ontlitcp   computer4   yk_alt         g=g_ca_yk,s=6

g_ca_sac       group      -           -              i=500
sacramento     ontlitcp   computer5   sac_alt        g=g_ca_sac,s=6

g_ca_stk       group      -           -              i=600
stockton       ontlitcp   computer6   stk_alt        g=g_ca_stk,s=6
```

The corresponding grid.servers file has the following contents:

```
#group     alias
g_ca_sf    sf_alt
```

```
g_ca_sj    sj_alt
g_ca_okl   okl_alt
g_ca_yk    yk_alt
g_ca_sac   sac_alt
g_ca_stk   stk_alt
```

Related tasks:
Configuring secure ports for connections between replication servers
Related reference:
ifx_grid_connect() procedure
ifx_grid_release() function
ifx_grid_remove() function
Related information:
GRID clause

Examples of grid queries

These examples show some of the options that you have when you run grid queries.

The following examples are based on the **stores_demo** database. A grid named **grid1** has eight servers, named **store1** through **store8**. The examples assume that you defined the **items**, **orders**, and **customer** tables as grid tables.

Example 1: Return chunk information about grid servers

Suppose you want to know about the chunks on all your grid servers. You want to know the number of chunks, which dbspaces each chunk is in, the total size of each chunk, and the amount of free space in each chunk.

You run the following grid query to return chunk information for each grid server. The tables in the **sysmaster** database are grid tables by default.

```
database sysmaster;

SELECT ifx_node_name()::char(12) AS node, chknum, dbsnum, nfree, chksize
FROM syschunks GRID ALL 'grid1';
```

The grid query returns the following results:

```
node          chknum dbsnum       nfree     chksize

store1             1      1     1777275     2000000
store1             2      2        5025      100000
store1             3      3       24974      100000
store2             1      1     1775579     2000000
store2             2      2        5025      100000
store2             3      3       24974      100000
store3             1      1     1769260     2000000
store3             2      2        5025      100000
store3             3      3       24974      100000

. . .
```

Example 2: Aggregate results by server and find skipped servers

Suppose you want a list of the orders by customer for each store in the grid named **grid1**. A store is represented by its grid server name. You want to return all results, including duplicate rows. You do not want the query to fail if any of the grid servers are unavailable, but you want to know which servers were skipped.

Before you run the grid query, you run the following statement to run the query on available grid servers and skip any unavailable grid servers:

```
SET ENVIRONMENT GRID_NODE_SKIP ON;
```

You run the following grid query to return the outstanding orders by customer for each store:

```
SELECT c.fname, c.lname, ifx_node_name() AS node
  SUM(i.total_price) AS tot_amt, SUM(i.quantity) AS tot_cnt
  FROM items i, orders o, customer c GRID ALL 'grid1'
  WHERE i.order_num = o.order_num
  AND o.customer_num = c.customer_num
  GROUP BY 1,2
  ORDER BY 2,1,3;
```

The grid query returns the following results:

```
fname    Alfred
lname    Grant
node     store1
tot_amt  $84.00
tot_cnt  2

fname    Alfred
lname    Grant
node     store2
tot_amt  $84.00
tot_cnt  4

. . .
```

You run the following statement to find how many grid servers were skipped:

```
EXECUTE FUNCTION ifx_gridquery_skipped_node_count();

2
```

Two servers were skipped. You run the **ifx_gridquery_skipped_nodes()** statement for each of the skipped servers:

```
EXECUTE FUNCTION ifx_gridquery_skipped_nodes();

store5

EXECUTE FUNCTION ifx_gridquery_skipped_nodes();

store8
```

Example 3: Query a region of the grid

Suppose you want to know the total sales and number of sales per person for each store in Kansas. Kansas has two stores whose grid servers are named **store3** and **store4**. You want all queries during your database session to be run as grid queries for the Kansas stores.

You run the following command to define a grid region named **region1** that contains the servers **store3** and **store4**:

```
cdr define region --grid=grid1 region1 store3 store4
```

You run the following statement to set all SELECT statements during the session as grid queries for the region **region1**:

```
SET ENVIRONMENT SELECT_GRID_ALL region1
```

You run the following statement to return the total sales and number of sales per person for each store. The GRID clause is not necessary because you set the SELECT_GRID_ALL option.

```
SELECT fname[1,10], lname[1,10], ifx_node_id() AS storenum,
  SUM(quantity) AS tot_cnt, SUM(total_price) AS tot_amt
  FROM items i, orders o, customer c
  WHERE i.order_num = o.order_num
  AND o.customer_num = c.customer_num
  GROUP BY 2,1
  ORDER BY 2,1,3;
```

The query returns the following results:

```
fname      lname          storenum          tot_cnt          tot_amt

Alfred     Grant                 3                8           $84.00
Alfred     Grant                 4                6           $84.00
Marvin     Hanlon                3               12          $438.00
Marvin     Hanlon                4               10          $438.00
Anthony    Higgins               3               45         $1451.80
Anthony    Higgins               4               36         $1451.80
Roy        Jaeger                3               16         $1390.00
Roy        Jaeger                4               13         $1390.00
Fred       Jewell                3               16          $584.00
Fred       Jewell                4               13          $584.00
Frances    Keyes                 3                4          $450.00
Frances    Keyes                 4                3          $450.00

. . .
```

Example 4: Use a grid query as a subquery

Suppose you want the total sales and number of sales for each customer across all stores. You use the same query that you use in example 2 as the subquery to return information by grid server. The main query aggregates the results of the subquery.

You run the following statement to return the total sales and number of sales per person:

```
SELECT fname, lname,
   SUM(tot_amt) AS amt_by_person, SUM(tot_cnt) AS tot_by_person
   FROM
       (
         SELECT c.fname, c.lname, ifx_node_name() AS node,
            SUM(i.total_price) AS tot_amt, SUM(i.quantity) AS tot_cnt
            FROM items i, orders o, customer c GRID ALL 'grid1'
            WHERE i.order_num = o.order_num
            AND o.customer_num = c.customer_num
            GROUP BY 1,2
       )
   GROUP BY fname, lname
   ORDER BY 2, 1;
```

The query returns the following results:

```
fname           lname              amt_by_person    tot_by_person

Alfred          Grant                    $336.00               20
Marvin          Hanlon                  $1752.00               40
Anthony         Higgins                 $5807.20              135
Roy             Jaeger                  $5560.00               50
Fred            Jewell                  $2336.00               50
Frances         Keyes                   $1800.00               10
Margaret        Lawson                  $1792.00              110
```

. . .

Related reference:
ifx_grid_connect() procedure
cdr define region
cdr delete region
cdr change gridtable
cdr remaster gridtable
ifx_node_id() function
ifx_node_name() function
ifx_gridquery_skipped_nodes() function
ifx_gridquery_skipped_node_count() function
ifx_grid_release() function
ifx_grid_remove() function
Related information:
SELECT_GRID session environment option
SELECT_GRID_ALL session environment option
GRID_NODE_SKIP session environment option
GRID clause

Shard cluster setup

Sharding is a way to horizontally partition a single table across multiple database servers in a shard cluster. Enterprise Replication moves the data from the source server to the appropriate target server as specified by the sharding method. You query a sharded table as if the entire table is on the local server. You do not need to know where the data is. Queries that are performed on one shard server retrieve the relevant data from other servers in a shard cluster. Sharding reduces the index size on each shard server and distributes performance across hardware. You can add shard servers to the shard cluster as your data grows.

Prerequisites

Before you create a shard cluster, the following system must be in place:

- You must have an Enterprise Replication domain that is composed of two or more nodes.
- On one of the Enterprise Replication nodes, you must have a table or collection to shard that conforms to the following requirements:
  - The table must have a dedicated column or field for tracking row or document versions.
  - The table cannot include data types that are not supported in sharded queries.
  - The databases on all shard servers must have same locale type.

Shard cluster architecture

Shard servers are uniquely identified by the SHARD_ID configuration parameter that you must set on each shard server. Because shard servers have unique IDs, Enterprise Replication can efficiently communicate between shard servers:

- Client connections are multiplexed over a common pipe and authenticated only on the local shard server.
- Sharded queries are run in parallel on all shard servers and their high-availability secondary servers.
- For insert, update, and delete operations, if you set the USE_SHARDING session environment option, transactions use the two-phase commit protocol to move data to appropriate shard server. Otherwise, changes are moved to appropriate shard server using the eventually consistent model after the transaction is committed. For select operations, if you set the USE_SHARDING session environment option, queries are run on all shard servers in the cluster instead of on only the local database server.
- The consistency of the sharded table is enforced on all shard servers. Shard servers do not need to transfer table information between each other. Data definition language statements that you run on a sharded table are propagated to all shard servers.

- Creating a shard cluster
  To create a shard cluster, prepare the shard servers and specify the sharding definition.
- Sharded queries
  You can query a sharded table as if it is a single table on one database server. However, restrictions for distributed queries between database servers and restrictions specific to sharded queries apply.
- Shard cluster management and monitoring
  You can scale out a shard cluster by adding new shard servers. You can also change the shard cluster's definition to change where rows or documents are distributed to.
- Shard edge server
  Shard edge server simplifies the administration of large shard cluster when shard cluster is only used for shard query functionality.

Related concepts:
Preparing the Replication Environment
Using High-Availability Clusters with Enterprise Replication
Grid setup and management
Managing Replication Servers and Replicates
Monitor and troubleshooting Enterprise Replication
Related tasks:
Defining Replication Servers, Replicates, Participants, and Replicate Sets
Related information:
JSON data sharding
Components supporting high availability and scalability

Creating a shard cluster

To create a shard cluster, prepare the shard servers and specify the sharding definition.

All shard servers must belong to the same Enterprise Replication domain.

To create a shard cluster:

1. On each shard server, set the SHARD_ID configuration parameter to a positive integer value that is unique in the shard cluster by running the following command:

   ```
   onmode -wf SHARD_ID=unique_positive_integer
   ```

   If the SHARD_ID configuration parameter is already set to a positive integer, you can change the value by editing the onconfig file and then restarting the database server. You can also set the SHARD_MEM configuration parameter to customize the number of memory pools that are used during shard queries.
2. On the shard server that contains the table to shard, run the **cdr define shardCollection** command.

When applications connect to shard servers, enable sharded queries to run against data across all shard servers by setting the USE_SHARDING session environment variable:

```
SET ENVIRONMENT USE_SHARDING ON;
```

- Shard cluster definitions
  The definition for a shard cluster includes information about the shard servers, the data to shard, and the sharding method.

Related reference:
cdr define shardCollection
SHARD_ID configuration parameter
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters
USE_SHARDING session environment option

Shard cluster definitions

The definition for a shard cluster includes information about the shard servers, the data to shard, and the sharding method.

To run the **cdr define shardCollection** command, which creates a sharding definition for partitioning your table data, you must specify the following information:

- A name for the sharding definition
- The name of the database that contains the table that is being sharded
- The name of the user that owns the table that is being sharded
- The sqlhosts file group name for each database server in the shard cluster
- The column that is used as a shard key
- Which sharding method the database server uses for determining where rows are distributed to:
  - With consistent hash-based sharding, the data is automatically distributed between shard servers in a way that minimizes the data movement when you add or remove shard servers.
  - With hash-based sharding, the data is automatically divided between shard servers, but when you change the shard cluster, all data is redistributed.
  - With expression-based sharding, you specify how the data is divided between shard servers. You must also specify the shard server to receive the data that is outside the scope of the expression.
- How you want to distribute the data:
  - Insert rows on any shard server, replicate the rows to the appropriate shard server, and then delete duplicate rows from the original server. The delete method is the default method and is the same behavior as when you define sharding with MongoDB commands.
  - Insert rows on any shard server, replicate the rows to the appropriate shard server, but then keep duplicate rows on the original server. The keep method is similar to a data dissemination system.
  - Insert rows on the appropriate shard server and do not replicate rows. The informational method is useful if you want to query across multiple servers that have the same table, but you do not need to shard the data during loading. For example, you have a different database server for each of your three stores. The data from each store is always inserted in the appropriate server. You set up the sharding definition with an expression that matches database servers with their store identifiers. Then you can run sharded queries to aggregate data from all three stores.
- The table column or collection field for tracking row updates

Consistent hash-based sharding

When you create a consistent hash-based sharding definition, IBM® Informix® uses a hash value of a specific column or field to distribute data to the servers of a shard cluster in a consistent pattern. When you add or remove a shard server, the consistent hashing algorithm redistributes a fraction of the data. You specify how many hashing partitions to create on each shard server. The default number of hashing partitions is three. The more hashing partitions, the more evenly the data is distributed among shard servers. However, if you specify more than 10 hashing partitions, the resulting SQL statement to create the sharded table might fail because it exceeds the maximum character limit for an SQL statement.

For example, the following command creates a consistent hashing index that has three partitions on each shard server:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1 g_shard_server_2 g_shard_server_3
```

You can dynamically change the number of hashing partitions per shard server by running the **cdr change shardCollection** command.

Hash-based sharding

When you create a hash-based sharding definition, IBM Informix uses a hash value of a specific column or field to distribute data to the servers of a shard cluster. When you add or remove a shard server, the hashing algorithm redistributes all the data.

For example, the following command creates a hashed index that is based on shard key values, and then the Enterprise Replication determines where rows with specific hashed index values are distributed to:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A g_shard_server_B g_shard_server_C g_shard_server_D
```

Expression-based sharding

When you create an expression-based sharding definition, IBM Informix uses WHERE-clause syntax on a specific column or field to distributes data to the servers of a shard cluster.

For example, the following command sends rows with a shard-key value of `NV` to **g_shard_server_B**:

```
cdr define shardCollection collection_1 db_1:joe.clients
--type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_A "IN ('WA','OR','ID')"
   g_shard_server_B "IN ('CA','NV','UT','AZ')"
   g_shard_server_C "IN ('TX','OK','NM','AR','LA')"
   g_shard_server_D  REMAINDER
```

Sharding definitions must include the `REMAINDER` expression for rows or documents that have values that are not accounted for by the other expressions. For example, the previous sharding definition sends rows with a shard-key value of `'NY'` to **g_shard_server_D**.

Expressions that are used for sharding data cannot overlap. For example, a sharding definition that is created with the following command is not valid because rows or documents with shard key values `40` to `60` would be sent to both **g_shard_server_A** and **g_shard_server_B**.

```
cdr define shardCollection collection_1 db_1:joe.clients)
--type=delete --key=age --strategy=expression --versionCol=version
   g_shard_server_A "BETWEEN 0 AND 60"
   g_shard_server_B "BETWEEN 40 AND 100"
   g_shard_server_C  REMAINDER
```


Sharded queries

You can query a sharded table as if it is a single table on one database server. However, restrictions for distributed queries between database servers and restrictions specific to sharded queries apply.

When you run a sharded query, do not include server name qualifications for remote servers.

If the SHARD_ID configuration parameter is set to unique values on each shard server in the shard cluster, sharded queries are run in parallel on each shard server.

If you set the USE_SHARDING session environment option, insert, update, and delete operations on shard tables use the two-phase commit protocol. Otherwise, sharded insert, update, and delete operations follow the eventually consistency model where data is moved to the appropriate shard server after the transaction is committed.

If your shard servers have high-availability secondary servers, you can run sharded queries from the secondary servers.

Data types

A sharded query can return the following data types: non-opaque atomic built-in data types, LVARCHAR, Boolean, BSON, and JSON. Sharded queries cannot return distinct data types.

To run sharded queries on time series data in a **TimeSeries** data type, shard a virtual table that is based on the time series table.

Restrictions

You cannot include the following SQL syntax elements in a query that includes a sharded table:

- DataBlade API routines
- Java user-defined routines
- Triggers
- A FOR UPDATE clause in a SELECT statement

You cannot run an EXECUTE FUNCTION or EXECUTE PROCEDURE statement for a routine to operate on a sharded table.

You cannot run a statement that contains an update to a shard key that requires the row to move to another shard server. To update the shard key of a row, delete the row and then insert it with the new values.

You cannot shard data in an XA environment.

Sharded Table Joins

IBM® Informix® supports joining between multiple sharded tables with parallel execution. However, such parallel joins between two shard tables are allowed ONLY when the following conditions are met:

- both the tables must have the joining column as key

- both the tables must have exactly the same strategy defined on the key
- both the tables must have exact partitioning conditions defined using the key
- both the tables must have same set of the participating nodes
- only equi-joins are allowed in case of other non-expression strategies

All the table filters on the sharded tables will be pushed to their respective participants

When the shard-join is rejected due to any restriction, Informix server will attempt a fall back mechanism.

Shard join fallback is enabled using the following command:

SET ENVIRONMENT SHARDJOIN_FALLBACK ON. For more information, see SHARDJOIN_FALLBACK session environment option.

Performance tips

You can improve the speed of sharded queries by customizing how shared memory for sharded queries is allocated. You can control shared memory allocation by setting the SHARD_MEM configuration parameter on each shard server.

If your sharded queries frequently include joins to another table, replicate that table to all the shard servers to improve query performance.

If your sharded queries included stored routines as a filter, define the routines on all the shard servers. Queries run faster when the data is filtered on each shard server before being returned.

If the SHARD_ID configuration parameter is set on all shard servers, the shard servers use server multiplexer group (SMX) connections. You can reduce latency between shard servers by increasing the number of pipes that are used for the SMX connections. Set the SMX_NUMPIPES configuration parameter to the number of pipes.

Related reference:
SHARD_MEM configuration parameter

Shard cluster management and monitoring

You can scale out a shard cluster by adding new shard servers. You can also change the shard cluster's definition to change where rows or documents are distributed to.

Modifying a shard-cluster and adding or removing shard servers

To modify the servers in a shard cluster, run the **cdr change shardCollection** command on one of the shard servers. The **cdr change shardCollection** command performs the following actions:

1. A new sharding definition is created.
2. Using the new sharding definition, existing data is distributed across the shard servers.
3. The original sharding definition is deleted.

Note: You can add new servers, remove existing servers, or modify the sharding definition, but you cannot change the type of sharding definition. A hash-based sharding definition cannot change to an expression-based sharding definition, and an expression-based sharding definition cannot change to a hash-based sharding definition.
The following example shows how to add capacity to a shard cluster that uses a hash-based sharding definition. The original shard cluster was defined with the following command:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=identifier --strategy=hash --versionCol=version
   g_shard_server_1
   g_shard_server_2
```

Run the following command to add **g_shard_server_3** to the shard cluster:

```
cdr change shardCollection collection_1 --add g_shard_server_3
```

The following example shows how to add a new region-specific server to a shard cluster that uses an expression-based sharding definition. The original shard cluster was created with the following command:

```
cdr define shardCollection collection_2 db_2:john.clients
   --type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_1 "IN ('WA','OR')"
   g_shard_server_2 "IN ('CA','NV')"
   g_shard_server_3  remainder
```

Run the following command to add **g_shard_server_4** to the shard cluster:

```
cdr change shardCollection collection_2 --add
   g_shard_server_4 "IN ('UT','ID')"
```

A new sharding definition is created. Any rows or documents that have a shard key of **UT** or **ID** are moved from **g_shard_server_3** to **g_shard_server_4**, and all future inserts are distributed according to the new sharding definition.
You can remove servers from a shard cluster using the **--drop** option or entirely replace a sharding definition with the **--replace** option.

Monitoring a shard cluster

To see the current definition for a shard cluster, you can run the **cdr list shardCollection** command on one of the shard servers, and specify the definition's name. For example:

```
cdr list shardCollection my_collection
```

To see information on the shard cache, run the **onstat -g shard** command on one of the shard servers.

Stopping data distribution and deleting a sharding definition

To stop data distribution and delete the sharding definition, run the **cdr delete shardCollection** on one of the shard servers, and specify the definition's name. For example:

```
cdr delete shardCollection my_collection
```

Related reference:
cdr change shardCollection
cdr delete shardCollection
cdr list shardCollection
Related information:
onstat -g shard command: Print information about the shard cache

Shard edge server

Shard edge server simplifies the administration of large shard cluster when shard cluster is only used for shard query functionality.

If you do not require automatic data partitioning, then you can designate one or more servers as shard edge servers so that you only need to define Enterprise replication at the shard coordinator server. Shard edge servers are standard Informix servers and you cannot start shard query from shard edge server.

Use the SHARD_EDGE_NODE configuration parameter to enable shard server as a edge server. For more information, see SHARD_EDGE_NODE configuration parameter.

Example

This example shows how to configure shard cluster between three servers: server1, server2 and server3. We will make server1 as our shard coordinator and server2 and server3 as shard edge servers.

1. Configure SQLHOSTS

SQLHOSTS file for server1:

```
group1 group - - i=1
server1 onsoctcp host1 10000 g=group1
group2 group - - i=2
server2 onsoctcp host2 10000 g=group2
group3 group - - i=3
server3 onsoctcp host3 10000 g=group3
```

Note: Even though we do not require Enterprise Replication defined for server2 and server3, server1 sqlhosts file need to be configured with group entries for server2 and server3.
SQLHOSTS file for server2:

```
SQLHOSTS file for server2:
server2 onsoctcp host2 10000
server1 onsoctcp host1 10000
```

SQLHOSTS file for server3:

```
SQLHOSTS file for server3:
server3 onsoctcp host3 10000
server1 onsoctcp host1 10000
```

2. Establish trusted host relationship between server1 and server2, and between server1 and server3

3. Define Enterprise replication for server1

```
cdr define server –connect server -I group1
```

4. Update server2 and server3 config file to set SHARD_EDGE_NODE config value to 1

5. For parallel shard query function, make sure to set unique value to SHARD_ID config parameter at server1, server2 and server3

6. Create shard definition one or more tables to run shard queries

```
cdr define shardCollection –connect server1 customer_shard stores_demo:usr1.sales_bson --type informational_noer --key
  bson_value_lvarchar(sales_data, 'amount') --strategy hash group group group
```


Managing Replication Servers and Replicates

These topics cover how to manage your Enterprise Replication system, including managing replication servers, replicates and participants, replicate sets, templates, replication server network connections, and resynchronizing data, and performing alter operations on replicated tables.

- Managing Replication Servers

- Managing Replicates
- Managing Replicate Sets
- Managing Templates
- Managing Replication Server Network Connections
- Resynchronizing Data among Replication Servers
  If replication has failed for some reason and data is not synchronized, there are different ways to correct data mismatches between replicated tables.
- Alter, rename, or truncate operations during replication
  When Enterprise Replication is active and data replication is in progress, you can perform many types of alter, rename, or truncate operations on replicated tables and databases.
- Recapture replicated transactions
  If you want a transaction to continue to be replicated after it reaches the target replication servers, you can use the **ifx_set_erstate()** procedure.

Related concepts:
Preparing the Replication Environment
Using High-Availability Clusters with Enterprise Replication
Grid setup and management
Shard cluster setup
Monitor and troubleshooting Enterprise Replication
Related tasks:
Defining Replication Servers, Replicates, Participants, and Replicate Sets

Managing Replication Servers

You manage replication servers with the **cdr** commands.

The *state* of the server refers to the relationship between the source server and the target server. To determine the current state of the server, use the **cdr list server** ***server_name*** command. For more information about the possible server states, see cdr list server.

Note: When switching a server to administration mode to perform administrative tasks, be aware that any Enterprise Replication on the server will be started (or continue to run normally if already started). In this situation data on which you might be relying may change as other users modify it, and concurrency problems may arise as others access the same data. To avoid this problem, launch the server using the **oninit -Dj** command; if the server is already running, use the **cdr stop** command to shut down any currently running replications.

- Modify server attributes
  You modify replication server attributes by running the **cdr modify server** command.
- Dynamically Modifying Configuration Parameters for a Replication Server
  You can alter the settings for Enterprise Replication configuration parameters and environment variables on a replication server while replication is active.
- Viewing Replication Server Attributes
- Connect to another replication server
  By default, when you view information about a server, Enterprise Replication connects to the global catalog of the database server specified by the INFORMIXSERVER environment variable. You can connect to the global catalog of another database server by using the **--connect** option.
- Temporarily stopping replication on a server
  You can temporarily stop replication on a server to perform maintenance tasks in several different ways.
- Restarting Replication on a Server
  You can restart replication after Enterprise Replication was temporarily stopped.
- Suspending Replication for a Server
- Resuming a Suspended Replication Server
- Deleting a Replication Server
  You can remove Enterprise Replication from a database server and then remove the database server from an Enterprise Replication domain.

Related reference:
Set configuration parameters for replication

Modify server attributes

You modify replication server attributes by running the **cdr modify server** command.

You can change the following attributes of the server:

- Idle timeout
- Whether Aborted Transaction Spooling (ATS) files or Row Information Spooling (RIS) files are generated
- Location of the directory for the ATS or RIS files
- The format of the ATS files: text, XML, or both
- The mode of all the participants on the server: primary, receive-only, or send-only

Related tasks:
Defining Replication Servers
Related reference:
cdr modify server

Dynamically Modifying Configuration Parameters for a Replication Server

You can alter the settings for Enterprise Replication configuration parameters and environment variables on a replication server while replication is active.

Use the following commands to dynamically update values of most Enterprise Replication configuration parameters:

cdr add onconfig
: Adds a value. This option is available only for configuration parameters and environment variables that allow multiple values.

cdr change onconfig
: Replaces the existing value. This option is available for all Enterprise Replication configuration parameters and environment variables.

cdr remove onconfig
: Removes a specific value. This option is available only for configuration parameters and environment variables that allow multiple values.

The commands change configuration parameters in the onconfig file. To update environment variables, use the CDR_ENV configuration parameter.

To dynamically update the value of the CDR_DELAY_PURGE_DTC configuration parameter, use the **onmode -wf** command.

The following table shows which changes are valid for Enterprise Replication configuration parameters.

Table 1. Options for dynamically updating Enterprise Replication configuration parameters

Configuration Parameter	cdr add onconfig	cdr change onconfig	cdr remove onconfig
CDR_APPLY	No	No	No
CDR_DBSPACE	No	Yes	No
CDR_DSLOCKWAIT	No	Yes	No
CDR_ENV CDR_ALARMS	No	No	No
CDR_ENV CDR_LOGDELTA	No	Yes	No
CDR_ENV CDR_PERFLOG	No	Yes	No
CDR_ENV CDR_RMSCALEFACT	No	Yes	No
CDR_ENV CDR_ROUTER	No	Yes	No
CDR_ENV CDRSITES_731	Yes	Yes	Yes
CDR_ENV CDRSITES_92X	Yes	Yes	Yes
CDR_ENV CDRSITES_10X	Yes	Yes	Yes
CDR_EVALTHREADS	No	Yes	No
CDR_LOG_LAG_ACTION	Yes	Yes	Yes
CDR_LOG_STAGING_MAXSIZE	Yes	Yes	Yes
CDR_MAC_DYNAMIC_LOGS	No	Yes	No
CDR_NIFCOMPRESS	No	Yes	No
CDR_QDATA_SBSPACE	Yes	Yes	Yes
CDR_QUEUEMEM	No	Yes	No
CDR_SERIAL	No	Yes	No
CDR_SUPPRESS_ATSRISWARN	Yes	Yes	Yes
ENCRYPT_CDR	No	Yes	No
ENCRYPT_CIPHERS	No	Yes	No
ENCRYPT_MAC	Yes	Yes	Yes
ENCRYPT_MACFILE	Yes	Yes	Yes
ENCRYPT_SWITCH	No	Yes	No

You can view the setting of Enterprise Replication configuration parameters and environment variables with the **onstat -g cdr config** command.

Related reference:
onstat -g cdr config: Print ER settings
cdr add onconfig
cdr change onconfig
cdr remove onconfig
Enterprise Replication configuration parameter and environment variable reference

Viewing Replication Server Attributes

After you define a server for replication, you can view information about the server using the **cdr list server** command. If you do not specify the name of a defined server on the command line, Enterprise Replication lists all the servers that are visible to the current server. If you specify a server name, Enterprise Replication displays information about the current server, including server ID, server state, and attributes.

For more information, see cdr list server.

Connect to another replication server

By default, when you view information about a server, Enterprise Replication connects to the global catalog of the database server specified by the INFORMIXSERVER environment variable. You can connect to the global catalog of another database server by using the **--connect** option.

For example, to connect to the global catalog of the database server **idaho**, enter: `cdr list server --connect=idaho`

Related concepts:
Enterprise Replication Terminology
Connect Option

Temporarily stopping replication on a server

You can temporarily stop replication on a server to perform maintenance tasks in several different ways.

You can stop Enterprise Replication on a server by shutting down the database server. Replication begins again when you restart the database server.

However, you might want to temporarily stop the Enterprise Replication threads without stopping the database server.

You can temporarily stop replication by running the **cdr stop** command. The stopped server does not capture data to be replicated. Other replication servers in the domain continue to queue replicated data for the stopped server in their send queues. Replication threads remain stopped (even if the database server is stopped and restarted) until you run the **cdr start** command. When you restart replication on the server, it receives and applies the replicated data from the other replication servers. However, if replication is stopped for long enough, the replay position on the logical log on the stopped server can be overrun and the send queues on the active replication servers can fill up. If either of these situations happens, you must synchronize the server that was stopped.

If your replicates use time stamp or delete wins conflict resolution rules, you should temporarily stop replication on the server by using the **cdr disable server** command. Disabling a replication server is also appropriate if you do not have enough disk space to avoid overrunning the replay position. Replication servers do not queue replicated transactions for the disabled replication server, nor does the disabled replication server queue its transactions. Therefore, you must synchronize the replication server that was disabled after you enable replication on it by using the **cdr check replicateset** command. However, because information about deleted rows on the disabled replication server is saved in delete tables, you can take advantage of a time stamp repair.

Related reference:
cdr stop
cdr disable server

Restarting Replication on a Server

You can restart replication after Enterprise Replication was temporarily stopped.

If replication was stopped by the **cdr disable server** command, you can restart it by running the **cdr check replicateset** command with the **--repair** and the **--enable** options or by running the **cdr enable server** command. If you use the **cdr enable server** command, you must subsequently synchronize the server.

If replication stopped due to an error, you can restart replication by shutting down and restarting the database server or by running the **cdr start** command.

If replication was stopped by the **cdr stop** command, restart replication by running the **cdr start** command.

When you run the **cdr start** command, Enterprise Replication resumes evaluating the logical logs at the replay position (where Enterprise Replication stopped evaluating the logical log when the server was stopped). If the replay position was overwritten in the logical log, replication cannot restart and event alarm 75 is raised. In this situation, run the **cdr cleanstart** command to restart Enterprise Replication and then synchronize the data.

Related reference:
cdr start
cdr enable server

Suspending Replication for a Server

If you do not want to completely shut down the Enterprise Replication threads, you can suspend replication of data to the server using the **cdr suspend server** command. When replication is suspended to the server, the source server queues replicated data but suspends delivery of replicated data to the target server. Note that this command does not affect the network connection to the suspended server. The source server continues to send other messages, such as acknowledgment and control messages.

For example, to suspend replication of data to the server group **g_papeete** from the server group **g_raratonga**, enter: `cdr suspend server g_papeete g_raratonga`

To suspend replication to **g_papeete** from all servers in the enterprise, enter:

```
cdr suspend server g_papeete
```

Important: When you suspend replication on a server, you must ensure that the send queues on the other Enterprise Replication servers participating in replication do not fill.
For more information, see cdr suspend server.

Resuming a Suspended Replication Server

To resume replication to a suspended server, use the **cdr resume server** command, specifying which server you want to resume. When you resume the server, the queued data is delivered.

For example, to resume replication to the **g_papeete** server group, enter:

```
cdr resume server g_papeete
```

For more information, see cdr resume server.

Deleting a Replication Server

You can remove Enterprise Replication from a database server and then remove the database server from an Enterprise Replication domain.

Run the **cdr delete server** command two times to remove Enterprise Replication from a database server, and then remove the database server from an Enterprise Replication domain. The first time, run the command on the server you want to remove Enterprise Replication from. The second time, connect to a different server in the Enterprise Replication domain and run the command, specifying the server you ran the first command on.

To remove Enterprise Replication from an inactive database server, use the **cdr delete server** command with the **--force** option.

To restart Enterprise Replication on a disabled database server, define the server again with the **cdr define server** command and then synchronize data. Row history is deleted when a server has Enterprise Replication removed, so the history is not recoverable if Enterprise Replication is restarted.

Important: If you are creating a replicate to replace the one you deleted, use the **cdr check queue --qname=ctrlq** command to make sure that the delete operation propagated to all the servers.

Examples

To remove Enterprise Replication from local database server **reynolds**, and then remove database server **reynolds** from the Enterprise Replication domain it shares with database server **stimpson**, run the following commands:

```
cdr delete server reynolds
cdr delete server --connect=stimpson reynolds
```

The first command removes Enterprise Replication from the local database server, **reynolds**. The second command connects to database server **stimpson**, which is another server in the Enterprise Replication domain, and then removes database server **reynolds** from the shared domain.

Related reference:
cdr delete server
cdr check queue

Managing Replicates

You can perform the following tasks on existing replicates:

- Modify replicate attributes or participants
- View replicate properties and state
- Change the state of a replicate (whether replication is being performed)
- Delete a replicate

- Modify replicates
  You can modify replicates while replication is active to add or remove participants, or to change some replicate attributes.
- Viewing Replicate Properties
- Starting a Replicate
- Stopping a Replicate
  You can temporarily stop replication for administrative purposes.
- Suspending a Replicate
- Resuming a Suspended Replicate
- Deleting a Replicate
  To delete a replicate from the global catalog, use the **cdr delete replicate** command.

Modify replicates

You can modify replicates while replication is active to add or remove participants, or to change some replicate attributes.

To change other attributes of replicates, you create a new replicate and then delete the original replicate.

- Adding or Deleting Participants
- Change replicate attributes
  You can change many replicate attributes by running the **cdr modify replicate** command.
- Changing the replication key of a replicate
  You can change the replication key of a replicate from the primary key, a unique index or constraint, or the ERKEY shadow columns to another unique index or constraint on your table.

Adding or Deleting Participants

To be useful, a replicate must include at least two participants. You can define a replicate that has fewer than two participants, but before you can use that replicate, you must add more participants.

To add a participant to an existing replicate, use the **cdr change replicate --add** command. For example, to add two participants to the **sales_data** replicate, enter:

```
cdr change replicate --add sales_data \
   "db1@hawaii:jane.table1" "select * from table1" \
   "db2@maui:john.table2" "select * from table2"
```

To delete a participant from the replicate, use the **cdr change replicate --delete** command.

For example, to delete these two participants from the replicate, enter:

```
cdr change replicate --delete sales_data \
   "db1@hawaii:jane.table1" "db2@maui:john.table2"
```

For more information, see cdr change replicate.

Related concepts:
Change replicate attributes
Related tasks:
Changing the replication key of a replicate

Change replicate attributes

You can change many replicate attributes by running the **cdr modify replicate** command.

You can change the following attributes of a replicate:

- Conflict-resolution rules and scope
- Replication frequency
- Error logging
- Replication of full rows or only changed columns
- Database triggers
- Participant type
- Code set conversion
- Serial processing
- Replication of unchanged large objects

You cannot change the conflict resolution from ignore to a non-ignore option (time stamp, SPL routine, or time stamp and SPL routine). You cannot change a non-ignore conflict resolution option to ignore.

For example, to change the replication frequency for the **sales_data** replicate to every Sunday at noon, enter:

```
cdr modify replicate sales_data Sunday.12:00
```

Related tasks:
Adding or Deleting Participants
Changing the replication key of a replicate
Related reference:
cdr modify replicate

Changing the replication key of a replicate

You can change the replication key of a replicate from the primary key, a unique index or constraint, or the ERKEY shadow columns to another unique index or constraint on your table.

To change the replication key of a replicate:

1. Define a new replicate by running the **cdr define replicate** command. Include the **--key** option to specify the new replication key. Do not include the **--erkey** option.
2. Start the new replicate by running the **cdr start replicate** command.
3. Stop the original replicate by running the **cdr stop replicate** command.
4. Delete the original replicate by running the **cdr delete replicate** command.

Related concepts:
Change replicate attributes
Unique key for replication
Related tasks:
Adding or Deleting Participants
Changing or re-creating primary key columns
Related reference:
cdr define replicate

Viewing Replicate Properties

After you define a replicate, you can view the properties of the replicate using the **cdr list replicate** command. If you do not specify the name of a defined replicate on the command line, Enterprise Replication lists detailed information on all the replicates defined on the current server. If you use the **brief** option, Enterprise Replication lists participant information about all the replicates. If you specify a replicate name, Enterprise Replication displays participant information about the replicate.

For information about this command, see cdr list replicate.

Starting a Replicate

When you define a replicate, the replicate does not begin until you explicitly change its state to *active*. When a replicate is active, Enterprise Replication captures data from the logical log and transmits it to the active participants. At least two participants must be active for data replication to occur.
Important: You cannot start replicates that have no participants.
To change the replicate state to active, use the **cdr start replicate** command. For example, to start the replicate **sales_data** on the servers **server1** and **server23**, enter:

```
sales_data server1 server23
```

This command causes **server1** and **server23** to start sending data for the **sales_data** replicate.

If you omit the server names, this command starts the replicate on all servers that are included in that replicate.

When you start a replicate, you can choose to perform an initial data synchronization, as described in Initially Synchronizing Data Among Database Servers.

Warning: Run the **cdr start replicate** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.
When replication is active on an instance, you may need to double the amount of lock resources, to accommodate transactions on replicated tables.

If a replicate belongs to an exclusive replicate set, you must start the replicate set to which the replicate belongs. For more information, see Starting a Replicate.

For more information, see cdr start replicate.

Stopping a Replicate

You can temporarily stop replication for administrative purposes.

To stop the replicate, use the **cdr stop replicate** command. This command changes the replicate state to *inactive* and deletes any data in the send queue for that replicate. When a replicate is inactive, Enterprise Replication does not transmit or process any database changes.

In general, you should only stop replication when no replication activity is likely to occur for that table or on the advice of Software Support. If database activity does occur while replication is stopped for a prolonged period of time, the replay position in the logical log might be overrun. If a message that the replay position is overrun appears in the message log, you must resynchronize the data on the replication servers. For more information on resynchronizing data, see Resynchronizing Data among Replication Servers.

You cannot stop replicates that have no participants.

For example, to stop the **sales_data** replicate on the servers **server1** and **server23**, enter:

```
cdr stop replicate sales_data server1 server23
```

This command causes **server1** and **server23** to purge any data in the send queue for the **sales_data** replicate and stops sending data for that replicate. Any servers not listed on the command line continue to capture and send data for the **sales_data** replicate (even to **server1** and **server23**).

If you omit the server names, this command stops the replicate on all servers that are included in that replicate.

If a replicate belongs to an exclusive replicate set, you must stop the replicate set to which the replicate belongs. For more information, see Exclusive Replicate Sets and Stopping a Replicate Set.

Stopping a replicate set also stops any direct synchronization or consistency checking that are in progress. To complete synchronization or consistency checking, you must rerun the **cdr sync replicateset** or **cdr check replicateset** command.

For more information, see cdr stop replicate.

Suspending a Replicate

If you do not want to completely halt all processing for a replicate, you can suspend a replicate using the **cdr suspend replicate** command. When a replicate is in a suspended state, the replicate captures and accumulates changes to the source database, but does not transmit the captured data to the target database.
Warning: Enterprise Replication does not support referential integrity if a replicate is suspended. Instead, you should suspend a server. For more information, see Suspending Replication for a Server.
For example, to suspend the **sales_data** replicate, enter:

```
cdr suspend replicate sales_data
```

If a replicate belongs to an exclusive replicate set, you must suspend the replicate set to which the replicate belongs. For more information, see Exclusive Replicate Sets and Suspending a Replicate Set.

For more information, see cdr suspend replicate.

Resuming a Suspended Replicate

To return the state of a suspended replicate to active, use the **cdr resume replicate** command. For example:

```
cdr resume replicate sales_data
```

If a replicate belongs to an exclusive replicate set, you must resume the replicate set to which the replicate belongs. For more information, see Exclusive Replicate Sets and Resuming a Replicate Set.

For more information, see cdr resume replicate.

Deleting a Replicate

To delete a replicate from the global catalog, use the **cdr delete replicate** command.

When you delete a replicate, Enterprise Replication purges all replication data for the replicate from the send queue at all participating database servers.

For example, to delete the **sales_data** replicate from the global catalog, enter:

```
cdr delete replicate sales_data
```

Important: If you are creating a replicate to replace the one you deleted, use the **cdr check queue --qname=ctrlq** command to make sure that the delete operation propagated to all the servers.
Related reference:
cdr check queue
cdr delete replicate

Managing Replicate Sets

When you create a replicate set, you can manage the replicates that belong to that set together or individually. If the replicate set is exclusive, you can only manage the individual replicates as part of the set.

Performing an operation on a replicate set (except **cdr delete replicateset**) is equivalent to performing the operation on each replicate in the replicate set individually.

For more information, see Managing Replicates.

- Connection management for client connections to participants in a replicate set
  You can configure Connection Managers to route connection requests from clients to the replication servers of a replicate set.
- Modifying Replicate Sets
- Viewing Replicate Sets
- Starting a Replicate Set
- Stopping a Replicate Set
- Suspending a Replicate Set
- Resuming a Replicate Set
- Deleting a Replicate Set
  To delete a replicate set, use the **cdr delete replicateset** command.

Connection management for client connections to participants in a replicate set

You can configure Connection Managers to route connection requests from clients to the replication servers of a replicate set.

Connection requests can be directed to replication servers based on Connection Manager service-level agreements (SLAs). You can configure Connection Manager SLAs to redirect connection requests based on various redirection policies. Connection Managers support the following redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.
- ROUNDROBIN: Connection requests are directed in a repeating, ordered fashion (round-robin) to a group of replication servers.
- WORKLOAD: Connection requests are directed to the replication server that has the lowest workload.

Related information:
Connection management through the Connection Manager
Example of configuring connection management for a grid or replicate set

Modifying Replicate Sets

You can modify replicate sets in two ways:

- Add or Delete Replicates
- Change Replication Frequency

- Adding or Deleting Replicates From a Replicate Set
- Changing Replication Frequency For the Replicate Set

Adding or Deleting Replicates From a Replicate Set

To add a replicate to an existing replicate set, use the command **cdr change replicateset --add**. For example, to add two replicates to **sales_set**, enter:

```
cdr change replicateset --add sales_set sales_kauai \
   sales_moorea
```

The state of the replicate when you add it to a replicate set depends on the type of replicate set:

- For a non-exclusive replicate set, the state of the new replicate remains as it was when you added it to the set. To bring all the replicates in the non-exclusive set to the same state, use one of the commands described in Managing Replicate Sets.
- For an exclusive replicate set, Enterprise Replication changes the existing state and replication frequency settings of the replicate to the current properties of the exclusive replicate set.

To delete a replicate from the replicate set, use **cdr change replicate --delete**.

For example, to delete the two replicates, **sales_kauai** and **sales_moorea**, from the replicate set, enter:

```
cdr change replicateset --delete sales_set sales_kauai\
   sales_moorea
```

When you add or remove a replicate from an exclusive replicate set that is suspended or that is defined with a frequency interval, Enterprise Replication transmits all the data in the queue for the replicates in the replicate set up to the point when you added or removed the replicate. For more information, see Suspending a Replicate Set and Frequency Options.

For more information, see cdr change replicateset.

Related tasks:
Changing Replication Frequency For the Replicate Set

Changing Replication Frequency For the Replicate Set

You can change the replication frequency for the replicates in an exclusive or non-exclusive replicate set using the **cdr modify replicateset** command. For more information, see Specifying Replication Frequency.

For example, to change the replication frequency for each of the replicates in the **sales_set** to every Monday at midnight, enter:

```
cdr modify replicateset sales_set Monday.24:00
```

For more information, see cdr change replicateset.

Related tasks:
Adding or Deleting Replicates From a Replicate Set

Viewing Replicate Sets

To view the properties of the replicate set, use the **cdr list replicateset** command. The **cdr list replicateset** command displays the replicate set name and a list of the replicates that are members of the set. To find out more about each replicate in the replicate set, see Viewing Replicate Properties.

For more information, see cdr list replicateset.

Starting a Replicate Set

To change the state of all the replicates in the replicate set to active, use the **cdr start replicateset** command. For example, to start the replicate set **sales_set**, enter:

```
set sales_set
```

When you start a replicate set, you can choose to perform an initial data synchronization, as described in Initially Synchronizing Data Among Database Servers.

Warning: Run the **cdr start replicateset** command on an idle system (when no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement after you successfully start the replicate.
For more information, see cdr start replicateset and cdr start replicate.

Stopping a Replicate Set

To stop the replicates in the replicate set, use the **cdr stop replicateset** command. This command changes the state of all the replicates in the set to inactive.

For example, to stop the **sales_set** replicate set, enter:

```
cdr stop replicateset sales_set
```

Stopping a replicate set also stops any direct synchronization or consistency checking that are in progress. To complete synchronization or consistency checking, you must rerun the **cdr sync replicateset** or **cdr check replicateset** command.

For more information, see cdr stop replicateset and cdr stop replicate.

Suspending a Replicate Set

If you do not want to completely halt all processing for the replicates in a replicate set, you can suspend the replicates in the set using the **cdr suspend replicateset** command.

For example, to suspend the **sales_set** replicate set, enter:

```
cdr suspend replicateset sales_set
```

For more information, see cdr suspend replicateset and cdr suspend replicate.

Related reference:
cdr change replicateset

Resuming a Replicate Set

To return the suspended replicates in the replicate set to active, use the **cdr resume replicateset** command. For example:

```
cdr resume replicateset sales_set
```

For more information, see cdr resume replicateset and cdr resume replicate.

Deleting a Replicate Set

To delete a replicate set, use the **cdr delete replicateset** command.

Tip: When you delete a replicate set, Enterprise Replication does not delete the replicates that are members of the replicate set. The replicates remain in the state they were in when the set was deleted.
For example, you can connect to the default database server specified by the INFORMIXSERVER environment variable and delete the **sales_set** replicate set by using running the following command:

```
cdr delete replicateset sales_set
```

Important: If you are creating a replicate to replace the one you deleted, use the **cdr check queue --qname=ctrlq** command to make sure that the delete operation propagated to all the servers.
Related reference:
cdr check queue
cdr delete replicateset

Managing Templates

You can use the **cdr list template** and **cdr delete template** commands to view information about your templates and to clean up obsolete templates. The commands are described in detail, including examples and sample output, in The cdr utility.

You cannot update a template. To modify a template, you must delete it with the **cdr delete template** command and then re-create it with the **cdr define template** command.

- Viewing Template Definitions
- Deleting Templates

Viewing Template Definitions

Use the **cdr list template** command to view detailed information about the template and the servers, databases and tables for which the template defines replication.

Related tasks:
Deleting Templates

Deleting Templates

Use the **cdr delete template** command to delete any templates that you no longer want to use to set up replication. The command also deletes any replicate sets associated with the template which exist if the template has been realized.

Important: Deleting a template does not delete replicates that have been created by realizing a template.
Related tasks:
Viewing Template Definitions

Managing Replication Server Network Connections

This section explains how you can view network connections status, drop network connections, and reestablish dropped network connections.

- Viewing Network Connection Status

Viewing Network Connection Status

To determine the current status of the network connection to each of the servers participating in replication, use the **cdr list server** command and look at the STATUS column of the output.

For more information, see cdr list server.

Dropping the Network Connection

To drop the Enterprise Replication network connection for a server, use the **cdr disconnect server** command. When you drop the connection, Enterprise Replication continues to function and queue transactions. For example, to disconnect the network connection between the current replication server and the server **g_papeete**, enter:

```
cdr disconnect server g_papeete
```

Warning: When you disconnect a server from Enterprise Replication, you must ensure that the send queues on **all** other Enterprise Replication servers participating in replication do not fill.
For more information, see cdr disconnect server.

Reestablishing the Network Connection

To reestablish a dropped network connection, use the **cdr connect server** command.

For example, to reestablish the network connection between the current replication server and the server **g_papeete**, enter:

```
cdr connect server g_papeete
```

The following conditions can cause reestablishing a network connection to fail:

- A network outage
- A server is offline
- The **cdr stop**, **cdr disconnect server**, or **cdr delete server** commands were run on a server
- The system clock times on the servers differ by more than 900 seconds

For more information, see cdr connect server.

Resynchronizing Data among Replication Servers

If replication has failed for some reason and data is not synchronized, there are different ways to correct data mismatches between replicated tables.

The following table compares each of the methods. All methods except manual table unloading and reloading can be performed while replication is active.

Table 1. Resynchronization methods

Method	Description
Direct synchronization	• Replicates all rows from the specified reference server to all specified target servers for a replicate or replicate set. • Runs as a foreground process by default, but can run as a background process. • Populates tables in a new participant. • Quickly synchronizes significantly inconsistent tables when used with the TRUNCATE statement.
Checking consistency and then repairing inconsistent rows	• Compares all rows from the specified target servers with the rows on the reference server, prepares a consistency report, and optionally repairs inconsistent rows. • Runs as a foreground process by default, but can run as a background process.
ATS or RIS file repairs	• Used to repair rows that other synchronization methods could not repair. • Repairs a single transaction at a time. • Replicates or replication server must have been configured with the ATS or RIS option.

Method	Description
Manual table unloading and reloading	• Manual process of unloading the target table, copying the reference table, and then loading the reference table into the target database. • Requires that replication be suspended.

- Performing Direct Synchronization
  Direct synchronization replicates every row in the specified replicate or replicate set from the reference server to all the specified target servers. You can use direct synchronization to populate a new target server, or an existing target server that has become severely inconsistent.
- Checking Consistency and Repairing Inconsistent Rows
  A consistency check compares the data between a reference server and one or more target servers and then generates a report that describes any inconsistencies. You can choose to repair inconsistent rows during a consistency check.
- Repairing Failed Transactions with ATS and RIS Files
  You can repair failed or inconsistent transactions using an ATS or RIS file if you defined the replicate or replication server with the **–ats** or **–ris** option and the ATS or RIS files are being generated in text format.
- Resynchronize data manually
  Manual resynchronization involves replacing the inconsistent table in the target database with a copy of the correct table from the reference database.

Related concepts:
Repair and Initial Data Synchronization
Related reference:
cdr stop
cdr stop replicate

Performing Direct Synchronization

Direct synchronization replicates every row in the specified replicate or replicate set from the reference server to all the specified target servers. You can use direct synchronization to populate a new target server, or an existing target server that has become severely inconsistent.

- The Enterprise Replication network connection must be active between the Connect server, reference server and the target servers while performing direct synchronization.
- The replicate must not be in a suspended or stopped state during direct synchronization.
- The replicate must not be set up for time based replication.

You can synchronize a single replicate or a replicate set. When you synchronize a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables). You can choose how to handle extra target rows and whether to enable trigger firing on target servers.

Important: Running direct synchronization can consume a large amount of space in your log files. Ensure you have sufficient space before running this command.
To perform direct synchronization, use the **cdr sync replicate** or **cdr sync replicateset** command.

You can monitor the progress of a synchronization operation with the **cdr stats sync** command if you provide a progress report task name in the **cdr sync replicate** or **cdr sync replicateset** command.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the **--name** option so that you can monitor the operation with the **cdr stats sync** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

You can significantly improve the performance of synchronizing a replicate set by synchronizing the member replicates in parallel. You specify the number of parallel processes with the **--process** option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

If direct synchronization cannot repair a row, the inconsistent row is recorded in an ATS or RIS file.

- Synchronizing Significantly Inconsistent Tables
  If your target tables are significantly inconsistent, you can speed the synchronization process by truncating the target tables before you perform direct synchronization.

Related tasks:
Repairing Failed Transactions with ATS and RIS Files
Related reference:
cdr sync replicate
cdr sync replicateset
cdr stats sync

Synchronizing Significantly Inconsistent Tables

If your target tables are significantly inconsistent, you can speed the synchronization process by truncating the target tables before you perform direct synchronization.

When you truncate a table by using the TRUNCATE statement, you remove all rows from the table while replication is active. After the tables on the target servers are empty, direct synchronization efficiently applies data from the source server to the target servers.

If you use the TRUNCATE statement on the supertable in a hierarchy, by default, rows in all the subtables are deleted as well. You can use the ONLY keyword to limit the truncate operation to the supertable. For more information on the TRUNCATE statement, see the *IBM® Informix® Guide to SQL: Syntax*.

To synchronize tables in conjunction with truncation:

1. Run the TRUNCATE statement on the tables to be synchronized on the target servers.
2. Run the **cdr sync replicate** or **cdr sync replicateset** command.

For the syntax of these commands, see cdr sync replicate and cdr sync replicateset.

Checking Consistency and Repairing Inconsistent Rows

A consistency check compares the data between a reference server and one or more target servers and then generates a report that describes any inconsistencies. You can choose to repair inconsistent rows during a consistency check.

The following conditions apply when you check consistency:

- Running a consistency check can consume a large amount of space in your log files. Ensure you have sufficient space before checking consistency.
- The Enterprise Replication network connection must be active between the Connect server, reference server and the target servers while performing consistency checking and repair.
- The replicate must not be in a suspended or stopped state during consistency checking.
- The replicate must not be set up for time based replication.

You can perform a consistency check and optional synchronization on a single replicate or a replicate set. When you synchronize a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables). You can choose how to handle extra target rows and whether to enable trigger firing on target servers.

To perform a consistency check, use the **cdr check replicate** or **cdr check replicateset** command. Use the **--repair** option to repair the inconsistent rows. A consistency report is displayed for your review.

You can monitor the progress of a consistency check with the **cdr stats check** command if you provide a progress report task name in the **cdr check replicate** or **cdr check replicateset** command.

You can run a consistency check as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the **--name** option so that you can monitor the check with the **cdr stats check** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database. If you use the **--background** option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

- Interpreting the Consistency Report
  The consistency report displays information about differences in replicated data within the replicate or replicate set.
- Increase the speed of consistency checking
  You can increase the speed of checking the consistency of replicates or replicate sets with the **cdr check replicate** or **cdr check replicateset** commands in several ways.
- Repair inconsistencies by time stamp
  You can repair inconsistencies based on the latest time stamps among the participants instead of specifying a master server.
- Repairing inconsistencies while enabling a replication server
  If a replication server is in disabled mode, you can enable it and repair inconsistencies with the **cdr check replicateset** command.
- Implementing a custom checksum function
  You can implement a custom checksum function for consistency checking if you do not want to use the checksum function that is included with the database server.

If synchronization during a consistency check cannot repair a row, the inconsistent row is recorded in an ATS or RIS file.

Related tasks:
Repairing Failed Transactions with ATS and RIS Files
Indexing the ifx_replcheck Column
Related reference:
cdr check replicate
cdr check replicateset
cdr stats check

Interpreting the Consistency Report

The consistency report displays information about differences in replicated data within the replicate or replicate set.

Inconsistencies listed in the consistency report do not necessarily indicate a failure of replication. Data on different database servers is inconsistent while replicated transactions are in progress. For example, the following consistency report indicates that two rows are missing on the server **g_serv2**:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         0
g_serv2                 65         0         2         0         0
```

```
WARNING: replicate is not in sync

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

The missing rows could be in the process of being replicated from **g_serv1** to **g_serv2**.
If you choose to repair inconsistent rows during a consistency check, the report shows the condition of the replicate at the time of the check, plus the actions taken to make the replicate consistent. For example, the following report shows two missing rows on **g_serv2** and that two rows were replicated from **g_serv1** to correct this inconsistency:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         2
g_serv2                 65         0         2         0         0

Validation of repaired rows failed.
WARNING: replicate is not in sync

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

The warning indicates that inconsistencies were discovered.
The report indicates whether the replicate became consistent after the repair process. In this example, the `Validation of repaired rows failed.` message indicates that the replicate is not consistent. This might occur because some replicated transactions were still being replicated. Use the **--inprogress** option to extend the validation time.

The verbose form of the consistency report also displays the differing values for each inconsistent row.

For more information about the contents of the consistency report, see cdr check replicate.

Related reference:
cdr check replicate

Increase the speed of consistency checking

You can increase the speed of checking the consistency of replicates or replicate sets with the **cdr check replicate** or **cdr check replicateset** commands in several ways.

To increase the speed of consistency checking of replicate sets by checking the member replicates in parallel, use the **--process** option to set the number of parallel processes equal to the number of replicates.

To increase the speed of consistency checking by limiting the amount of data that is checked, use one or more of the following options:

- Skip the checking of large objects with the **--skipLOB** option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.
- Check from a specific time with the **--since** option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.
- When checking a replicate, you can check a subset of the data with the **--where** option.

If you have large tables, you can index the **ifx_replcheck** shadow column.

- Indexing the ifx_replcheck Column
  You can index the **ifx_replcheck** shadow column to increase the speed of consistency checking.

Related reference:
cdr check replicateset
cdr check replicate

Indexing the ifx_replcheck Column

You can index the **ifx_replcheck** shadow column to increase the speed of consistency checking.

If you have a large replicated table, you can add the **ifx_replcheck** shadow column and then create a new unique index on that column and the existing replication key columns. The index on the **ifx_replcheck** shadow column allows the database server to determine whether rows in different tables have different values without comparing the values in those rows. You must create the index on the table in each database server that participates in the replicate.

Before you can create an index on the **ifx_replcheck** shadow column and the replication key columns, you must prepare the replicated table by adding the **ifx_replcheck** shadow column. You can add the **ifx_replcheck** shadow column when you create the table with the WITH REPLCHECK clause, or you can alter an existing table to add the **ifx_replcheck** shadow column with the ADD REPLCHECK clause.

You can create the index while replication is active.
To index the **ifx_replcheck** shadow column, create a unique index based on the existing replication key columns and the **ifx_replcheck** column. The **ifx_replcheck** shadow column must be the last column in the index.
For example, the following statement creates an index on a table named customer on the primary key column **id** and **ifx_replcheck**:

```
CREATE UNIQUE INDEX customer_index ON  customer(id, ifx_replcheck);
```

Related concepts:
Preparing Tables for a Consistency Check Index
Related tasks:
Checking Consistency and Repairing Inconsistent Rows
Related reference:
cdr check replicate
cdr check replicateset

Repair inconsistencies by time stamp

You can repair inconsistencies based on the latest time stamps among the participants instead of specifying a master server.

If your replicates use the time stamp or delete wins conflict resolution rule, you can repair inconsistencies between the participants based on the latest time stamp on any participant. If you run a time stamp repair, you do not specify a master server whose data is considered correct and to which all the other participants are matched.

To ensure that a time stamp repair is accurate, follow these guidelines:

- When you need to temporarily stop replication on a server, disable it with the **cdr disable server** command instead of stopping it with **cdr stop** command.
- If you are using the delete wins conflict resolution rule, set the CDR_DELAY_PURGE_DTC configuration parameter on all replication servers to the maximum age of modifications to rows that are being actively updated.

To run a time stamp repair, use the **cdr check replicate** or **cdr check replicateset** command with the **--repair** and **--timestamp** options. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option.

If a time stamp repair finds an extra row on any participant, the result depends on the conflict resolution rule and the last transaction for that row:

- If the conflict rule is time stamp and the most recent time stamp for the row is a delete transaction, the row will be deleted on all servers.
- If the conflict rule is time stamp and a participant has a deleted row but the most recent time stamp for that row is an update transaction, the updated row is replicated to all servers.
- If the conflict rule is delete wins and any participant has deleted that row, the row is deleted from all servers, regardless of any later update transactions.

If a time stamp repair finds mismatched rows on different servers, then the most recent update transaction for that row is replicated to the other server.

Related concepts:
Time stamp conflict resolution rule
Delete wins conflict resolution rule
Related reference:
CDR_DELAY_PURGE_DTC configuration parameter

Repairing inconsistencies while enabling a replication server

If a replication server is in disabled mode, you can enable it and repair inconsistencies with the **cdr check replicateset** command.

The server must have been put in disabled mode with the **cdr disable server** command.
To enable a disabled server and synchronize it, run the **cdr check replicateset** command with the **--repair** and **--enable** options.
By default, the enable process times out after 128 seconds if the disabled replication server cannot be enabled and repaired during that time. You can specify a shorter time out period by setting the **--timeout** option to a value less than or equal to 60 seconds.

To repair all replicate sets on the disabled server, also include the **--allrepl** option and omit the **--replset** option.

Related reference:
cdr check replicateset

Implementing a custom checksum function

You can implement a custom checksum function for consistency checking if you do not want to use the checksum function that is included with the database server.

The $INFORMIXDIR/extend/checksum directory contains sample checksum function code and registration statements.
To implement a custom checksum function:

1. Using the idschecksum.c file in the $INFORMIXDIR/demo/checksum directory as a template, write a C language function that creates a checksum. Overload the function for each of the supported data types.
2. Compile the function code into a shared object file.
3. Save a copy of the shared object file in the $INFORMIXDIR/extend/checksum directory on all replication servers.
4. To register the function:
    a. Modify the idschecksum.sql file in the $INFORMIXDIR/demo/checksum directory to include the name of your function.
    b. Run the SQL statements on each replication server.

Specify your checksum function name with the **--checksum** option when you run the **cdr check replicate** or **cdr check replicateset** command.
If the **cdr check replicate** or **cdr check replicateset** command fails with return code 172, your checksum function is not installed and registered on all replication servers.

- Rules for custom checksum functions
  The idschecksum.c file in the $INFORMIXDIR/demo/checksum directory contains code that you can use for your checksum functions. You can replace the checksum generation portion of the code with your custom code.

Rules for custom checksum functions

The idschecksum.c file in the $INFORMIXDIR/demo/checksum directory contains code that you can use for your checksum functions. You can replace the checksum generation portion of the code with your custom code.

A checksum function summarizes the data in a replicated row. During consistency checking, the checksum values of corresponding rows on different replication servers are compared to determine whether the rows are consistent.

A checksum function runs recursively through every column in the replicated table to generate a checksum value for a replicated row. A checksum value is generated for the last column in the table. The checksum value is used to calculate the checksum value for the previous column, and so on. The checksum value that is calculated for the first column in the table is based on the accumulated checksum value of all the other columns.

Custom checksum functions must conform to the following rules:

- The first parameter of the function is the data type of a column.
- The second parameter of the function is an integer that is the checksum value of the previous column.
- The function returns an integer.
- The function is a DBA function.
- The function attributes include NOT VARIANT, HANDLESNULLS, and PARALLELIZABLE.

You must register a checksum function for each of the following data types, regardless of whether the data types are used in your replicated tables. All other data types are ignored by checksum functions.

- BLOB
- BYTE
- CLOB
- DATE
- DATETIME YEAR TO FRACTION
- DATETIME YEAR TO MONTH
- DECIMAL
- FLOAT
- INT8
- INTEGER
- LIST
- LVARCHAR (includes all character data types)
- MONEY
- MULTISET
- REAL
- ROW
- SET
- SMALLINT
- TEXT

However, if you do not want to create a checksum value for certain data types, you can provide non-operative function definitions. For example, you might not want to create checksum values for BLOB columns. The following statement registers a checksum function for the BLOB data type that returns the previous checksum value instead of calculating an accumulated checksum value:

```
CREATE DBA FUNCTION ercheck_checksum(p1 blob, p2 integer)
      RETURNS integer;
RETURN p2;
END FUNCTION;
```


Repairing Failed Transactions with ATS and RIS Files

You can repair failed or inconsistent transactions using an ATS or RIS file if you defined the replicate or replication server with the **–ats** or **–ris** option and the ATS or RIS files are being generated in text format.

A repair using an ATS or RIS file repairs the rows associated with the single transaction that is recorded in the specified ATS or RIS file. To apply repairs based on an ATS or RIS file, use the **cdr repair** command.

Note: The **cdr repair** command is not supported for replicates that are defined with the `--UTF8=y` option. For replicates that are defined with the `--UTF8=y` option, use the **cdr check replicate --repair** or **cdr check replicateset --repair** command to repair data.
The **cdr repair** command processes one ATS or RIS file each time you specify the command. The following table shows how failed operations are handled.

Failed Operation	Action Taken
Delete	Delete on the target server

Failed Operation	Action Taken
Insert or Update	• If the row is found on the source server, does an update • If the row is not found on the source server, but is found on the target server, does a delete on the target server. If the row is not found on either server, performs no action.

Each operation is displayed to stderr, unless you use the **–quiet** option with the **cdr repair** command. You can preview the operations without performing them by using the **–check** option with the **cdr repair** command.

Related concepts:
Failed Transaction (ATS and RIS) Files
The cdr utility
Repair and Initial Data Synchronization
Related tasks:
Performing Direct Synchronization
Checking Consistency and Repairing Inconsistent Rows

Resynchronize data manually

Manual resynchronization involves replacing the inconsistent table in the target database with a copy of the correct table from the reference database.

Important: Manual resynchronization is not recommended for repairing your replicated tables because you must suspend replication to avoid producing further inconsistencies.
The following example shows how to manually resynchronize two replication database servers.

To synchronize the replication server **g_papeete** with the server **g_raratonga**:

1. Suspend replication to the replication server group **g_papeete.**
   See Suspending Replication for a Server.
2. Unload the table from the server group **g_raratonga.**
   See Load and unload data.
3. Load the table on **g_papeete** and specify BEGIN WORK WITHOUT REPLICATION.
   See Load and unload data and Blocking Replication.
4. Resume replication to **g_papeete**.
   See Resuming a Suspended Replication Server.

Alter, rename, or truncate operations during replication

When Enterprise Replication is active and data replication is in progress, you can perform many types of alter, rename, or truncate operations on replicated tables and databases.

Most of the supported operations do not require any special steps when performed on replicated tables or databases; some, however, do require special steps. None of the supported alter, rename, or truncate operations are replicated. You must perform these operations on each replicate participant.

You can run the alter, rename, and truncate operations that are listed in the following table on active, replicated tables or databases without performing extra steps.

Table 1. Requirements for operations on replicated tables

Operation	Requirements
Add or drop default values and SQL checks	None
Add or drop fragments	Requires mastered replicate to be defined
Add or drop unique, distinct, and foreign keys	None
Alter the locking granularity	None
Alter the next extent size	None
Change an existing fragment expression on an existing dbspace	Requires mastered replicate to be defined
Convert a fragmented table to a non-fragmented table	Requires mastered replicate to be defined
Convert a non-fragmented table to a fragmented table	Requires mastered replicate to be defined
Convert from one fragmentation strategy to another	Requires mastered replicate to be defined
Create a clustered index	Requires mastered replicate to be defined
Modify the data type of a replicated column	Requires mastered replicate to be defined
Modify the data type of a replicated column in a multiple-column replication key	Requires mastered replicate to be defined
Move a fragment expression from one dbspace to another dbspace	Requires mastered replicate to be defined
Move a non-fragmented table from one dbspace to another dbspace	Requires mastered replicate to be defined

Operation	Requirements
Recluster an existing index	Requires mastered replicate to be defined
Rename a database	None
Rename a replicated column	Requires non-strict mastered replicate to be defined
Rename a table	Requires non-strict mastered replicate to be defined
Truncate a replicated table	Requires mastered replicate to be defined

You can perform the following alter operations on active, replicated tables, but you must perform extra steps, which are described in following sections:

- Add a column to a replicated table
- Remove a column from replication
- Attach a fragment to a replicated table
- Change or recreate a replication key

Enterprise Replication uses shadow replicates to manage alter operations on replicated tables without causing any interruption to replication. By using shadow replicates, the replicate participants SELECT clause can be modified while replication is active. For example, a new column can be brought into the replicate definition, an existing replicated column can be removed from the replicate definition and the data type or size of a replicated column can be changed without interrupting replication. See Defining Shadow Replicates for more information about shadow replicates.

Before altering a replicated table, ensure that you have sufficient log space allocated for long transactions, a sufficient number of locks available, and sufficient space available for the queue sbspace.

When you issue a command to alter a replicated table, Enterprise Replication places the table in *alter* mode before performing the alter operation. Alter mode is a state in which only DDL (data-definition language) and SELECT operations are allowed but DML (data-manipulation language) operations are not allowed. After the transaction that initiated the alter operation completes, Enterprise Replication unsets alter mode. Any schema changes are automatically applied to any delete tables.

The following restrictions apply when you use alter operations on replicated tables.

- Enterprise Replication must be in an active state, unless you are only adding or dropping check constraints and default values.
- Tables must have a master replicate defined.
- The DROP TABLE statement is not supported.

Recommendation: If you need to perform more than one alter operation, enclose them in a single transaction so that alter mode only needs to be set and unset one time. For a list of common alter operation problems and how to solve them, see Troubleshooting Tips for Alter Operations.

- Altering multiple tables in a replicate set
  You can alter multiple replicated tables for replicates that belong to the same replicate set and then remaster those tables as a group.
- Adding a Replicated Column
- Removing replicated columns
  You can alter replicated tables to remove columns from replication.
- Modifying the data type or size of a replicated column
  You can modify the size or type of a replicated column for all basic data types and for the BOOLEAN and LVARCHAR extended types. Modifying the data type or size of columns of other extended types is not supported. The replicate must be a master replicate.
- Changing the Name of a Replicated Column, Table, or Database
- Changing or re-creating primary key columns
  You can change or re-create the primary key columns definition of a replicated table while replication is active.
- Attaching a New Fragment to a Replicated Table
  You can attach a new fragment to a replication table while replication is active.
- Remastering a Replicate
  You must remaster a replicate if you add a replicated column, drop a replicated column, or change a classic replicate into a mastered replicate. If you modify a replicated column, you can remaster, but remastering is not mandatory.

Related concepts:
SQL statements and replication
Related reference:
cdr alter

Altering multiple tables in a replicate set

You can alter multiple replicated tables for replicates that belong to the same replicate set and then remaster those tables as a group.

Instead of remastering the replicates individually for each table that you alter, you create a derived replicate set that contains only the replicates that must be remastered. You do not specify the names of the replicates. The server identifies which replicates have tables that must be remastered and adds them to the derived set for you. After you remaster and synchronize the derived replicate set, you delete it.
To alter replicated tables in a replicate set:

1. Run the ALTER operation on the tables on all replication servers. If the replicate set is included in a grid, you can alter the tables on one server and propagate the changes to all other servers.
2. Create a derived replicate set by running the **cdr define replicateset** command with the **--needRemaster** option.
3. Remaster the tables in the derived replicate set by running the **cdr remaster replicateset** command. The replicate definitions are updated in the global catalogs of the replication servers.
4. Synchronize the derived replicate set by running the **cdr check replicateset** with the **--repair** option or by running the **cdr sync replicateset** command.
5. Drop the derived replicate set by running the **cdr delete replicateset** command.

Related reference:
Example of rolling out schema changes in a grid

cdr define replicateset
cdr check replicateset
cdr sync replicateset
cdr remaster replicateset
cdr delete replicateset

Adding a Replicated Column

You can alter a replicated table to add a new column to be replicated. The replicate must be a master replicate.

To add a new replicated column

1. Use the ALTER TABLE statement to add the column to the replicated table at all participating nodes.
2. Remaster the replicate to include the newly added column in the replicate definition, as described in Remastering a Replicate.

Removing replicated columns

You can alter replicated tables to remove columns from replication.

The replicates must be master replicates.
To remove one or more replicated columns from one or more replicates, run the **cdr remaster** command with the **--remove** option. You specify the database, table, and column names instead of the replicate names.

After you remove columns from replication, you can drop the columns.
Related tasks:
Adding an existing replicate to a grid replicate set by altering a table
Related reference:
cdr remaster

Modifying the data type or size of a replicated column

You can modify the size or type of a replicated column for all basic data types and for the BOOLEAN and LVARCHAR extended types. Modifying the data type or size of columns of other extended types is not supported. The replicate must be a master replicate.

When you modify a replicated column, do not insert data into the modified column that does not fit into the old column definition until all participants are altered, because the data might be truncated or data conversion to and from the master dictionary format to the local dictionary format might fail. Enterprise Replication handles the data type mismatch by having the source server convert data that is in the local dictionary format to the master dictionary format, and the target server convert data from the master dictionary format to the local dictionary format. If Enterprise Replication detects a mismatch in data type or size between the master replicate definition and the local table definition, a warning is printed in the log file.

If Enterprise Replication is not able to convert the replicated row data into the master dictionary format on the source server while queuing replicated data into the send queue, the replicate is stopped for the local participant. If the replicate is stopped, you must correct the problem and then restart the replicate from the local participant with the **--syncdatasource** option. If the correction is to delete the problematic row data, delete the row by running the BEGIN WORK WITHOUT REPLICATION statement. Otherwise, the deleted row is moved from the replicated table to the associated delete table, which might cause problems for the subsequent alter operation on the replicated table.

If Enterprise Replication cannot convert row data from the master dictionary format to local table dictionary format at the target server after receiving replicated data, the replicated transaction is spooled to ATS and RIS files. For example, if you modify a SMALLINT column to an INTEGER column, make sure that you do not insert data that is too large for the SMALLINT data type until the alter operation is performed at all replicate participants, and remastering is performed so that the master dictionary reflects the INTEGER data type.

Important: While modifying a replicated column, sometimes it is possible that the alter operation on the base table succeeds, but the delete table modification might fail when Enterprise Replication unsets alter mode. If the delete modification fails, you see a message similar to the following in the server message log file:

```
CDRGC: cannot populate data into the new delete table
SQL error=-1226, ISAM error=0
```

This situation can happen while modifying a replicated column from a data type larger in length or size to a data type smaller in length or size, for example, from an INTEGER column to a SMALLINT column, and if the delete table has data which cannot fit in the new type column.
To avoid this situation, do not convert between data types that cause data truncation or produce cases where data cannot fit into the new type. If the above situation has already occurred, carefully update or delete the problematic rows from the delete table and attempt to unset alter mode manually by using the **cdr alter** command. If you cannot resolve the problem, contact Software Support.

To modify a replicated column:

1. Issue the alter command to modify the replicated column.
2. Perform the alter operation at all the replicate participants.
3. Optionally remaster the replicate to update the column definition in the replicate definition, as described in Remastering a Replicate.

After an alter operation, the master dictionary no longer matches the replicated table dictionary. Because data transfer is always done in master dictionary format, data conversion between the local dictionary format and the master dictionary format is performed. Data conversion can slow the performance of your replication system. The remastering process changes the master dictionary to match the altered replicated table dictionary. Therefore, after remastering, data conversion is not necessary.

Replication keys have special considerations. For more information, see Changing or re-creating primary key columns.

Changing the Name of a Replicated Column, Table, or Database

You can change the name of a replicated column, table, or database while replication is active. The replicate must be a master replicate.

To change the name of a replicated column, table, or database, run the SQL statement RENAME COLUMN, RENAME TABLE, or RENAME DATABASE on all participants in the replicate.

Related information:
RENAME TABLE statement
RENAME COLUMN statement
RENAME DATABASE statement

Changing or re-creating primary key columns

You can change or re-create the primary key columns definition of a replicated table while replication is active.

You can change the primary key columns without restriction if either of the following conditions are true:

- The table uses ERKEY shadow columns or another unique index or constraint as the replication key.
- The primary key contains multiple columns. The column modification implicitly re-creates the primary key.

To change a primary key column if the primary key is a single column, enclose the primary key column modification and the primary key recreation operations in a single transaction. If you frequently update a primary key that is a single column, consider changing the replication key to another unique index or constraint.

To drop and re-create a primary key:

1. Set alter mode by running the **cdr alter on** command.
2. Drop the primary key columns.
3. Create the new primary key columns.
4. Unset alter mode by running the **cdr alter off** command.

Related concepts:
SQL statements and replication
Related tasks:
Changing the replication key of a replicate

Attaching a New Fragment to a Replicated Table

You can attach a new fragment to a replication table while replication is active.

Enterprise Replication cannot automatically set alter mode for this operation because of an SQL restriction that requires attaching a fragment to be performed in multiple steps.

To attach a new fragment to a replicated table:

1. Set alter mode on the replicate by running the **cdr alter on** command.
2. Drop the replication key of the table.
3. Attach the new fragment.
4. Re-create the replication key.
5. Unset alter mode by running the **cdr alter off** command.

Related tasks:
Preparing tables without primary keys

Remastering a Replicate

You must remaster a replicate if you add a replicated column, drop a replicated column, or change a classic replicate into a mastered replicate. If you modify a replicated column, you can remaster, but remastering is not mandatory.

To redefine an existing master replicate that is defined with name verification, or turn an existing classic replicate into a master replicate, run the **cdr remaster** command.

If the master replicate does not include name verification, you manually remaster the replicate.

- Remastering replicates without name verification
  You manually remaster replicates if the participants to not have matching column names and replicate has name verification turned off by the **--name=n** option of the **cdr define replicate** command.

Related reference:
cdr remaster

Remastering replicates without name verification

You manually remaster replicates if the participants to not have matching column names and replicate has name verification turned off by the **--name=n** option of the **cdr define replicate** command.

To manually remaster a replicate:

1. Use the **cdr define replicate** command to create a shadow replicate with the same attributes as the primary replicate and with the **--mirrors** option, but with a SELECT statement that is correct for the table after the alter operation. The SELECT statement can include newly added columns or omit newly dropped columns.
2. Use the **cdr swap shadow** command to exchange the existing primary replicate and the newly created shadow replicate.

While performing the **cdr swap shadow** operation, Enterprise Replication stores the BEGIN WORK position of the last known transaction sent to the grouper as a *swap log position* for the current swap operation. Any transaction that is begun before the swap log position uses the original replicate definition. Any transaction that is begun after the swap log position uses the new replicate definition.

The old replicate definition is deleted automatically after the replicate definition is no longer required by Enterprise Replication.

Related reference:
cdr swap shadow

Recapture replicated transactions

If you want a transaction to continue to be replicated after it reaches the target replication servers, you can use the **ifx_set_erstate()** procedure.

By default, when Enterprise Replication reads the logical logs to capture transactions, replicated transactions are ignored. For example, if a transaction is replicated from **serv1** to **serv2**, that transaction is not captured for replication on **serv2** because it has already been replicated. Replication stops when transactions reach target servers, but you can configure a transaction to be recaptured and continue to be replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Example

Suppose that a retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory and then replicates that summary information to a central server. A stored procedure named **low_inventory()** that creates a low inventory report exists on all replications servers. The following example creates a new procedure named **xqt_low_inventory()** that enables replication for the **low_inventory()** procedure, and then runs the **low_inventory()** procedure:

```
CREATE PROCEDURE xqt_low_inventory()
        DEFINE curstate integer;
        EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
        EXECUTE PROCEDURE ifx_set_erstate(1);
        EXECUTE PROCEDURE low_inventory();
        EXECUTE PROCEDURE ifx_set_erstate(curstate);
END PROCEDURE;
```

The following events occur in this procedure:

1. The **xqt_low_inventory()** procedure defines a data variable called curstate to hold the Enterprise Replication state information.
2. The **ifx_get_erstate()** function obtains the Enterprise Replication state and stores it in the curstate variable. The **ifx_set_state()** procedure enables replication.
3. The **low_inventory()** procedure is run.
4. The replication state is reset back to its original value.

When a transaction runs the **xqt_low_inventory()** procedure, the execution of the procedure is replicated to all replication servers and the result of the **low_inventory()** procedure is then replicated like any normal updating activity.

Related reference:
ifx_set_erstate() procedure
ifx_get_erstate() function

Monitor and troubleshooting Enterprise Replication

You can monitor and diagnose problems with the Enterprise Replication system by using several different methods, depending on your needs.

You can monitor the status of Enterprise Replication servers in the following ways:

- Use the **cdr view** command. Specify one or more subcommands, depending on what information you want to monitor.
- Use SQL queries on the system monitoring tables.
- Run **onstat** commands to view local server information.
- Run the **cdr check queue --qname=cntrlq** command to determine whether the operation is finished propagating to all servers.
- Run the **DBINFO('cdrsession')** function to determine if a session thread is performing an Enterprise Replication apply or sync operation.

Set the ALARMPROGRAM script to capture event alarms for the following situations:

- Enterprise Replication errors
- The Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files
- Dropped connections between replication servers
- Replication state changes caused by Enterprise Replication commands, if state change event alarms are enabled

- Solve Replication Processing Problems
  Diagnose, monitor, and solve possible problems that can occur while Enterprise Replication is running.
- Failed Transaction (ATS and RIS) Files
  Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files can be generated when replicated transactions fail.
- Preventing Memory Queues from Overflowing
  In a well-tuned Enterprise Replication system, the send queue and receive queue do not regularly overflow from memory to disk. However, if the queues in memory fill, the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of transaction records, replicate information, and row data. Spooled transaction records and replicate information are stored in the transaction tables and the replicate information tables in a single dbspace. Spooled row data is stored in one or more sbspaces.
- Common configuration problems
  If you experience problems setting up Enterprise Replication, check the configuration of your environment and database.
- Troubleshooting Tips for Alter Operations
  Alter operations on replicated tables might result in errors.
- Enterprise Replication Event Alarms
  Certain Enterprise Replication errors and other actions generate event alarms. You can use event alarms specific to Enterprise Replication to automate many administrative tasks.

Related concepts:
Preparing the Replication Environment
Using High-Availability Clusters with Enterprise Replication
Grid setup and management
Shard cluster setup
Managing Replication Servers and Replicates
Related tasks:
Defining Replication Servers, Replicates, Participants, and Replicate Sets
Setting Up Failed Transaction Logging
Related reference:
cdr view
SMI Tables for Enterprise Replication Reference
onstat -g commands for Enterprise Replication
Enterprise Replication Event Alarms
cdr check queue
cdr define server
cdr modify server
Related information:
DBINFO Function

Solve Replication Processing Problems

Diagnose, monitor, and solve possible problems that can occur while Enterprise Replication is running.

You should understand the typical behavior of your Enterprise Replication system. There are many factors that contribute to the performance and other behaviors, including: hardware configuration, network load and speed, type of replication, and number of replicated transactions.

Use the **cdr view** command or the SMI tables to understand the typical behavior of your system, establish benchmarks, and track trends. Deviations from typical behavior do not necessarily indicate a problem. For example, transactions might take longer to replicate during peak usage times or during end-of-month processing.

The following table describes some replication processing problems that might occur.

Table 1. Potential Replication Problems and Solutions

Problem	How to diagnose	How to solve
Enterprise Replication is not running	• Run the **cdr view state** command • Query the **syscdr_state** SMI table • Examine event alarms captured by the alarm program	Start replication with the **cdr start** command.

Problem	How to diagnose	How to solve
One or more Enterprise Replication servers are not running or connected to the network	• Run the **cdr view servers** command • Run the **cdr view nif** command • Query the **syscdr_nif** SMI table • Examine event alarms captured by the alarm program	Start the database server or fix the connection problem.
Replicated transactions failed	Determine if there are ATS or RIS files: • Look at the ATS and RIS directories on the local server for the existence of ATS or RIS files • Run the **cdr view atsdir risdir** command to see the number of ATS and RIS files for each server • Query the **syscdr_atsdir** or **syscdr_risdir** SMI table for a specific server • Examine event alarms captured by the alarm program	Run one of the following commands: • **cdr repair** • **cdr check replicate --repair** • **cdr check replicateset --repair** See cdr repair, cdr check replicate, and cdr check replicateset.
Transactions are spooling to disk	Determine how much spool memory is being used: • Run the **cdr view profile** command to see the status of all queues on all servers • Run the **cdr view sendq** command to see the status of the send queue on all servers • Run the **cdr view rcv** command to see the status of the receive queue on all servers	See Increasing the Sizes or Numbers of Storage Spaces.
Potential log wrap situation	Determine how many log pages must be used before Enterprise Replication reacts a potential log wrap situation: • Run the **cdr view ddr** command to see the number of unused log pages for all servers • Query the **syscdr_ddr** SMI table to see the number of unused log pages for a specific server	See Handle potential log wrapping.

If you do need to call Software Support, find the version of the database server that is running Enterprise Replication with the **cdr -V** command.

Failed Transaction (ATS and RIS) Files

Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files can be generated when replicated transactions fail.

You can use the ATS and RIS files to identify problems or as input to the **cdr repair** command or custom utilities that extract or reapply the aborted rows.

When ATS or RIS file generation is enabled for a replicate, all failed replication transactions are recorded in ATS or RIS files. Each ATS file contains all the information pertinent to a single failed transaction, while each RIS file contains information about a single failed row. If a replicated transaction fails for any reason (constraint violation, duplication, and so forth), all the buffers in the replication message that compose the transaction are written to a local file.

ATS file generation occurs if the entire transaction is aborted. Transactions defined with row scope that have aborted rows but are successfully committed on the target tables are not logged. All rows that fail conflict resolution for a transaction that has row scope defined are also written to the RIS file, if RIS is enabled.

RIS files can contain the following types of information:

- Individual aborted row errors
- Replication exceptions (such as when a row is converted by Enterprise Replication from insert to update, or from update to insert, and so forth)
- Special SPL routine return codes, as defined by the application (if an SPL routine is called to resolve a conflict)

In some cases, such as with long transactions, the database server itself aborts transactions. In these cases, Enterprise Replication does *not* generate an ATS or RIS file.

ATS and RIS files can be generated under the following circumstances:

- ATS or RIS generation is enabled for a replicate, the replicate uses a conflict resolution rule other than ignore or always-apply, and a conflict is detected on a target server.
- Under some error conditions, ATS or RIS files can be generated on a source server, regardless if ATS or RIS generation is enabled or the conflict resolution rule.

When an ATS or RIS file is generated, an event alarm with a class ID for 48 is also generated. You can use event alarms to send notifications to a database administrator.

- Enabling ATS and RIS File Generation
  You can enable the generation of ATS and RIS files when you define a replicate.
- ATS and RIS File Names
  Each ATS and RIS file has a unique name based on the conditions under which it was generated.
- ATS and RIS File Formats
  You can choose to generate ATS and RIS files in text format, XML format, or both formats.
- Disabling ATS and RIS File Generation
  You can prevent the generation of ATS or RIS files, or both.
- Suppressing Data Sync Errors and Warnings
  You prevent certain data sync errors and warnings from appearing in ATS and RIS files by using the CDR_SUPPRESS_ATSRISWARN configuration parameter.

Related concepts:
Conflict Resolution Scope

Related tasks:
Creating ATS and RIS directories
Repairing Failed Transactions with ATS and RIS Files
Related reference:
cdr view
CDR_DISABLE_SPOOL Environment Variable
cdr define replicate

Enabling ATS and RIS File Generation

You can enable the generation of ATS and RIS files when you define a replicate.

Failed transactions are not automatically recorded in ATS and RIS files. You can choose to generate either ATS or RIS files, or both.
You should create a separate directory to store ATS and RIS files. If you do not create a separate directory and specify it when you define the replication server, Enterprise Replication stores the ATS and RIS files in the /tmp directory on UNIX and the %INFORMIXDIR%\tmp directory on Windows.

To collect ATS and RIS information

1. Create a directory for Enterprise Replication to store ATS and RIS files. You can create two directories if you want to generate both types of file and store them in separate directories.
   - If you are using primary-target replication, create the directory on the target system.
   - If you are using update-anywhere replication and have a conflict resolution rule other than ignore or always-apply enabled, create the directory on all participating replication systems.
2. When you define or modify a replication server, specify the location of the ATS and RIS directory by using the **--ats** and **--ris** options of the **cdr define server** command or the **cdr modify server** command.
3. When you define or modify a replicate, specify that ATS and RIS file generation is enabled by using the **--ats** and **--ris** options of the **cdr define replicate** command or the **cdr modify replicate** command.

Related tasks:
Creating ATS and RIS directories
Related reference:
cdr define server
cdr define replicate
cdr modify server
cdr modify replicate

ATS and RIS File Names

Each ATS and RIS file has a unique name based on the conditions under which it was generated.

The following table provides the naming convention for ATS and RIS files:

type.target.source.threadID.timestamp.sequence.extension

Table 1. ATS and RIS file naming conventions

Name	Description
type	The format of the file: `ats` or `ris`.
target	The name of the database server receiving this replicate transaction.
source	The name of the database server that originated the transaction.
threadID	The identifier of the thread that processed this transaction.
timestamp	The value of the internal time stamp at the time that this ATS or RIS file was generated.
sequence	A unique integer, incremented each time an ATS or RIS file is generated.
extension	The file type. No extension indicates a text file; `xml` indicates an XML file.

The naming convention ensures that all ATS and RIS file names that are generated are unique. However, when an ATS or RIS file is opened for writing, any previous file contents are overwritten. (Enterprise Replication does not append to a spool file; if a name collision does occur with an existing file, the original contents of the file are lost.)

The default delimiter for the *timestamp* portion of text file names is a colon (`:`) on UNIX and a period (`.`) on Windows. You can define the delimiter between the hour, minute, and second values with the CDR_ATSRISNAME_DELIM environment variable. XML files always use a period (`.`) delimiter between the hour, minute, and second values.

The following is an example of a name of an ATS file in text format on UNIX for a transaction sent by server **g_amsterdam** to server **g_beijing**:

```
ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6
```

The following is an example of the same ATS file name in XML format:

```
ats.g_beijing.g_amsterdam.D_2.000529_23.27.16.6.xml
```

The following is an example of a similar RIS file name in XML format:

```
ris.g_beijing.g_amsterdam.D_2.000529_23.27.16.5.xml
```

Related reference:
CDR_ATSRISNAME_DELIM Environment Variable

ATS and RIS File Formats

You can choose to generate ATS and RIS files in text format, XML format, or both formats.

The format of ATS and RIS files is part of the server definition that you create with the **cdr define server** command:

Text (Default)
: ATS and RIS files are generated as text files that Enterprise Replication can process during a repair operation. Text format is useful if you intend to use the **cdr repair** command to repair inconsistencies.

XML
: ATS and RIS files are generated as XML files that you can use if you write your own custom repair scripts. You cannot use ATS or RIS files in XML format with the **cdr repair** command.

Both
: ATS and RIS files are generated in both text and XML format so that you can choose how to process failed transactions.

Enterprise Replication raises event alarms when ATS and RIS files are generated regardless of format.

- XML File Format
  The information in ATS and RIS files that are in XML format is organized in specific XML tags.
- ATS and RIS Text File Contents
  The information about failed replicated transactions that are shown in ATS and RIS text files is listed in rows that are prefaced by information labels.

XML File Format

The information in ATS and RIS files that are in XML format is organized in specific XML tags.

The XML format uses an XML schema that is stored in the **INFORMIXDIR/etc** directory.

Data in XML files uses the UTF-8 encoding format.

Columns that appear empty could contain a null value or an empty string. The XML format differentiates between null data and empty strings by setting the isNull="true" attribute of the COLUMN tag for null data.

Data Types That are Not Shown

The values of the following data types are not shown in XML files:

- Smart large objects
- Simple large objects
- User-defined data types

For these data types, the following attributes are set for the COLUMN tag:

- isLOBorUDT="true"
- dataExists="false"

Special Symbols

The following symbols are replaced if they exist in row data:

- < is replaced by `<`
- > is replaced by `>`
- & is replaced by `&`
- " is replaced by `"`
- ' is replaced by `'`

Example

The following example shows an ATS file displaying a transaction with two failed insert operations. The third column in each row contains a data type that is not shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<ERFILE version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/informix/etc/idser.xsd">
  <ATS version="1">
    <TRANSACTION RISFile="/tmp/ris.g_cdr_ol_3.g_cdr_ol_2.D_5.080411_14.08.57.3.xml"
```

```
                generateRISFile="true" processedRows="2">
      <SOURCE id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.erep.doc_ids_erp_515_20" name="g_cdr_ol_2"
commitTime="2008-04-11T14:08:57"/>
      <TARGET id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.erep.doc_ids_erp_515_30" name="g_cdr_ol_3"
receiveTime="2008-04-11T14:08:57"/>
      <MESSAGE>All rows in a transaction defined with row scope were rejected</MESSAGE>
    </TRANSACTION>
    <ATSROWS>
      <ATSROW num="1" replicateID="655362" database="bank" owner="testadm" table="customer"
            operation="Insert">
        <REPLICATED>
          <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                         cdrTimeString="2008-04-11T14:08:57"/>
          <DATA>
            <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">261</COLUMN>
            <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">cdr_ol_2</COLUMN>
            <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                    isNull="false"></COLUMN>
          </DATA>
        </REPLICATED>
      </ATSROW>
      <ATSROW num="2" replicateID="655362" database="bank" owner="testadm" table="customer"
            operation="Insert">
        <REPLICATED>
          <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                         cdrTimeString="2008-04-11T14:08:57"/>
          <DATA>
            <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">262</COLUMN>
            <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">cdr_ol_2</COLUMN>
            <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                    isNull="false"></COLUMN>
          </DATA>
        </REPLICATED>
      </ATSROW>
    </ATSROWS>
  </ATS>
</ERFILE>
```

The following example shows the corresponding RIS file for the failed transaction shown in the ATS example.

```
<?xml version="1.0" encoding="UTF-8"?>
<ERFILE version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/informix/etc/idser.xsd">
  <RIS version="1">
    <SOURCE id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.erep.doc_ids_erp_515_20" name="g_cdr_ol_2"
commitTime="2008-04-11T14:08:57"/>
    <TARGET id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.erep.doc_ids_erp_515_30" name="g_cdr_ol_3"
receiveTime="2008-04-11T14:08:57"/>
    <RISROWS>
      <RISROW num="1" replicateID="655362" database="bank" owner="testadm" table="customer"
            operation="Insert">
        <CDRERROR num="0"/>
        <SQLERROR num="-668"/>
        <ISAMERROR num="-1"/>
        <SPLCODE num="63"/>
        <LOCAL>
          <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1206852121"
                         cdrTimeString="2008-04-11T12:08:57"/>
          <DATA>
            <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">261</COLUMN>
            <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">cdr_ol_2</COLUMN>
            <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                    isNull="false"></COLUMN>
          </DATA>
        </LOCAL>
        <REPLICATED>
          <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                         cdrTimeString="2008-04-11T14:08:57"/>
          <DATA>
            <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">261</COLUMN>
            <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">cdr_ol_2</COLUMN>
            <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                    isNull="false"></COLUMN>
          </DATA>
        </REPLICATED>
      </RISROW>
    </RISROWS>
    <TXNABORTED ATSFile="/tmp/ats.g_cdr_ol_3.g_cdr_ol_2.D_5.080411_14.08.57.4.xml"
                generateATSFile="true"/>
  </RIS>
</ERFILE>
```

- XML Tags
  XML tags are used in ATS and RIS files that are generated in XML format.

XML Tags

XML tags are used in ATS and RIS files that are generated in XML format.

Table 1. XML tags in ATS and RIS files

Tag name	Description	Attributes	Parent tag	Child tags
ERFILE	Top level tag for ATS and RIS files	`version`: XML file format version number.	None	ATS RIS
ATS	Parent tag for ATS files	`version`: ATS file format version number.	ERFILE	TRANSACTION ATSROWS
RIS	Parent tag for RIS files	• `version`: RIS file format version number. • `fromSource`: Set to `true` if the RIS file is generated at the source server.	ERFILE	SOURCE TARGET RISROWS TXNABORTED TXNCOMMITTED
TRANSACTION	Contains the name of the RIS file (if it exists) and the number of rows processed before the transaction was aborted.	• `RISFile`: The name of the RIS file, if it was created. • `generateRISFile`: Set to `true` if an RIS file exists for this aborted transaction. • `processedRows`: Number of rows processed before the transaction was aborted.	ATS	SOURCE TARGET MESSAGE CDRERROR SQLERROR ISAMERROR SPLCODE
ATSROWS	Contains the replicated aborted rows	None	ATS	ATSROW
SOURCE	Contains source server information	• `id`: Server ID. • `name`: Server group name. • `commitTime`: Transaction commit time.	TRANSACTION RIS	None
TARGET	Contains target server information	• `id`: Server ID. • `name`: Server group name. • `receiveTime`: Transaction receive time.	TRANSACTION RIS	None
SQLERROR	Contains the SQL error code	`num`: Error number.	TRANSACTION RISROW	None
ISAMERROR	Contains the ISAM error code	`num`: Error number.	TRANSACTION RISROW	None
CDRERROR	Contains the data sync error code	• `num`: Error number. • `description`: Error description.	TRANSACTION RISROW	None
MESSAGE	Contains the notification message	None	TRANSACTION RISROW	None
SPLCODE	Contains the SPL code number if a stored procedure conflict rule is being used	`num`: SPL code number.	TRANSACTION RISROW	None
RISROWS	Contains the local and replicated aborted rows	None	RIS	RISROW

Tag name	Description	Attributes	Parent tag	Child tags
RISROW	Contains information about local or replicated row data for one aborted row	• num: Row sequence number. • replicateID: Replicate ID. • database: Database name. • owner: Table owner name. • table: Table name. • operation: DML operation type.	RISROWS	MESSAGE CDRERROR SQLERROR ISAMERROR SPLCODE MESSAGE LOCAL REPLICATED
LOCAL	Contains the local row data for an aborted row	None	RISROW	SHADOWCOLUMNS DATA
REPLICATED	Contains replicated row data for an aborted row	None	ATSROW RISROW	SHADOWCOLUMNS DATA
ATSROW	Contains information for one replicated aborted row	• num: Row sequence number. • replicateID: Replicate ID. • database: Database name. • owner: Table owner name. • table: Table name. • operation: DML operation type.	ATSROWS	REPLICATED
SHADOWCOLUMNS	Optional shadow column values for local and replicated rows	• serverID: Server ID. • serverName: Server group name. • cdrTimeInt: The **cdrtime** column value in integer format (GMT time). • cdrTimeString: Time in string format. For example: 2008-11-08T20:16:25.	LOCAL REPLICATED	None
DATA	Contains aborted row data	dataExists: Identifies whether data exists for this row or not.	ATSROW RISROW	COLUMN
COLUMN	Contains column data for an aborted row	• name: The column name. • dataExists: Identifies whether data is displayed for this column or not. • isLOBorUDT: Set to true if the column is of type UDT, smart large object or simple large object. If set to true, data for the column is skipped and the dataExists value is set to false. • isHex: Set to true if column data is displayed in hex format because Enterprise Replication does not have enough information to interpret the row data. • isNull: Set to true if the column value is NULL. Set to false if the column has a valid value or an empty string.	DATA	None
TXNABORTED	Indicates that the replicated transaction was aborted	• ATSFile: The name of the ATS file if the transaction was aborted and an ATS file was created for this aborted row. • generateATSFile: Set to true if an ATS file was created. • TxnErr: Error description for the aborted transaction.	RIS	None

Tag name	Description	Attributes	Parent tag	Child tags
TXNCOMMITTED	Indicates that the replicated transaction was committed	`totalRows`: Total number of rows processed.	RIS	None

ATS and RIS Text File Contents

The information about failed replicated transactions that are shown in ATS and RIS text files is listed in rows that are prefaced by information labels.

The first three characters in each line of the ATS and RIS file describe the type of information for the line, as the following table defines. The first four labels apply to both ATS and RIS files. The last three labels apply to only RIS files.

Table 1. Information labels

Label	Name	Description
TXH	Transaction heading	This line contains information from the transaction header, including the sending server ID and the commit time, the receiving server ID and the received time, and any Enterprise Replication, SQL, or ISAM error information for the transaction.
RRH	Replicated row heading	This line contains header information from the replicated rows, including the row number within the transaction, the group ID, the replicate ID (same as replicate group ID if replicate is not part of any replicate group), the database, owner, table name, and the database operation.
RRS	Replicated row shadow columns	This line contains shadow column information from replicated rows, including the source server ID and the time when the row was updated on the source server. This line is printed only if the replicate is defined with a conflict-resolution rule.
RRD	Replicated row data	This line contains the list of replicated columns in the same order as in the SELECT statement in the **cdr define replicate** command. Each column is separated by a pipe character (|) and displayed in ASCII format. When the spooling program encounters severe errors (for example: cannot retrieve the replicate ID for the replicated row; unable to determine the replicated column type, size, or length), it displays this row data in hexadecimal format. The spooling program also displays the row data in hexadecimal format if a row includes replicated UDT columns.
LRH	Local-row header	RIS only. Indicates if the local row is found in the delete table and not in the target table
LRS	Local-row shadow columns	RIS only. Contains the server ID and the time when the row was updated on the target server This line is printed only if the replicate is defined with a conflict resolution rule.
LRD	Local-row data	RIS only. Contains the list of replicated columns extracted from the local row and displayed in the same order as the replicated row data. Similar to the replicated row data, each column is separated by a '|' and written in ASCII format. When the spooling program encounters severe errors (for example: cannot retrieve the replicate ID for the replicated row; unable to determine the replicated column type, size, or length) or the table includes UDT columns (whether defined for replication or not), it displays the replicated row data in hexadecimal format. In this case, the local row data is not spooled.

Changed Column Information

If you define a replicate to only replicate columns that changed, the RRD entry in the ATS and RIS file shows a `?` for the value of any columns that are not available. For example:

```
RRD 427|amsterdam|?|?|?|?|?|?|?|?|?|?
```

For more information, see Replicate only changed columns.

BLOB and CLOB Information

If a replicate includes one or more BLOB or CLOB columns, the RRD entry in the ATS and RIS file displays the smart large object metadata (the in-row descriptor of the data), not the smart large object itself, in hexadecimal format.

BYTE and TEXT Information

When the information recorded in the ATS or RIS file includes BYTE or TEXT data, the replicated row data (RRD) information is reported, as the following examples show.

Example 1

```
<1200, TEXT, PB 877(necromsv) 840338515(00/08/17 20:21:55)>
```

In this example:

- `1200` is the size of the data.
- `TEXT` is the data type (it is either BYTE or TEXT).
- `PB` is the storage type (`PB` when the BYTE or TEXT is stored in the tblspace, `BB` for blobspace storage).
- The next two fields are the server identifier and the time stamp for the column if the conflict-resolution rule is defined for this replicate and the column is stored in a tblspace.

```
<500 (NoChange), TEXT, PB 877(necromsv) 840338478(00/08/17 20:21:18)>
```

Example 2

In this example, `500 (NoChange)` indicates that the TEXT data has a size of 500, but the data is not changed on the source server. Therefore, the data is not sent from the source server.

Example 3

```
<(Keep local blob),75400, BYTE, PB 877(necromsv) 840338515(00/08/17 20:21:55)>")
```

In this example, (`Keep local blob`) indicates that the replicated data for this column is not applied on the target table, but instead the local BYTE data was kept. This usually happens when time stamp conflict resolution is defined and the local column has a time stamp greater than the replicated column.

UDT Information

If a replicate includes one or more UDT columns, the RRD entry in the ATS and RIS files displays the row data in delimited format as usual, except the string `<skipped>` is put in place of UDT column values. For example, the following row shows information about a table with columns of type INTEGER, UDT, CHAR(10), and UDT:

```
RRD 334|<skipped>|amsterdam|<skipped>
```

TimeSeries information

If a replicate includes a **TimeSeries** column, an RTS row displays the time series instance ID and the timestamp of the element. If the failed replicated transaction includes a TimeSeries routine that affects a range of elements, both the starting and ending timestamps are shown. The following example shows three failed replication transactions that include a **TimeSeries** column:

```
TXH Source ID:100 / Name:g_delhi / CommitTime:12-01-27 12:27:39
TXH Target ID:200 / Name:g_bombay / ReceiveTime:12-01-27 12:27:39
----------
RRH Row:1 / Replicate Id: 6553638 / Table: test@tstr1.tpk / DbOp:TSInsert
RRH CDR:2 (ERROR DESCRIPTION) / SQL:0 / ISAM:0
RTS  Instance id=100, Timestamp=12-01-27 12:27:39

RRH Row:2 / Replicate Id: 6553638 / Table: test@tstr1.tpk / DbOp:TSDelete
RRH CDR:2 (ERROR DESCRIPTION) / SQL:0 / ISAM:0
RTS  Instance id=100, Timestamp=12-01-27 12:27:39

RRH Row:2 / Replicate Id: 6553638 / Table: test@tstr1.tpk / DbOp:TSDelRange
RRH CDR:2 (ERROR DESCRIPTION) / SQL:0 / ISAM:0
RTS  Instance id=100, Begin Timestamp=12-01-27 12:27:39, End Timestamp=12-01-27 12:27:39
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```


Disabling ATS and RIS File Generation

You can prevent the generation of ATS or RIS files, or both.

To prevent the generation of both ATS and RIS files, set the CDR_DISABLE_SPOOL environment variable to `1`.
To prevent the generation of either ATS or RIS files, set the ATS or RIS directory to **/dev/null** (UNIX) or **NUL** (Windows) with the **cdr define server** or **cdr modify server** commands.

Related reference:
cdr modify server
cdr define server
CDR_DISABLE_SPOOL Environment Variable

Suppressing Data Sync Errors and Warnings

You prevent certain data sync errors and warnings from appearing in ATS and RIS files by using the CDR_SUPPRESS_ATSRISWARN configuration parameter.

For more information on the CDR_SUPPRESS_ATSRISWARN configuration parameter, see CDR_SUPPRESS_ATSRISWARN Configuration Parameter.

For a list of error and warning messages that you can suppress, see Data sync warning and error messages.

Preventing Memory Queues from Overflowing

In a well-tuned Enterprise Replication system, the send queue and receive queue do not regularly overflow from memory to disk. However, if the queues in memory fill, the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of transaction records, replicate information, and row data. Spooled transaction records and replicate information are stored in the transaction tables and the replicate information tables in a single dbspace. Spooled row data is stored in one or more sbspaces.

The following situations can cause Enterprise Replication to spool to disk:

- Receiving server is down or suspended.
- Network connection is down.
  If the receiving server or network connection is down or suspended, Enterprise Replication might spool transaction buffers to disk.

  To check for a down server or network connection, run **cdr list server** on a root server. This command shows all servers and their connection status and state.

- Replicate is suspended.
  If a replicate is suspended, Enterprise Replication might spool transaction buffers to disk.

  To check for a suspended replicate, run **cdr list replicate**. This command shows all replicates and their state.

- Enterprise Replication is replicating large transactions.
  Enterprise Replication is optimized to handle small transactions efficiently. Very large transactions or batch jobs force Enterprise Replication into an exceptional processing path that results in spooling. For best results, avoid replicating these types of transactions.

- Logical log files are too small or too few.
  If the logical log files are too small or the number of logical log files is too few, Enterprise Replication is more likely to spool transaction buffers to disk.

- Server is overloaded.
  If a server is low on resources, Enterprise Replication might not be able to hold all transactions that are replicating from a source server in memory during processing, and the transactions spool to disk.

  If transactions spool to disk, check the system resources; in particular, check disk speed, RAM, and CPU resources.

- Handle potential log wrapping
  The potential for log wrap occurs when Enterprise Replication log processing lags behind the current log and the Enterprise Replication replay position is in danger of being overrun.
- Monitoring Disk Usage for Send and Receive Queue Spool
  Periodically monitor disk usage for the dbspace.
- Increasing the Sizes or Numbers of Storage Spaces
- Recovering when Storage Spaces Fill

Related concepts:
Send queues and receive queues
Setting Up Send and Receive Queue Spool Areas
Transaction processing impact
Logical Log Configuration Guidelines
Related reference:
cdr list server
cdr list replicate

Handle potential log wrapping

The potential for log wrap occurs when Enterprise Replication log processing lags behind the current log and the Enterprise Replication replay position is in danger of being overrun.

There are two log positions you should be aware of: the snoopy log position, which is the log position that keeps track of transactions being captured for replication, and the log replay position, which is the log position that keeps track of which transactions have been applied.

A potential log wrap situation is usually caused by the logical logs being misconfigured for the current transaction activity or by the Enterprise Replication system having to spool more than usual. More-than-usual spooling could be caused by one of the following situations:

- A one-time job might be larger than normal and thus require more log space.
- One of the target servers is currently unavailable and more spooling of replicated transactions is required.
- The spool file or paging space could be full and needs to be expanded.

You can configure how Enterprise Replication responds to a potential log wrap situation by specifying one or more of the following solutions, in order of priority, with the CDR_LOG_LAG_ACTION configuration parameter:

- Block user transactions until Enterprise Replication log processing advances far enough that the danger of log wrapping is diminished. Blocking user transactions prevents the current log position from advancing. This solution increases user response time. When user transactions are blocked, event alarm 30 unique ID 30002 is raised and the following message appears in the online log:

  ```
  DDR Log Snooping - DDRBLOCK phase started, userthreads blocked
  ```

- Compress the logical logs and save them to a log staging directory. Log files in the staging directory are deleted after they are no longer required by Enterprise Replication. You must specify the location and maximum size of the log staging directory. This solution uses very little additional disk space to temporarily save log files until the danger of log wrapping is over. The staged log files are deleted after advancing the log replay position.
  If log staging is configured, Enterprise Replication monitors the log lag state and stages log files even when Enterprise Replication is inactive.

- Dynamically add logical logs. This solution requires enough free space to be available in the logical log dbspace to add dynamic logs. You can specify how many dynamic logical logs to add. You must manually drop the dynamic log files when the danger for log wrapping is over.
- Ignore the potential for log wrap. This solution shuts down Enterprise Replication when an overrun of the snoopy log replay position is detected. Enterprise Replication continues to function if the log replay position is overrun. If the snoopy replay position is overrun, Enterprise Replication is stopped, event alarm 47 is raised, and the following message appears in the message log file:

  ```
  WARNING:  The replay position was overrun, data may not be replicated.
  ```

  If the replay position is overrun, restart Enterprise Replication with the **cdr cleanstart** command to reset replay position to current log position and synchronize the data.

- Stop Enterprise Replication on the affected server as soon as it is detected that the log replay position is running behind. When you are ready to restart Enterprise Replication it is necessary to run the **cdr cleanstart** command only if the log replay position was overrun.

For example, you can specify that during a potential log wrap situation, Enterprise Replication stages compressed logical logs. If the log staging directory reaches its maximum size, then logical logs are added. If the maximum number of logical logs are added, then Enterprise Replication blocks user transactions. Not all options can be combined together in every possible priority order. For example, specifying to stop Enterprise Replication, to ignore the potential for log wrap, or to block user actions must always be either the only option or the last option in the list.

Related concepts:
Recovering when Storage Spaces Fill
Logical Log Configuration Guidelines
Related tasks:
Monitoring Disk Usage for Send and Receive Queue Spool
Increasing the Sizes or Numbers of Storage Spaces
Related reference:
CDR_LOG_LAG_ACTION configuration parameter
CDR_LOG_STAGING_MAXSIZE Configuration Parameter
CDR_MAX_DYNAMIC_LOGS Configuration Parameter

Monitoring Disk Usage for Send and Receive Queue Spool

Periodically monitor disk usage for the dbspace.

The sbspace that Enterprise Replication uses to spool the queues to disk is specified by the CDR_QDATA_SBSPACE configuration parameter.

To check disk usage for the spooling sbspace, run one or more of the following commands:

- **onstat -g rqm SBSPACES**
- **onstat -d**
  Tip: When you use the **onstat -d** command to monitor disk usage, the S flag in the **Flags** column indicates an sbspace. For each sbspace chunk, the first row displays information about the whole sbspace and user-data area. The second row displays information about the metadata area.
- The **oncheck** command with the **-cs**,**-cS**, **-ce**, **-pe**, **-ps**, and **-pS** options

Related concepts:
Handle potential log wrapping
Recovering when Storage Spaces Fill
Related tasks:
Increasing the Sizes or Numbers of Storage Spaces
Related reference:
onstat -g rqm: Prints statistics for RQM queues
CDR_QDATA_SBSPACE Configuration Parameter
Related information:
Manage sbspaces
onstat -d command: Print chunk information
The oncheck Utility

Increasing the Sizes or Numbers of Storage Spaces

If you notice that the Enterprise Replication dbspace or sbspace is running out of disk space, you can increase the size of the space by adding chunks to the space. You can also add additional sbspaces for Enterprise Replication.

To add a chunk to a dbspace, use **onspaces -a**. For example, to add a 110 kilobyte chunk with an offset of 0 to the **er_dbspace** dbspace, enter:

```
onspaces -a er_dbspace -p /dev/raw_dev2 -o 0 -s 110
```

To add a chunk to an sbspace, use the same **onspaces** command above, however you can specify more information about the chunk that you are adding. After you add a chunk to the sbspace, you must perform a level-0 backup of the root dbspace and the sbspace.

See the sections on adding chunks to dbspaces and sbspaces in the *IBM® Informix® Administrator's Guide* and the *IBM Informix Administrator's Reference* for more information.

To increase the number of sbspaces that can be used for Enterprise Replication, create new sbspaces with the **onspaces -c -S** command and then add their names to the CDR_QDATA_SBSPACE configuration parameter with the **cdr add onconfig** command. For more information, see cdr add onconfig.

Related concepts:
Handle potential log wrapping
Recovering when Storage Spaces Fill
Related tasks:
Monitoring Disk Usage for Send and Receive Queue Spool

Recovering when Storage Spaces Fill

When the Enterprise Replication dbspace runs out of disk space, Enterprise Replication raises an alarm and writes a message to the log. When the sbspace runs out of disk space, Enterprise Replication hangs. In either case, you must resolve the problem that is causing Enterprise Replication to spool (Preventing Memory Queues from Overflowing) or you must allocate additional disk space (Increasing the Sizes or Numbers of Storage Spaces) before you can continue replication.

Related concepts:
Handle potential log wrapping
Related tasks:
Monitoring Disk Usage for Send and Receive Queue Spool
Increasing the Sizes or Numbers of Storage Spaces

Common configuration problems

If you experience problems setting up Enterprise Replication, check the configuration of your environment and database.

To solve configuration problems:

- Make sure that you created an sbspace for the row data and set the CDR_QDATA_SBSPACE in the onconfig file.
  For more information, see Setting Up Send and Receive Queue Spool Areas and CDR_QDATA_SBSPACE Configuration Parameter.
- Verify that the trusted environment is set up correctly.
  For more information, see Configuring secure ports for connections between replication servers.
- Verify that your sqlhosts file is set up properly on each server that participates in replication. You must set up database server groups in the sqlhosts file.
  For more information, see Creating sqlhost group entries for replication servers.
- Verify the format of the sqlhosts file.
  The network connection (not the shared memory connection) entry must be immediately after the database server group definition. If the network connection entry is not immediately after the database server group definition, you might see the following error when you run **cdr define server**:

  ```
  command failed -- unable to connect to server specified (5)
  ```

  You might also see a message like the following in the message log for the target server:

  ```
  Reason: ASF connect error (-25592)
  ```
- Make sure that the unique identifier for each database server (**i=** in the **options** field of the sqlhosts information) is consistent across all nodes in the domain.
  For more information, see Creating sqlhost group entries for replication servers.
- Verify that the operating system times of the database servers that participate in the replicate are synchronized.
  For more information, see Time synchronization.
- Make sure that the database server has adequate logical log disk space. If the database server does not have enough logical log space at initialization, you see the following error:

  ```
  command failed -- fatal server error (100)
  ```
- Check the files in the $INFORMIXDIR directory to see if a problem occurred when the database server built the SMI tables.
- Make sure that the databases on all database server instances that are involved in replication are set to logging (unbuffered logging is recommended).
  For more information, see Unbuffered Logging.
- For replicates that use any conflict-resolution rule except ignore and always-apply, make sure that you define shadow columns (CRCOLS) for each table that is involved in replication.
  For more information, see Preparing Tables for Conflict Resolution.
- If you defined a participant using SELECT * from *table_name* statement, make sure that the tables are identical on all database servers that are defined for the replicate.
  For more information, see Participant definitions and Participant and participant modifier.
- Verify that each replicated column in a table on the source database server has the same data type as the corresponding column on the target server. Enterprise Replication does not support replicating a column with one data type to a column on another database server with a different data type.

  The exception to this rule is cross-replication between simple large objects and smart large objects.

  For more information, see Replication and data types.
- Verify that all tables defined in a replicate have a replication key.
  For more information, see Unique key for replication.
- If high-availability clusters are also in use in the domain, then all row data sbspaces must be created with logging by using the **-Df "LOGGING=ON"** option of the **onspaces** command.
  For more information, see Row Data sbspaces and the *IBM® Informix® Administrator's Guide*.

Troubleshooting Tips for Alter Operations

Alter operations on replicated tables might result in errors.

The following problems illustrate common issues with performing alter operations on replicated tables:

- **Problem:** You receive an error that the replicate is not defined after running the following command:

  ```
  cdr alter -o test:tab
  Error:Replicate(s) not defined on table test:.tab
  ```

  The owner name is missing from the table name, **test:tab**.

  Solution: Include the table owner name, for example:

  ```
  cdr alter -o test:user1.tab
  ```

- **Problem:** You receive an error that the replicated table is in alter mode after running the following command:

  ```
  > insert into tab values(1,1);

  19992: Cannot perform insert/delete/update operations on a replicated table
  while the table is in alter mode
  Error in line 1 Near character position 27
  >
  ```

  The table (**tab**) is in alter mode. DML operations cannot be performed while the table is in alter mode.

  Solution: Wait for the table to be altered and then issue the DML operation. If no alter statement is in progress against the table, then unset alter mode on the table using the **cdr alter --off** command. For example:

  ```
  cdr alter --off test:user1.tab
  ```

  You can check the alter mode status using the **oncheck -pt** command. For example:

  ```
  $ oncheck -pt db1:user1.t1

  TBLspace Report for db1:user1.t1

      Physical Address             1:63392
      Creation date                02/01/2011 16:02:00
      TBLspace Flags               400809     Page Locking
                                              TBLspace flagged for replication
                                              TBLspace flagged for CDR alter mode
                                              TBLspace use 4 bit bit-maps
      Maximum row size              4
  ...
  ```

- **Problem:** How can you tell if a replicate is a mastered replicate?
  Solution: You can check the alter mode status using the **oncheck -pt** command. For example:

  ```
  oncheck -pt test:nagaraju.tab
  ```

- **Problem:** How can you tell if a replicate is a mastered replicate?
  Solution: When you execute the **cdr list repl** command, it shows that the REPLTYPE is Master for master replicates. For example:

  ```
  $cdr list repl
  CURRENTLY DEFINED REPLICATES
  ------------------------------
  REPLICATE: rep2
  STATE: Active ON:delhi
  CONFLICT: Timestamp
  FREQUENCY: immediate
  QUEUE SIZE: 0
  PARTICIPANT: test:nagaraju.tab12
  OPTIONS: transaction,ris,ats,fullrow
  REPLTYPE: Master

  REPLICATE: rep1
  STATE: Active ON:delhi
  CONFLICT: Timestamp
  FREQUENCY: immediate
  QUEUE SIZE: 0
  PARTICIPANT: test:nagaraju.tab
  OPTIONS: transaction,ris,ats,fullrow
  ```

  In the above output, **rep1** is defined as a non-master replicate and **rep2** is defined as master replicate.
- **Problem:** An alter operation on a replicated table fails.
  For example:

  ```
  $dbaccess test -

  Database selected.

  > alter table tab add col4 int;

  19995: Enterprise Replication error encountered while setting alter mode. See
  message log file to get the Enterprise Replication error code
  Error in line 1Near character position 27
  >
  ```

  The message log output is:

  ```
  12:36:09 CDRGC: Classic replicate rep1 found on the table test:nagaraju.tab
  12:36:09 CDRGC:Set alter mode for replicate rep1
  ```

```
12:36:09 GC operation alter mode set operation on a replicated table failed:
Classic replicate(s) (no mastered dictionary) found on the table.
```

Solution: The above message shows that there is a classic replicate, **rep1**, defined on the table (**tab**). Adding a new column to a replicated table is allowed when only master replicates are defined for the table.

To perform the above alter operation, first convert the classic replicate to a master replicate. You can convert the replicate definition of **rep1** to a master replicate by issuing the following command:

```
cdr remaster -M g_delhi rep1 "select * from tab"
```

Now look at the **cdr list repl** output:

```
$cdr list repl
CURRENTLY DEFINED REPLICATES
------------------------------
REPLICATE: rep1
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: rep2
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab12
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: Shadow_4_rep1_GMT1112381058_GID100_PID29935
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Shadow
PARENT REPLICATE: rep1
```

You can see that **repl1** has been converted to a master replicate. You can also see that a new replicate definition, **Shadow_4_rep1_GMT1112381058_GID100_PID29935**, was also created against the table (**tab1**). Notice the last two fields of the output for **Shadow_4_rep1_GMT1112381058_GID100_PID29935**:

```
REPLTYPE: Shadow
PARENT REPLICATE: rep1
```

The Shadow attribute indicates that this replicate is a shadow replicate, and PARENT REPLICATE: **rep1** shows that this is a shadow replicate for the primary replicate **rep1**. Notice that the Master attribute is not present for this replicate definition. This shadow replicate is actually the old non-master replicate. The **cdr remaster** command created a new master replicate, **rep1**, for the table **tab** and converted the old non-master replicate (**rep1**) to a shadow replicate for the new master replicate.
This table is not yet ready to be altered because there is still a non-master replicate, **Shadow_4_rep1_GMT1112381058_GID100_PID29935**, defined for the table, **tab**. You must wait for **Shadow_4_rep1_GMT1112381058_GID100_PID29935** to be deleted automatically by Enterprise Replication after all the data queued for this shadow replicate is applied at all the replicate participants. This process can take some time. Alternatively, if you are sure that there is no data pending for this old non-master replicate, then you can issue the **cdr delete repl** command against **Shadow_4_rep1_GMT1112381058_GID100_PID29935**.

After making sure that **Shadow_4_rep1_GMT1112381058_GID100_PID29935** no longer exists, you can attempt the **ALTER TABLE tab add col4 int;** statement against the table.

Enterprise Replication Event Alarms

Certain Enterprise Replication errors and other actions generate event alarms. You can use event alarms specific to Enterprise Replication to automate many administrative tasks.

You can set your alarm program script to capture Enterprise Replication class IDs and messages and initiate corrective actions or notifications for each event. For example, you can add a chunk to the queue data sbspace or dbspace if you detect (using class ID 31) that the storage space is full.

Most event alarms operate in the background. For events that operate in the foreground, the session that triggered the alarm is suspended until the alarm program execution completes. For information on setting alarm program scripts to capture events, see Event Alarms.

Many Enterprise Replication event alarms are enabled by default, but most state change event alarms are disabled by default. You can control which Enterprise Replication event alarms are enabled with the CDR_ALARMS environments variable.

The following table lists the information about Enterprise Replication event alarms:

- The class ID is an integer value identifying the category of the event.
- The event ID is a unique identifier for the specific message.
- The class message provides general information about the event.
- The specific message provides detailed information about the event.
- The severity describes the seriousness of the event on a scale from 1 to 5, where 5 is the most serious.

- Whether the event operates in the foreground and explanations for the events.
- Whether the event is disabled by default.

Table 1. Enterprise Replication Event Alarms

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 30 Event ID: 30002	Class message: DDR subsystem notification Specific message: DDR Log Snooping - Catchup phase started, userthreads blocked	3	User transactions are being blocked to prevent the database server from overwriting a logical log that Enterprise Replication has not yet processed. **Online log:** The following message appears in the online log: `DDR Log Snooping - DDRBLOCK phase started, userthreads blocked` **ER state:** Active and replicating data. User transactions are temporarily blocked. **User action:** None. For information about preventing this situation, see Handle potential log wrapping.
Class ID: 30 Event ID: 30003	Class message: DDR subsystem notification Specific message: DDR Log Snooping - Catchup phase completed, userthreads unblocked	3	User transactions are no longer blocked. **Online log:** The specific message also appears in the online log. **ER state:** Active and replicating data. **User action:** None.
Class ID: 30 Event ID: 30004	Class message: DDR subsystem failure Specific message: WARNING: The replay position was overrun, data may not be replicated.	4	The log replay position was overwritten. **Online log:** The following message appears in the online log: `WARNING: The replay position was overrun, data may not be replicated.` **ER state:** Active and replicating data. Enterprise Replication shuts down if the log read position also gets overwritten. If Enterprise Replication shuts down, event alarm 47 is raised. **User action:** For information about preventing this situation, see Handle potential log wrapping.
Class ID: 30 Event ID: 30005	Class message: DDR Subsystem notification Specific message: CDR DDR: Log staging disk space usage reached its allowed configured maximum size *size* (KB). Temporarily disabling log staging.	3	The disk space where logs are stored reached its maximum size. **Online log:** The following message appears in the online log: `CDR DDR: Log staging disk space usage reached its allowed configured maximum size` *`size`* `(KB). Temporarily disabling log staging.` **ER state:** Active and running. Enterprise Replication uses the next configured logical log lag action to protect the replay position. If no other log lag action is configured, the replay position can be overrun. If Enterprise Replication shuts down due to replay position being overrun, restart Enterprise Replication using **cdr cleanstart** command and resynchronize the data. **User action:** Consider increasing the maximum disk space configured for log staging using the CDR_LOG_STAGING_MAXSIZE configuration parameter. The value for the CDR_LOG_STAGING_MAXSIZE configuration parameter can be updated while the server is active using the following command: **`onmode -wf CDR_LOG_STAGING_MAXSIZE=`***`size`*
Class ID: 30 Event ID: 30006	Class message: DDR Subsystem notification Specific message: CDR: Created staging file *filename* for log unique id *unique_log_id*	3	The log staging file was created. **Online log:** The following message appears in the online log: `CDR: Created staging file` *`filename`* `for log unique id` *`uniqui_log_id`* **ER state:** Enterprise Replication is active and staging log files because a log lag state was detected. **User action:** If high-availability secondary servers are configured, consider copying log files to the secondary server. See Transferring log files to a high-availability cluster secondary server when using ER.
Class ID: 30 Event ID: 30007	Class message: DDR Subsystem notification Specific message: CDR: Completed processing log unique id *unique_log_id*. Deleted log staging file *filename*	2	A log staging file was deleted. **Online log:** The following message appears in the online log: `CDR: Completed processing log unique id` *`unique_log_id`*`. Deleted log staging file` *`filename`* **ER state:** Active and replicating data. **User action:** If staged log files are being copied to high-availability secondary servers, consider deleting the log staged log file name specified in the alarm message and the related token log file. See Transferring log files to a high-availability cluster secondary server when using ER.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 30 Event ID: 30008	Class message: DDR Subsystem notification Specific message: CDR: Deleted all staging files from log staging directory.	2	The staging files were deleted from the log staging directory. **Online log:** The following message appears in the online log: `CDR: Deleted all staging files from log staging directory.` **ER state:** Active or deleted. Enterprise Replication deletes all files in the log staging directory when they are no longer required. The log files are deleted when any of the following occur: • Enterprise Replication is deleted on the local server. • After the **cdr cleanstart** command is run. • When the value of the LOG_STAGING_DIR configuration parameter is changed (any log files that exist in the previous directory are also deleted). • When Enterprise Replication is defined. **User action:** If staged log files are being manually copied to high-availability secondary server then delete all staged log files on the secondary servers. See Transferring log files to a high-availability cluster secondary server when using ER.
Class ID: 31 Event ID: 31001	Class message: ER stable storage pager sbspace is full Specific message: CDR Pager: Paging File full: Waiting for additional space in *sbspace_name*	4	This event runs in the foreground. The grouper paging sbspace ran out of space. **ER state:** Active and waiting for the space to be added to the sbspace name specified in alarm-specific message. **User action:** Add a chunk to the specified sbspace. For information about preventing this situation, see Increasing the Sizes or Numbers of Storage Spaces.
Class ID: 31 Event ID: 31002	Class message: ER stable storage queue sbspace is full Specific message: CDR QUEUER: Send Queue space is FULL - waiting for space in *sbspace_name*. CDR QUEUER: Send Queue space is FULL - waiting for space in CDR_QDATA_SBSPACE	4	This event runs in the foreground. The storage space of a queue is full. **Online log:** The specific message also appears in the online log. **ER state:** Active and waiting for space to be added to the sbspace listed. **User action:** Add a chunk to the specified sbspace. If the message specifies CDR_QDATA_SBSPACE, add a chunk to one or more of the sbspaces specified by the CDR_QDATA_SBSPACE configuration parameter. For information about preventing this situation, see Recovering when Storage Spaces Fill.
Class ID: 31 Event ID: 31003	Class message: ER stable storage queue dbspace is full Specific message: CDR QUEUER: Send Queue space is FULL - waiting for space in CDR_QHDR_DBSPACE..	4	This event runs in the foreground. The storage space of a queue is full. **Online log:** The specific message also appears in the online log. **ER state:** Active and waiting for space to be added to the dbspace specified by the CDR_DBSPACE configuration parameter. **User action:** Add a chunk to the dbspace specified by the CDR_DBSPACE configuration parameter. For information about preventing this situation, see Recovering when Storage Spaces Fill.
Class ID: 32 Event ID: 32002	Class message: ER: error detected in grouper sub component Specific message: CDR Grouper Fanout thread is aborting.	4	The grouper fanout thread is quitting. **ER state:** Enterprise Replication was shut down internally. Event alarm 47 is also raised. **User action:** Restart Enterprise Replication using the **cdr start** command.
Class ID: 32 Event ID: 32003	Class message: ER: error detected in grouper sub component Specific message: CDR Grouper Evaluator thread is aborting.	4	The grouper evaluator thread is quitting. **ER state:** Active and replicating transactions. **User action:** Stop Enterprise Replication with the **cdr stop** command and restart it using the **cdr start** command.
Class ID: 32 Event ID: 32004	Class message: ER: error detected in grouper sub component Specific message: CDR: Could not copy transaction at log id *log_unique_id* position *log_position*. Skipped.	4	The grouper subcomponent cannot copy the transaction into the send queue. **ER state:** Active and replicating transactions. **User action:** Shut down Enterprise Replication by running the **cdr stop** command, clear the receive queue and restart replication by running the **cdr cleanstart** command, and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 32 Event ID: 32005	Class message: ER: error detected in grouper sub component Specific message: CDR: Paging error detected.	4	The grouper subcomponent detected a paging error. **ER state:** Inactive. **User action:** Restart Enterprise Replication by running the **cdr start** command.
Class ID: 32 Event ID: 32006	Class message: ER: error detected in grouper sub component Specific message: CDR Grouper: Local participant (*participant_name*) stopped for the replicate *replicate_name* (or exclusive replicate set), table (*database*:*owner*.*table*). Data may be out of sync. If replicated column definition was modified then please perform the alter operation at all the replicate participants, remaster the replicate definition then restart the replicate (or exclusive replicate set) definition for the local participant with the data sync option (**-S**).	4	If the grouper subcomponent is not able to convert the replicated row data from the local dictionary format to the master dictionary format, the grouper stops the local participant from the corresponding replicate (or exclusive replicate set) definition and invokes this event alarm.
Class ID: 32 Event ID: 32007	Class message: ER: error detected in grouper sub component Specific message: CDR *CDR_subcomponent_name*: Could not apply undo properly. SKIPPING TRANSACTION. TX Begin Time: *datetime* TX Restart Log Id: *log_id* TX Restart Log Position: *log_position* TX Commit Time: *datetime* TX End Log Id: *log_id* TX End Log Position: *log_position*	3	The grouper subcomponent did not roll back a transaction to a savepoint. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 33 Event ID: 33001	Class message: ER: error detected in data sync sub component Specific message: Received aborted transaction, no data to spool.	2	Data sync received a transaction that was aborted in the first buffer, so the transaction cannot be spooled to an ATS or RIS file. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 33 Event ID: 33002	Class message: ER: error detected in data sync sub component Specific message: CDR DS *thread_name* thread is aborting.	4	The data sync thread is quitting. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 33 Event ID: 33002	Class message: ER: error detected in data sync sub component Specific message: Table in alter mode is blocking application of transactions. Table: *dbname*:'*owner*'.*tabname*.	3	A table in alter mode is blocking the application of transactions. While the table is in alter mode, Enterprise Replication cannot apply transactions that involve this table or are in a referential relationship with this table. Enterprise Replication also cannot apply subsequent transactions from the site where the failed transaction originated. **User action:** Run `cdr alter --off dbname:owner.tabname`.
Class ID: 34 Event ID: 34001	Class message: ER: error detected in queue management sub component Specific message: CDR *CDR_subcomponent_name*: bad replicate ID *replicate_id*	3	This event runs in the foreground. RQM cannot find the replicate in the global catalog for which it has a transaction. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 35 Event ID: 35001	Class message: ER: error detected in global catalog sub component Specific message: CDR GC peer request failed: command: *command_string*, error *error_code*, CDR server *CDR_server_ID*	3	Execution of the control command requested by the peer server failed at the local server. **ER state:** Active and replicating transactions. **User action:** Correct the problem identified by the error code. Make sure that the replicate object is the same across all participating servers.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 35 Event ID: 35002	Class message: ER: error detected in global catalog sub component Specific message: CDR GC peer processing failed: command: *command_string*, error *error_code*, CDR server *CDR_server_ID*	3	Control command execution at the peer server failed. **ER state:** Active and replicating transactions. **User action:** Correct the problem identified by the error code. Make sure that the replicate object is the same across all participating servers.
Class ID: 35 Event ID: 35004	Class message: ER: error detected in global catalog sub component Specific message: CDR: Could not drop delete table. SQL code *sql_error_code*, ISAM code *isam_error_code*. Table '*database:table*'. Please drop the table manually.	3	The delete table was not dropped while the replicate was being deleted from the local participant. **ER state:** Active and replicating transactions. **User action:** Manually drop the delete table.
Class ID: 36 Event ID: 36001	Class message: ER: enterprise replication network interface sub component notification Specific message: Enterprise Replication: Connection to *servergroupname* closed. Reason: connection request received from an unknown server.	3	Enterprise Replication received a reconnect connection request from an unknown server. **ER state:** Active. **User action:** Check the connection requester server definition in the local server. If the definition is not available on the local server, the remote server definition was probably deleted on the local server by running the **cdr delete server** command, but the **cdr delete server** command was not run on the remote server. In this case, run the **cdr delete server** command on the remote server and, if necessary, redefine the server.
Class ID: 37 Event ID: 37001	Class message: ER: error detected while recovering Enterprise Replication Specific message: CDR *CDR_subcomponent_name*: bad replicate ID *replicate_id*	3	This event runs in the foreground; Enterprise Replication is blocked until this issue is resolved. **ER state:** Active and replicating transactions. **User action:** If the replicate ID is still valid and exists in **syscdr** catalog tables, run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 38 Event ID: 38001	Class message: ER: resource allocation problem detected Specific message: CDR *CDR_subcomponent_name* memory allocation failed (*reason*).	2	The specified Enterprise Replication component did not allocate memory. **ER state:** Active. **User action:** Perform these actions: 1. Correct the resource issue. 2. Stop replication by running the **cdr stop** command. 3. Restart replication by running the **cdr start** command. 4. Make sure that the data is consistent by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 39 Event ID: 39001	Class message: Please notify IBM® Informix® Technical Support Specific message: Log corruption detected or read error occurred while snooping logs.	4	A logical log is corrupted and cannot be processed by the log capture component. Event alarm 47 is also raised in this situation. **Online log:** The following message appears in the online log: `Log corruption detected while snooping logs, logid=`*`log_unique_id`* `logpos=`*`log_position`*`.` **ER state:** Inactive. **User action:** Clear the receive queue and restart replication by running the **cdr cleanstart** command, and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 39 Event ID: 39002	Class message: Please notify IBM Informix Technical Support Specific message: CDR: Unexpected log record type *record_type* for subsystem *subsystem* passed to DDR.	4	A log record of unexpected type was passed to the log capture component. **ER state:** Active and replicating transactions. **User action:** Contact Software Support.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 47 Event ID: 47001	Class message: CDR is shutting down due to internal error: failure Specific message: CDR is shutting down due to internal error: Memory allocation failed	4	Data sync threads encountered a memory allocation error while replaying replicated transactions and replication is stopped. **Online Log:** When the memory allocation error is discovered, the following message appears in the online log: `CDR DS processes is aborting. Signaling CDR system to shutdown as it is low on resources.` When Enterprise Replication is shutting down and the event alarm is being raised, the following message appears in the online log: `CDR is shutting down due to internal error: Memory allocation failed` **ER State:** No replicated transactions are lost while replication is stopped. **User Action:** To resume replication, solve the memory issue and run the **cdr start** command or shut down and restart the database server. If the replay position was overrun while replication was stopped, event alarm 75 is raised.
Class ID: 47 Event ID: 47005	Class message: CDR is shutting down due to internal error: failure Specific message: CDR is shutting down due to an internal error.	4	Enterprise Replication stopped. **ER state:** Inactive. **User action:** Try restarting Enterprise Replication using the **cdr start** command. If replay position overrun is detected and the **cdr start** command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data.
Class ID: 47 Event ID: 47006	Class message: CDR is shutting down due to internal error: log lag state Specific message: CDR DDR: Shutting down ER to avoid a DDRBLOCK situation.	4	Enterprise Replication stopped. **ER state:** Inactive. **User action:** If replay position overrun was detected then restart Enterprise Replication using **cdr cleanstart** command and synchronize the data. If the replay position was not overrun then restart Enterprise Replication using **cdr start** command; there is no need to synchronize the data. If replay position overrun is detected and the **cdr start** command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data.
Class ID: 48 Event ID: 48001	Class message: ATS and/or RIS files spooled to disk. Specific message: *file name*|*file name*.	3	One or more failed transactions caused the generation of one or more ATS or RIS files. The generated file names are listed in the specific message, separated with a pipe (|) character. **ER State:** Replication is continuing normally. **User Action:** To process the failed transactions, run the **cdr repair** command for each file, or run the **cdr check replicateset** command with the **--repair** option.
Class ID: 49 Event ID: 49001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is started on server *server_name*.	3	This event alarm is disabled by default. The **cdr start** command was run.
Class ID: 50 Event ID: 50001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is stopped on server *server_name*.	3	The **cdr stop** command was run.
Class ID: 51 Event ID: 51001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is suspended on server *server_name* .	3	This event alarm is disabled by default. The **cdr suspend server** command was run.
Class ID: 52 Event ID: 52001	Class message: A replication state change event has happened. Specific message: Enterprise Replication is resumed on server *server_name*.	3	This event alarm is disabled by default. The **cdr resume server** command was run.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 53 Event ID: 53001	Class message: A replication state change event has happened. Specific message: Server *server_name* is connected.	3	This event alarm is disabled by default. The **cdr connect server** command was run.
Class ID: 54 Event ID: 54001	Class message: A replication state change event has happened. Specific message: Server *server_name* is disconnected.	3	This event alarm is disabled by default. The **cdr disconnect server** command was run.
Class ID: 55 Event ID: 55001	Class message: A replication state change event has happened. Specific message: Replication is suspended on replicate *replicate_name* on server *server_name* .	3	This event alarm is disabled by default. The **cdr suspend replicate** command was run.
Class ID: 56 Event ID: 56001	Class message: A replication state change event has happened. Specific message: Replication is suspended on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr suspend replicateset** command was run.
Class ID: 57 Event ID: 57001	Class message: A replication state change event has happened. Specific message: Replication is resumed on replicate *replicate_name* on server *server_name* .	3	This event alarm is disabled by default. The **cdr resume replicate** command was run.
Class ID: 58 Event ID: 58001	Class message: A replication state change event has happened. Specific message: Replication is resumed on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr resume replicateset** command was run.
Class ID: 59 Event ID: 59001	Class message: A replication state change event has happened. Specific message: Replication is started on replicate *replicate_name* on server *server_name* .	3	This event alarm is disabled by default. The **cdr start replicate** command was run.
Class ID: 60 Event ID: 60001	Class message: A replication state change event has happened. Specific message: Replication is started on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr start replicateset** command was run.
Class ID: 61 Event ID: 61001	Class message: A replication state change event has happened. Specific message: Replication is stopped on replicate *replicate_name* on server *server_name.*	3	This event alarm is disabled by default. The **cdr stop replicate** command was run.
Class ID: 62 Event ID: 62001	Class message: A replication state change event has happened. Specific message: Replication is stopped on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr stop replicateset** command was run.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 63 Event ID: 63001	Class message: A replication state change event has happened. Specific message: Replication attribute is modified on replicate *replicate_name* on server *server_name* .	3	This event alarm is disabled by default. The **cdr modify replicate** command was run.
Class ID: 64 Event ID: 64001	Class message: A replication state change event has happened. Specific message: Replication attribute is modified on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr modify replicateset** command was run.
Class ID: 65 Event ID: 65001	Class message: A replication state change event has happened. Specific message: Change in replicate *replicate_name* on server *server_name*: operation *action*, node[s] *participant_name*.	3	This event alarm is disabled by default. The **cdr change replicate** command was run to add or delete one or more participants.
Class ID: 66 Event ID: 66001	Class message: A replication state change event has happened. Specific message: Change in replicateset *replicateset_name* on server *server_name*: operation *action*, member[s] *replicate_name*.	3	This event alarm is disabled by default. The **cdr change replicateset** command was run to add or delete one or more replicates.
Class ID: 67 Event ID: 67001	Class message: A replication state change event has happened. Specific message: Server *server_name* is deleted.	3	This event alarm is disabled by default. The **cdr delete server** command was run.
Class ID: 68 Event ID: 68001	Class message: A replication state change event has happened. Specific message: Replicate *replicate_name* is deleted on server *server_name*.	3	This event alarm is disabled by default. The **cdr delete replicate** command was run.
Class ID: 69 Event ID: 69001	Class message: A replication state change event has happened. Specific message: Replicate set *replicateset_name* is deleted on server *server_name*.	3	This event alarm is disabled by default. The **cdr delete replicateset** command was run.
Class ID: 70 Event ID: 70001	Class message: A replication state change event has happened. Specific message: Server *server_name* is modified.	3	This event alarm is disabled by default. The **cdr modify server** command was run.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 71 Event ID: 71001	Class message: ER: Network connection disconnected. Specific message: Network connection was dropped from the server *server_name* to the server *server_name*. Connection closed due to an Enterprise Replication administrative activity.	3	The connection was closed as the result of an Enterprise Replication command, such as **cdr stop**, **cdr disconnect server**, or **cdr delete server**. This event alarm appears on the database server on which the command was run and might or might not appear on the peer server. The peer server might receive event alarm 71 with the specific message that the connection closed for an unknown reason because the administrative control message might not reach the peer server before the connection is closed. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name, and that an administrative command was run. **ER State:** How replication is affected and how to reestablish the connection depends on which command closed the connection. • If the **cdr stop** command was run, replicated transactions are no longer being captured from this database server. • If the **cdr disconnect server** command was run, replicated transactions continue to be captured and queued. • If the **cdr delete server** command was run, the database server is no longer a participant in the replication domain and no replicated data is captured on or for this database server. **User Action:** Solve the issue that prompted the running of the administrative command and reestablish the connection between the servers.
Class ID: 71 Event ID: 71002	Class message: ER: Network connection disconnected. Specific message: Network connection was dropped from the server *server_name* to the server *server_name*. Connection closed due to the idle time-out set for the replication server.	3	An idle timeout occurs when there are no replication messages sent between the replication servers for the number of seconds specified as the idle timeout period. The connection is reestablished automatically when replication messages are ready to be sent. This event alarm appears on both database servers affected by the dropped connection. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name, and the reason of idle timeout. **ER State:** Replication continues normally. **User Action:** None. Replication resumes automatically. You can increase or eliminate the idle timeout period by using the **cdr modify server** command.
Class ID: 71 Event ID: 71003	Class message: ER: Network connection disconnected. Specific message: Network connection was dropped from the server *server_name* to the server *server_name*. Connection unexpectedly closed for an unknown reason.	3	This event alarm occurs when there is a connection problem not related to Enterprise Replication, such as a network outage or one of the database servers shutting down. This event alarm might appear on both database servers affected by the dropped connection. This alarm does not appear on a database server that shut down. This alarm might appear when a peer server closed the connection with an administrative activity, in which case that server receives event alarm 71 with the specific message that an administrative activity closed the connection. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name. **ER State:** Replicated transactions continue to be captured and queued, except on database servers that are shut down. Replicated transactions to and from the affected servers are not transmitted. **User Action:** Examine both servers to determine the cause of the dropped connection. • If there was a network problem, solve it and restart any database servers that might be shut down. The Enterprise Replication connection reconnects automatically. • If there was an administrative action, solve the issue that prompted the running of the administrative command and reestablish the connection between the servers.

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 73 Event ID: 73001	Class message: Enterprise replication NIF connection terminated. Specific message: Enterprise Replication: Connection to *server_name* closed. Reason: CDR server *server_name* not found.	3	The remote server initiated a connection request that was not completed by the local server. This alarm appears when the remote server has an sqlhosts entry for the local server, but the local server does not have a corresponding sqlhosts entry for the remote server. This situation can occur when a new server is added to the domain but the local server does not have an entry for that server in its sqlhosts file. **Online Log:** The specific message also appears in the online log. **ER State:** The new server cannot participate in replication until the sqlhosts entries are correct. Replication between the established replication servers continues normally. **User Action:** To solve this issue, update the sqlhosts entry on the local server with the appropriate entry for the remote server. Make sure that all the sqlhosts files are consistent on all replication servers in domain.
Class ID: 74 Event ID: 74001	Class message: Enterprise replication recovery failed Specific message: Server name/id mismatch in sqlhosts file while recovery, recovered name = *server_name*, id = *ID*, current name = *server_name*, id = *ID*	3	The server information in the sqlhosts file was updated after the server was defined for replication. This alarm can appear after the following sequence of events: 1. Replication is stopped on a server because the **cdr stop** command was run or the server was shut down. 2. The sqlhosts file was updated. 3. Replication was attempted to be restarted by running the **cdr start** command or by starting the server. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Update the sqlhosts file to restore the original server information and then restart replication by running the **cdr start** command or restarting the database server.
Class ID: 75 Event ID: 75001	Class message: ER: the logical log replay position is not valid. Restart ER with the **cdr cleanstart** command, and then synchronize the data with the **cdr check --repair** command. Specific message: The replay position (logical log ID *log_number* and log position *log_position*) has been overwritten.	4	This event alarm occurs if a database server was shut down or replication was stopped for long enough to fill a logical log. When the database server was restarted or the **cdr start** command was run, replication failed. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Clear the receive queue and restart replication by running the **cdr cleanstart** command and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 75 Event ID: 75002	Class message: ER: the logical log replay position is not valid. Restart ER with the **cdr cleanstart** command, and then synchronize the data with the **cdr check --repair** command. Specific message: The replay position (logical log ID *log_number* and log position *log_position*) is later than the current position.	4	This event alarm can occur after a point in time restore was performed on the database server. The point in time restore applied log records beyond the current replay position. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Clear the receive queue and restart replication by running the **cdr cleanstart** command and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 76 Event ID: 76001	Class message: A replication state change event has happened. Specific message: Server *server_name* is disabled.	3	This event alarm is disabled by default. The **cdr disable server** command was run.
Class ID: 77 Event ID: 77001	Class message: A replication state change event has happened. Specific message: Server *server_name* is enabled.	3	This event alarm is disabled by default. The **cdr enable server** command or the **cdr check replicateset** command with the **--enable** option was run.

- Enabling or Disabling Enterprise Replication Event Alarms
  You can control which Enterprise Replication event alarms can be raised.

Related concepts:
Monitor and troubleshooting Enterprise Replication
Related reference:
cdr delete replicate
cdr delete replicateset

cdr delete server
cdr change replicate
cdr change replicateset
cdr connect server
cdr start
cdr stop
cdr suspend server
cdr resume server
cdr delete template
cdr suspend replicate
cdr suspend replicateset
cdr resume replicate
cdr resume replicateset
cdr start replicate
cdr start replicateset
cdr stop replicate
cdr stop replicateset
cdr modify replicate
cdr modify replicateset
cdr modify server

Enabling or Disabling Enterprise Replication Event Alarms

You can control which Enterprise Replication event alarms can be raised.

By default, Enterprise Replication event alarms are enabled, except most of the state change event alarms that are raised by specific **cdr** commands. You might want to enable state change event alarms to track which **cdr** commands are being run, but if you are setting up your replication system and running many **cdr** commands, the resulting large number of event alarms might affect system performance.

To change which Enterprise Replication event alarms are enabled, reset the values of the CDR_ENV configuration parameter for the CDR_ALARMS environment variable:

1. Add a new line or update the existing line for CDR_ENV CDR_ALARMS in the onconfig file. List all the Enterprise Replication event alarms that you want to be enabled.
2. Restart the database server.

Example

The following example shows the CDR_ENV value in the onconfig file for the CDR_ALARMS environment variable with event alarm 49 enabled in addition to the default event alarms:

```
CDR_ENV CDR_ALARMS=30-39,47-50,71,73-75
```

Related reference:
CDR_ALARMS Environment Variable
CDR_ENV Configuration Parameter

Push data feature

Push data feature lets clients register for changes in a dataset using simple SELECT statements and WHERE clauses. Once the server captures data for push data event conditions which evaluates to true for WHERE clause condition, the server pushes committed data to the client, based on registered events. Scaling is achieved by clients not having to poll for data, and not having to parse, prepare, and execute SQL queries. Database servers with parallel architecture – Enterprise Replication log snooper and grouper -- feed the data to all clients by asynchronously reading logical log file changes. This design lets client applications scale linearly without adding significant overhead to the database server or any OLTP applications making changes to the database. Data that is returned to the client is in a developer-friendly JSON format.

Table 1. . JSON attributes for registering new event conditions:

Input attribute name	Description
table	Table name to be registered
owner	Table owner
database	Database name
query	SELECT statement including projection list and WHERE clause to register for changes in a data set.
label	User defined string to be returned along with an event document. This attribute is useful to differentiate between events when more than one push-data event is registered within the same session

Input attribute name	Description
timeout	The amount of time a client is blocked in the smartblob read API for an event data. The server returns timeout json document when a timeout condition is triggered. Supported range of values are: • -1 to wait forever • >=0 to wait for a specified amount of time in seconds.
commit_time	Returns event data that is committed after the stated transaction commit time.
txnid	A unique 8 byte ID: • Higher order 4 bytes: commit work log ID • Lower order 4 bytes: commit work log position
max_pending_ops	Maximum number of event records to be kept in the pending session .
maxrecs	Maximum number of records to be returned by the smartblob API read call.

Grant replication permission on sysadmin database for the user registering push data events:

```
execute function task('grant admin', 'user1', 'replication');
```

Register client as a push data session by using the sysadmin task command:

```
execute function informix.task('pushdata open')
```

The above command registers the client as a push-data session, and returns a unique session ID. This ID is needed for reading event documents using the smartblob readAPI.

This command also auto-registers enterprise replication, when it has not been defined earlier.

To internally define enterprise replication automatically, the pushdata open command relies on the existence of at least one storagepool entry to create the dbspace and subspace required for defining enterprise replication. You must create a storagepool entry using the task API.

For example:

```
Execute function task( 'storagepool add', '/informix/storage', '0', '0', '20000', '1' );
```

Registering one or more push data event conditions using the sysadmin task command:

```
execute function informix.task('pushdata register', {table:"creditcardtxns",owner:"informix",database:"creditdb",query:"select
uid, cardid,
 carddata from creditcardtxns where carddata.Amount::int >= 100",label:"card txn alert"})
```

Registering session-specific attributes, like timeout, using the pushdata register task command:

```
execute function informix.task('pushdata register', { timeout:"60",max_pending_ops:"0",maxrecs:"1"})
```

De-registering one or more registered event conditions using the pushdata deregister command:

```
To de-register one or more event conditions for the given table:
execute function informix.task('pushdata deregister', {table:"usertable",owner:"informix",database:"ycsb"})
```

To de-register all event conditions with the same label attribute tag:

```
execute function informix.task('pushdata deregister', { label:"card txns"})
```

Note: To deregister a specific event condition, either use the label attribute, or specify a query attribute, along with the table, owner and database attributes.
API to read event data:
The client must invoke the smartblob read API to read an event data. Input for the smartblob read API must include:

- The session ID returned from running the pushdata open task command.
- The input buffer pointer
- The input buffer size—this should be at least equal to the sum of the before image size, the after image size, and 1024 bytes. If multiple records are expected from one read call, then the input buffer size should be equal to the sum of the before image size, the after image size, and 1024, multiplied by the number of records.
- The error code pointer.

ESQLC READ API Example:

```
/*
 * Read data into the buffer
 */
 bytesread = ifx_lo_read(sessionid, databuf, bytes_per_read, &loreaderr);
```

Table 2. . Event data JSON attributes

Attribute name	Description
operation	Operation type: Insert/Delete/Update
table	Table name
owner	Table owner

Attribute name	Description
database	Database name
label	Optional user-specified data for the event condition
txnid	8 byte unique ID: • higher order 4 bytes: commit work log ID • lower order 4 bytes: commit work log position.
operation_owner_id	User id of the user executing the IUD operation.
operation_session_id	Session id of the session executing the IUD operation
commit_time	Transaction commit time for the event data.
op_num	Increasing sequence number for the event document within a given transaction. If the transaction generates 10 events, then each document returned will have an incremental op_num value, starting from 1 to 10.
restart_logid	Restart (replay) position logical log unique id. This position may be used to reset Enterprise replication capture position upon server failure using ‘ "pushdata reset_capture’ ADMIN/TASK API.
restart_logpos	Restart (replay) position logical log position within the given log unique id. This position may be used to reset Enterprise replication capture position upon server failure using ‘ "pushdata reset_capture’ ADMIN/TASK API.
rowdata	Row data in JSON document format. Data is returned using the column name as key and the column data as value.
before_rowdata	Before row data for an Update operation.
ifx_isTimeout	Document with this attribute is returned with its value set to true if no event gets triggered within the timeout interval that is specified by the client.
ifx_warn_total_skipcount	A warning document with this attribute is returned, containing the cumulative number of events that are discarded, from exceeding the max_pending_ops attribute threshold.

Sample output from the smartblob read API for an Insert operation:

```
{“operation”:"insert",”table”:"creditcardtxns",”owner”:"informix",
”database”:"creditdb",”label”:"card txn alert",”txnid”:2250573177224,”operation_owner_id”:201,”operation_session_id”:200,
”commit_time”:1488243530,”op_num”:1,”restart_logid”:31,”restart_logpos”:24,”rowdata”:{“uid”:22,
”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}
```

Sample output from the smartblob read API for Update operation:

```
{“opertion”:"update",table:"creditcardtxns",”owner”:"informix",”database
”:"creditdb",”label”:"card txn alert",”txnid”:2250573308360, ”operation_owner_id”:201,”operation_session_id”:205,
”commit_time”:1488243832,”op_num”:1,”restart_logid”:31,”restart_logpos”:24,”rowdata”:{uid:21,cardid:"7777-7777-7777-7777",
”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":"25-Jan-2017 16:15"} },”before_rowdata”:{“uid”:21,
”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-01T10:35:10.000Z } }}
```

Sample output from the smartblob read API for Delete operation:

```
{“opertion”:"delete",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",
”txnid”:2250573287760,”operation_owner_id”:201,
”operation_session_id”:209,”commit_time”:1488243797,”op_num”:1,”restart_logid”:31,”restart_logpos”:24,
”rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-
01T13:35:06.000Z } }}
```

Sample output from the smartblob read API for a multi-row buffer, when the maxrecs input attribute is set to greater than 1:

```
{[
{“operation”:"insert",”table”:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn alert",
“txnid”:2250573309999, ,”operation_owner_id”:201,”operation_session_id”:212,”commit_time”:1487781325,”op_num”:1,
”rowdata”:{uid:"7",”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-
01T15:10:10.000Z } }},
{“operation”:"insert",table:"creditcardtxns",”owner”:"informix",”database”:"creditdb",”label”:"card txn
alert",”txnid”:2250573177224,
”operation_owner_id”:201,”operation_session_id”:215,”commit_time”:1488243530,”op_num”:1,”restart_logid”:31,”restart_logpos”:24,
”rowdata”:{“uid”:22,”cardid”:"6666-6666-6666-6666",”carddata”:{"Merchant":"Sams Club","Amount":200,"Date":2017-05-
01T16:20:10.000Z } }}
]}
```

Using the sample *pushdata* ESQL/C program

You can run the *pushdata* ESQL/C program to safely preview the process of registering event triggers with your Informix server, to retrieve event data in JSON format.

The program file, **pushdata.ec**, can be found in the *$INFORMIXDIR/demo/cdc* folder of your Informix installation folder.

- Push data session survival
- Detach trigger

Push data session survival

By default, if client gets disconnected from server, server deletes push-data event conditions for the session, and server no longer capture the event data. The following APIs allow server to capture the event data for the push-data client if client is temporarily disconnected from the server.

Detachable sessions can survive from server failure. Client can re-attach to detachable sessions after bouncing the server or after primary server failover operation in a cluster environment with secondary servers. After bouncing server, it is possible for the pushdata client to get duplicate records. Client need to save last received record txnid and op_num and discard the processed records. *restart_logid* and *restart_logpos* are added to INSER, UPDATE and DELETE records returned to the push data sessions. After server failover or upon restarting Informix server, client can re-attach to the detached session and reset Enterprise replication capture position back to *restart_logid* and *restart_logpos* using 'pushdata reset_capture' sysadmin task/admin API.

Sample record for Insert operation:

```
{"operation":"insert","table":"t1","owner":"informix","database":"testdb","txnid":133151498608,
"operation_owner_id":37188,"operation_session_id":67,"commit_time":1568823415,"op_num":4,"restart_logid":31,
"restart_logpos":24,"rowdata":{"c1":4001,"c2":4001 }}
```

- To mark a session as a detachable session:

  ```
  execute function informix.task('pushdata setdetach') into :retvalstr;
  ```

  This API returns unique session id.
- Optional command to recapture data from a previous log position after bouncing server or primary server failover operation:

  ```
  execute function admin("pushdata reset_capture", '{"logid":"%d", "logpos":"%d"}');
  ```

- This command API stops Enterprise Replication(ER) and restarts ER with the given log position set to replay position. This API impacts all pushdata sessions and existing replicate definitions in Enterprise Replication.

  Note: Care must be taken to set logical log unique id and position to the last known *restart_logid* and *restart_logpos* returned to push data session. *restart_logid* and *restart_logpos* are added to INSERT, UPDATE and DELETE records returned to the push data sessions. After server failover or upon restarting Informix server, user can attach back to the detached session and reset Enterprise replication capture position back to *restart_logid* and *restart_logpos*.
- To re-join a detached session after reconnecting to server:

  ```
  execute function informix.task('pushdata join', "{session_id:"245"}) into :retvalstr;
  ```

  Returns smartblob file descriptor to read event data. Unique session id returned from 'pushdata setdetach' API is used as input to 'pushdata join' API. 'pushdata join' returns smartblob file descriptor -- same as 'pushdata open'.
  Note: 'pushdata open' and 'pushdata join' APIs should not be called in the same session.
  'pushdata open' is used to establish new push-data session, and 'pushdata join' API is used to join existing detached push-data session.
- To delete "detachable" push-data session information from the server:
  - To delete currently attached push-data session:

    ```
    execute function informix.task('pushdata delete') into :retvalstr;
    ```

    This is only required if the push-data session is marked as a detachable session using API 'pushdata setdetach'.
  - To delete a specific unique session id currently not attached to any client application:

    ```
    execute function task('pushdata delete', '{session_id:"12"}') ;
    ```

  - To delete all push-data sessions that are not attached to any client applications:

    ```
    execute function task('pushdata delete', '{delete_all:"1"}');
    ```


Detach trigger

Using the detach trigger methods in **IfmxThreadedSmartTrigger**, you can declare a Smart Trigger to be 'detachable'. A detachable trigger has an unique identifier which allows you to reconnect to the session on the server.

```
/* Detach a trigger */
IfxSmartTrigger push = new IfxSmartTrigger("jdbc-url-here");
push.detachable(true); //Set the trigger as detachable
push.open();
String session1 = push.getDetachableSessionID(); //Get the session id
//Closes the JDBC connection and returns the session ID
//This is the same session id as you get from the call above
Session1 = push.detach();
```

On detaching from the session, you can create a new Smart Trigger object and pass in the session ID.

```
push = new IfxSmartTrigger("jdbc-url-here");
//Assign the session ID before you start the smart trigger
push.sessionID(sessionID);
TestPushCallback callback1 = new TestPushCallback();
push.registerCallback("test-label-pushtest", callback1);
push.start();
//You pick up where you left off, retrieving any messages you missed from the server
```


Loopback Replication

You can setup replication between tables within the same Informix server with loopback replication. Source and target tables can be in the same database or on two different databases within the same server.

This section discusses the intended audience and the associated software products that you must have to use Enterprise Replication.

Loopback Configuration

To setup loopback replication, define pseudo ER group in SQLHOSTS, and define new service within DBSERVERALIAS other than the one used for primary ER group name. Service name added for loopback pseudo group should appear after service name used for primary Enterprise Replication group.

SQLHOSTS file example:

```
g_er_server group - - i=1
er_server onsoctcp *myhost 17001  g=g_er_server
g_loopback group - - i=2
loopback onsoctcp *myhost 17002  g= g_loopback
```

DBSERVERALIAS example:

```
DBSERVERALIAS er_server,loopback
```

where g_er_server is the primary group name for the local server, g_loopback is the pseudo ER group name for the local server.

Replication definition between primary and pseudo groups

Replicate is defined only using **primary->target configuration**, and replication definition should include only primary group and pseudo group participants.

Note: Only **ignore** and **always apply** conflict resolutions rules are supported for loopback replication.

```
$ cdr define repl --connect=g_er_server test_t1 --conflict=always --scope=row --ats --ris --floatieee --master=g_er_server
"P test@g_er_server:informix.t1" "select * from t1" "R test@g_loopback:informix.t2"  "select * from t2"
```

In the above replicate definition is established between table t1 and table2 in test database on the same server
All ER commands including 'cdr sync', 'cdr check', template commands work with pseudo group. The only restriction is that pseudo ER group cannot be added to grid definition, and grid commands are not supported with pseudo ER group.

Template example

Template can be used to define replication across multiple tables from two different databases.

```
$ cdr define template --connect=g_er_server ifxt_test_test --conflict=always --scope=row --ats --ris --floatieee --
master=g_er_server --database=test  --all
$ cdr realize template --connect=g_er_server ifxt_test_test "test@g_er_server"
$ cdr realize template --connect=g_loopback ifxt_test_test --target "test2@g_loopback"
```

The above commands define replication between tables in test and test2 databases on the same server
'cdr list replicate' command show 'loopback' replication attribute

```
REPLICATE:         rep1
STATE:             Active ON:g_informix
CONFLICT:          Always Apply
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       ycsb:informix.usertable
OPTIONS:           row,ris,ats,fullrow,loopback
REPLID:            65537 / 0x10001
REPLMODE:          PRIMARY  ON:g_informix
APPLY-AS:          INFORMIX ON:g_informix
REPLTYPE:          Master
```

Sample procedure to reorg table online

1. Create RAW table with appropriate fragmentation strategy
   Note: Create standard table if you have secondary servers.
2. Initial data load -- Unload and load data from old table to new table
3. Create indexes
4. Convert table type to "standard"
5. Define replication and repair data using 'cdr check --repair' command
   Note: Define replicate with --name=n option, if you plan to rename table while replication is still active.
6. Use rename table DDL or synonym to point applications to new table
7. Drop replicate once queues are empty

Sample procedure for online migration of database codeset

1. Create new database with target codeset and schema
2. Unload data from old database using appropriate CLIENT and DB locales, and load data into new database use CLIENT_LOCALE set to unloaded data codeset format, and DB_LOCALE set to target database codeset

3. Define replication using –utf8 option and execute data re-synchronization task using **cdr check –repair** or **cdr sync replset** commands
4. Use rename database DDL to point applications to new database

Appendixes

- The cdr utility
  You use the **cdr** utility to configure and control Enterprise Replication from the command line on your UNIX or Windows operating system.
- Enterprise Replication configuration parameter and environment variable reference
  You can use configuration parameters and environment variables to configure the behavior of Enterprise Replication.
- Grid routines
  Grid routines are used to create and maintain the grid and to administer servers in the grid by propagating commands from a source server to all other servers in the grid.
- Enterprise Replication routines
  Enterprise Replication routines used to control if a replicated transaction is recaptured.
- onstat -g commands for Enterprise Replication
  You can monitor and debug Enterprise Replication activity using **onstat -g** commands.
- syscdr Tables
  These tables in the **syscdr** database contain progress information about consistency checking and synchronization operations.
- SMI Tables for Enterprise Replication Reference
  The system-monitoring interface (SMI) tables in the **sysmaster** database provide information about the state of the database server. Enterprise Replication uses the following SMI tables.
- Replication Examples
  This appendix contains simple examples of replication using the command-line utility (CLU).
- Data sync warning and error messages
  Data sync warning and error messages describe problems with replicated transactions.

The cdr utility

You use the **cdr** utility to configure and control Enterprise Replication from the command line on your UNIX or Windows operating system.

You must be the Enterprise Replication server administrator to run any of the **cdr** commands except the **cdr list** commands, unless otherwise noted.

The **cdr** utility requires a certain amount of memory resources. If you encounter out-of-memory errors for **cdr** commands, your operating system limits on memory use might be set too low. For example, you can run the **ulimit** command in a UNIX environment to configure limits on memory resources. Increase the values for memory resources to avoid out-of-memory errors.

You can run **cdr** commands from within SQL statements by using the SQL administration API. Most **cdr** commands that perform actions are supported by the SQL administration API; **cdr** commands that show information are not supported.

- Interpret the cdr utility syntax
  The **cdr** utility uses specific terminology and conventions.
- cdr add onconfig
  The **cdr add onconfig** command adds one or more values to a configuration parameter in the ONCONFIG file.
- cdr alter
  The **cdr alter** command puts the specified tables in alter mode.
- cdr autoconfig serv
  The **cdr autoconfig serv** command can autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can automatically configure replication.
- cdr change grid
  The **cdr change grid** command adds or deletes replication servers to or from a grid.
- cdr change gridtable
  The **cdr change gridtable** command has multiple uses. It can verify that specific tables can be used in gird queries and then add the tables to a list of verified tables, and it can delete tables from the list of verified tables.
- cdr change onconfig
  The **cdr change onconfig** command replaces the existing value of a configuration parameter with a new value in the ONCONFIG file.
- cdr change replicate
  The **cdr change replicate** command modifies an existing replicate by adding or deleting one or more participants.
- cdr change replicateset
  The **cdr change replicateset** command changes a replicate set by adding or deleting replicates.
- cdr change shardCollection
  The **cdr change shardCollection** command changes the sharding definition that determines which database servers are part of the shard cluster.
- cdr check catalog
  The **cdr check catalog** command compares the metadata information related to servers, replicates and replicate sets on replication servers for any inconsistency.
- cdr check queue
  Use the **cdr check queue** command to check the consistency of Enterprise Replication metadata, and to check the consistency of user data before running critical tasks in the Enterprise Replication domain. The command returns successfully when all of the commands that were queued when **cdr check queue** was run are complete.
- cdr check replicate
  The **cdr check replicate** command compares the data on replication servers to create a report that lists data inconsistencies and can optionally repair the inconsistent data within a replicate.

- cdr check replicateset
  The **cdr check replicateset** command compares the data on replication servers to create a report listing data inconsistencies. Optionally you can use the command to repair the inconsistent data within a replicate.
- cdr check sec2er
  The **cdr check sec2er** command determines whether a high availability cluster can be converted to replication servers.
- cdr cleanstart
  The **cdr cleanstart** command starts an Enterprise Replication server with empty queues.
- cdr connect server
  The **cdr connect server** command reestablishes a connection to a database server that has been disconnected with a **cdr disconnect server** command.
- cdr define grid
  The **cdr define grid** command creates a named grid of replication servers to simply administration.
- cdr define qod
  The **cdr define qod** command defines a master server for monitoring the quality of data (QOD) for replication servers.
- cdr define region
  The **cdr define region** command creates a region that contains a subset of the servers in a grid.
- cdr define replicate
  The **cdr define replicate** command defines a replicate on the specified replication servers.
- cdr define replicateset
  The **cdr define replicateset** command defines a replicate set on all the servers that are included as participants in the replicates. A replicate set is a collection of several replicates to be managed together.
- cdr define server
  The **cdr define server** command defines a replication server in an Enterprise Replication domain. You can add a replication server to an existing domain or create a new domain.
- cdr define shardCollection
  The **cdr define shardCollection** command creates a sharding definition for distributing a table or collection across multiple shard servers.
- cdr define template
  The **cdr define template** command creates a template for replicates and a replicate set.
- cdr delete grid
  The **cdr delete grid** command deletes the specified grid.
- cdr delete region
  The **cdr delete region** command deletes a region from a grid.
- cdr delete replicate
  The **cdr delete replicate** command deletes a replicate.
- cdr delete replicateset
  The **cdr delete replicateset** command deletes an exclusive or non-exclusive replicate set from the global catalog.
- cdr delete server
  The **cdr delete server** disables a database server from participating in Enterprise Replication.
- cdr delete shardCollection
  The **cdr delete shardCollection** command deletes a sharding definition, and then stops data sharding.
- cdr delete template
  The **cdr delete template** command deletes a template from the replication domain. It also deletes any underlying replicate sets associated with the template (these will exist if the template has been realized). No replicates are deleted.
- cdr disable grid
  The **cdr disable grid** command removes the authorization to run grid routines from users or servers.
- cdr disable server
  The **cdr disable server** command disables replication on a server.
- cdr disconnect server
  The **cdr disconnect server** command stops a server connection.
- cdr enable grid
  The **cdr enable grid** command authorizes users to run commands on the grid and designates servers from which grid commands can be run.
- cdr enable server
  The **cdr enable server** command enables replication on a replication server that was disabled by the **cdr disable server** command.
- cdr error
  The **cdr error** command manages the **syscdrerror** table and provides convenient displays of errors.
- cdr finderr
  The **cdr finderr** command looks up a specific Enterprise Replication return code and displays the corresponding error text.
- cdr list grid
  The **cdr list grid** command shows information about a grid.
- cdr list replicate
  The **cdr list replicate** command displays information about the replicates on the current server.
- cdr list replicateset
  The **cdr list replicateset** command displays information about the replication sets defined on the current server.
- cdr list server
  The **cdr list server** command displays a list of the Enterprise Replication servers that are visible to the server on which the command is run.
- cdr list shardCollection
  The **cdr list shardCollection** command displays the sharding definition for all database servers in a shard cluster.
- cdr list catalog
  The **cdr list catalog** command lists the commands that created the specified replication objects.
- cdr list template
  The **cdr list template** command displays information about the templates on the server on which the command is run.
- cdr migrate server
  The **cdr migrate server** command automates data migration task between two or more servers. This command also automates setting up of Enterprise Replication between two Informix server instances.
- cdr modify grid
  The **cdr modify grid** command modifies grid attributes.
- cdr modify replicate
  The **cdr modify replicate** command modifies replicate attributes.
- cdr modify replicateset
  The **cdr modify replicateset** command modifies all the replicates in a replicate set.

- cdr modify server
  The **cdr modify server** command modifies the Enterprise Replication attributes of a database server.
- cdr realize template
  The **cdr realize template** command creates the replicates, replicate set, and participant tables as specified in a template, and then synchronizes data on all or a subset of the database servers within the replication domain.
- cdr remaster
  The **cdr remaster** command changes the SELECT clause or the server from which to base the master replicate definition of an existing master replicate. This command can also convert a classic (non-master) replicate to a master replicate.
- cdr remaster gridtable
  The **cdr remaster gridtable** command validates tables in a grid after an alter operation.
- cdr remaster replicateset
  The **cdr remaster replicateset** command updates the definitions of the set of replicates whose participants were changed by ALTER operations.
- cdr remove onconfig
  The **cdr remove onconfig** command removes the specified value from a configuration parameter in the ONCONFIG file.
- cdr repair
  The **cdr repair** command synchronizes data based on ATS or RIS files.
- cdr reset qod
  The **cdr reset qod** command resets failed-transaction counts for replicates on replicate servers. Connection Manager service-level agreements (SLA) that contains a FAILURE or LATENCY redirection policy use failed-transaction counts to determine where to route client requests.
- cdr resume replicate
  The **cdr resume replicate** command resumes delivery of replication data.
- cdr resume replicateset
  The **cdr resume replicateset** command resumes delivery of replication data for all the replicates in a replicate set.
- cdr resume server
  The **cdr resume server** command resumes delivery of replication data to a suspended database server.
- cdr start
  The **cdr start** command starts Enterprise Replication processing.
- cdr start qod
  The **cdr start qod** command starts quality of data (QOD) monitoring for replication servers.
- cdr start replicate
  The **cdr start replicate** command starts the capture and transmittal of replication transactions.
- cdr start replicateset
  The **cdr start replicateset** command starts the capture and transmittal of replication transactions for all the replicates in a replicate set.
- cdr start sec2er
  The **cdr start sec2er** command converts a high availability cluster to replication servers.
- cdr stats rqm
  The **cdr stats rqm** command displays information about the reliable queue manager (RQM) queues used for Enterprise Replication.
- cdr stats recv
  The **cdr stats recv** command displays receiver parallelism statistics and latency statistics by source node.
- cdr stats check
  The **cdr stats check** command displays the progress of a consistency check that specified a progress report task name.
- cdr stats sync
  The **cdr stats sync** command displays the progress of a synchronization operation that specified a progress report task name.
- cdr stop
  The **cdr stop** command stops replication on the server to which you are connected without shutting down the database server.
- cdr stop qod
  The **cdr stop qod** command stops quality of data (QOD) monitoring for replication servers.
- cdr stop replicate
  The **cdr stop replicate** command stops the capture, transmittal, and reception of transactions for replication.
- cdr stop replicateset
  The **cdr stop replicateset** command stops capture and transmittal transactions for all the replicates in a replicate set.
- cdr suspend replicate
  The **cdr suspend replicate** command suspends delivery of replication data.
- cdr suspend replicateset
  The **cdr suspend replicateset** command suspends delivery of replication data for all the replicates in a replicate set.
- cdr suspend server
  The **cdr suspend server** command suspends the delivery of replication data to a database server from either a specified list of database servers or from all database servers in the domain.
- cdr swap shadow
  The **cdr swap shadow** command switches a replicate with its shadow replicate during manual remastering.
- cdr sync replicate
  The **cdr sync replicate** command synchronizes data among replication servers to repair inconsistent data within a replicate.
- cdr sync replicateset
  ASDF The **cdr sync replicateset** command synchronizes data among replication servers to repair inconsistent data within a replicate set.
- cdr -V
  The **cdr -V** command displays the version of Informix® that is currently running.
- cdr view
  The **cdr view** command shows information about every Enterprise Replication server in the domain.

Related tasks:
Repairing Failed Transactions with ATS and RIS Files
Related information:
cdr argument: Administer Enterprise Replication (SQL administration API)

Interpret the cdr utility syntax

The **cdr** utility uses specific terminology and conventions.

Each **cdr** command follows the same approximate format, with the following components:

- Command and its variation
  The command specifies the action that is taken.
- Options
  The options modify the action of the command. Each option starts with a minus (-) or a double-minus (--).
- Target
  The target specifies the Enterprise Replication object that is acted upon.
- Other objects
  Other objects specify objects that are affected by the change to the target.

If you enter an incorrect **cdr** command at the command-line prompt, the database server returns a usage message that summarizes the **cdr** commands. For a more detailed usage message, enter **cdr** *variation* **-h**. For example, **cdr list server -h**.

- Command Abbreviations
  For most commands, each of the words that make up a **cdr** command variation can be abbreviated to three or more characters.
- Option Abbreviations
  Each option for a **cdr** command has a long form and a short form. You can use either form, and you can mix long and short forms within a single command.
- Option Order
  You can specify the options of the **cdr** commands in any order. Some of the syntax diagrams show the options in a specific order because it makes the diagram easier to read.
- Long Command-Line Examples
  The examples in this guide use the convention of a backslash (\) to indicate that a long command line continues on the next line.
- Long Identifiers
  Identifier names used in **cdr** commands follow the guidelines of SQL syntax.
- Connect Option
  Most **cdr** commands allow a connect option to specify the database server to connect to for performing the command.
- Participant and participant modifier
  A participant defines the data (database, table, and columns) to be replicated on a specific database server. You can choose whether to allow the participant to both send and receive replicated data, or to only receive or only send replicated data. You can choose to check for table owner permissions when applying operations. By default, permissions are not checked. The participant modifier is a restricted SELECT statement that specifies the rows and columns that are replicated.
- Return Codes for the cdr Utility
  If a **cdr** command encounters an error, the database server returns an error message and a return code value.
- Frequency Options
  You can specify the interval between replications or the time of day when replication occurs for a replicate.

Related concepts:
Enterprise Replication Server administrator

Command Abbreviations

For most commands, each of the words that make up a **cdr** command variation can be abbreviated to three or more characters.

For example, the following commands are all equivalent:

```
cdr define replicate
cdr define repl
cdr def rep
```

The exceptions to this rule are the **replicateset** commands, which can be abbreviated to **replset**. For example, the following commands are equivalent:

```
cdr define replicateset
cdr def replset
```

Command abbreviations are not allowed when you run **cdr** commands within SQL statements using the SQL administration API. For more information, see the *IBM® Informix® Administrator's Reference*.

Related concepts:
Option Abbreviations
Option Order
Long Command-Line Examples
Long Identifiers
Connect Option
Return Codes for the cdr Utility
Frequency Options
Related reference:
Participant and participant modifier

Option Abbreviations

Each option for a **cdr** command has a long form and a short form. You can use either form, and you can mix long and short forms within a single command.

On UNIX, a long form example might look like:

```
cdr define server --connect=ohio --idle=500 \
   --ats=/cdr/ats --initial utah
```

On WINDOWS, the same long form example would look like:

```
cdr define server --connect=ohio --idle=500 \
   --ats=D:\cdr\ats --initial utah
```

Using short forms, you can write the previous examples as follows:

UNIX:

```
cdr def ser -c ohio -i 500 -A /cdr/ats -I utah
```

WINDOWS:

```
cdr def ser -c ohio -i 500 -A D:\cdr\ats -I utah
```

The long form is always preceded by a double minus (--). Most (but not all) long forms require an equal sign (=) between the option and its argument. The short form is preceded by a single minus (-) and is usually the first letter of the long form. The short form never requires an equal sign. However, sometimes the short form is capitalized and sometimes it is not. To find the correct syntax for the short form, check the table that accompanies each command variation.
Tip: Use the long forms of options to increase readability.
With the exception of the keyword **transaction**, all keywords (or letter combinations) that modify the command options must be written as shown in the syntax diagrams. For example, in the Conflict Options, the option **conflict** can be abbreviated, but the keyword **ignore** cannot be abbreviated. Both of the following forms are correct:

```
--conflict=ignore
-C ignore
```

Related concepts:
Command Abbreviations
Option Order
Long Command-Line Examples
Long Identifiers
Connect Option
Return Codes for the cdr Utility
Frequency Options
Related reference:
Participant and participant modifier

Option Order

You can specify the options of the **cdr** commands in any order. Some of the syntax diagrams show the options in a specific order because it makes the diagram easier to read.

Do not repeat any options. The following fragment is incorrect because **-c** appears twice. In most cases, if you duplicate an option you will receive an error. However, if no error is given, the database server uses the last instance of the option. In the following example, the database server uses the value `-c utah`:

```
-c ohio -i 500 -c utah
```

Tip: For ease of maintenance, always use the same order for your options.
Related concepts:
Command Abbreviations
Option Abbreviations
Long Command-Line Examples
Long Identifiers
Connect Option
Return Codes for the cdr Utility
Frequency Options
Related reference:
Participant and participant modifier

Long Command-Line Examples

The examples in this guide use the convention of a backslash (\) to indicate that a long command line continues on the next line.

The following two commands are equivalent. The first command is too long to fit on a single line, so it is continued on the next line. The second example, which uses short forms for the options, fits on one line.

On UNIX, the command line might look like:

```
cdr define server --connect=katmandu --idle=500 \
   --ats=/cdrfiles/ats
```

```
cdr def ser -c katmandu -i 500 -A /cdrfiles/ats
```

On Windows, these command lines might look like:

```
cdr define server --connect=honolulu --idle=500 \
   --ats=D:\cdrfiles\ats
```

```
cdr def ser -c honolulu -i 500 -A D:\cdr\ats
```

For information on how to manage long lines at your command prompt, check your operating system documentation.

Related concepts:
Command Abbreviations
Option Abbreviations
Option Order
Long Identifiers
Connect Option
Return Codes for the cdr Utility
Frequency Options
Related reference:
Participant and participant modifier

Long Identifiers

Identifier names used in **cdr** commands follow the guidelines of SQL syntax.

Identifiers are the names of objects, such as database servers, databases, columns, replicates, replicate sets, and so on, that Informix® and Enterprise Replication use.

An identifier is a character string that must start with a letter or an underscore. The remaining characters can be letters, numbers, or underscores. On IBM® Informix, all identifiers, including replicates and replicate sets, can be 128 bytes long. However, if you have any database servers in your replication environment that are an earlier version, you must follow the length restrictions for that version.

For more information about identifiers, see the *IBM Informix Guide to SQL: Syntax*.

The length of a path and file name, such as the names of ATS files, can be 256 bytes.

User login IDs can be a maximum of 32 bytes. The owner of a table is derived from a user ID and is thus limited to 32 bytes.

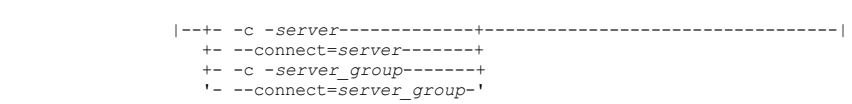
Related concepts:
Command Abbreviations
Option Abbreviations
Option Order
Long Command-Line Examples
Connect Option
Return Codes for the cdr Utility
Frequency Options
Related reference:
Participant and participant modifier

Connect Option

Most **cdr** commands allow a connect option to specify the database server to connect to for performing the command.

The **--connect** option causes the command to use the global catalog that is on the specified server. If you do not specify this option, the connection defaults to the database server specified by the **INFORMIXSERVER** environment variable.

Connect Option



Element	Purpose	Restrictions	Syntax
server	Name of the database server to connect to	The name must be the name of a database server or server connection.	Long Identifiers
server_group	Name of the database server group that includes the database server to connect to	The name must be the name of an existing database server group.	Long Identifiers

You must use the **--connect** option when you add a database server to your replication environment with the **cdr define server** command.

You might use the **--connect** option if the database server to which you would normally attach is unavailable.

If your replication domain contains database servers that are running different server versions, **cdr** commands must connect to the server running the latest version of IBM® Informix®. If you are connected to a database server running an older version of IBM Informix, you cannot run a **cdr** command on a database server running a later version of IBM Informix.

If the database server uses trusted connections between replication servers by including the **s=6** option in the sqlhosts entries, you configure a regular connection to an alias of the server for the **cdr** utility to use. In a trusted connection environment, the **cdr** utility can only connect to the local replication server.

Related concepts:
Command Abbreviations
Option Abbreviations
Option Order
Long Command-Line Examples
Long Identifiers
Return Codes for the cdr Utility
Frequency Options
Enterprise Replication Terminology
Creating sqlhost group entries for replication servers
Related tasks:
Connect to another replication server
Configuring secure ports for connections between replication servers
Related reference:
Participant and participant modifier
Related information:
The onpassword utility

Participant and participant modifier

A participant defines the data (database, table, and columns) to be replicated on a specific database server. You can choose whether to allow the participant to both send and receive replicated data, or to only receive or only send replicated data. You can choose to check for table owner permissions when applying operations. By default, permissions are not checked. The participant modifier is a restricted SELECT statement that specifies the rows and columns that are replicated.

Syntax

```
Participant

     .-P-. .-I-.
|--"-+---+-+---+-database@server_group:owner.table-"------------|
     +-R-+ '-O-'
     '-S-'

Participant Modifier

              .-,----------.
              V            |
|--"SELECT--+-----column---+-+--FROM--table--WHERE_Clause"------|
            '-*--------------'
```

Element	Purpose	Restrictions	Syntax
column	Name of a column in the table that is specified by the participant. The replication key columns must be included.	The column must exist.	Long Identifiers
database	Name of the database that includes the table to be replicated.	The database server must be registered with Enterprise Replication.	Long Identifiers
owner	User ID of the owner of the table to be replicated.		Long Identifiers
server_group	Name of the database server group that includes the server to connect to.	The database server group name must be the name of an existing Enterprise Replication server group in the sqlhosts information and must be used only once in the same replicate.	Long Identifiers
table	Name of the table to be replicated. Must be the same table name in the participant and participant modifier.	The table must be an actual table. It cannot be a synonym or a view.	Long Identifiers
WHERE_Clause	Clause that specifies a subset of table rows to be replicated.	Can include opaque user-defined types that are always stored in row. Cannot contain a column of a **TimeSeries** data type.	WHERE Clause of SELECT

The following table describes the participant options.

Option	Meaning
I	Default. Disables the table-owner option (**O**).

Option	Meaning
O	Enables permission checks for table owner that is specified in the participant to be applied to the operation (such as INSERT or UPDATE) that is to be replicated and to all actions fired by any triggers. When the **O** option is omitted, all operations are run with the privileges of user **informix** or the server owner. On UNIX, if a trigger requires any system-level commands (as specified by the **system()** command in an SPL statement), the system-level commands are run as the table owner, if the participant includes the **O** option. On Windows, if a trigger requires any system-level commands, the system-level commands are run as a less privileged user because you cannot impersonate another user without having the password, whether the participant includes the **O** option.
P	For primary-target replicates, specifies that the participant is a primary participant, which both sends and receives replicated data. Do not use for an update-anywhere replicate. Enterprise Replication defines all the participant as primary in an update-anywhere replication system.
R	For primary-target replicates, specifies that the participant is a receive-only target participant, which only receives data from primary participants.
S	For primary-target replicates, specifies that the participant is a send-only primary participant, which only sends data to target participants. You cannot use this option for replicates that include **TimeSeries** columns.

Usage

Each participant in a replicate must specify a different database server. The participant definition includes the following information:

- Database in which the table is located
- Table name
- Table owner
- Participant type
- Participant modifier with a SELECT statement

You must include the server group, database, table owner, and table name when you define a participant, and enclose the entire participant definition in quotation marks.

If you use a SELECT * FROM *table_name* statement, the tables must be identical on all database servers that are defined for the replicate, unless you implement a data consolidation system by defining one server to receive data and several other servers that only send data.

Restriction: Do not create more than one replicate definition for each row and column combination to replicate. If the participant overlaps, Enterprise Replication attempts to insert duplicate values during replication.
You can define participants with the following commands:

- **cdr define replicate**
- **cdr modify replicate**
- **cdr change replicate**
- **cdr define template**

The following restrictions apply to a SELECT statement that is used as a participant modifier:

- The statement cannot include a join or a subquery.
- The statement cannot run operations on the selected columns.
- The statement cannot exceed 15 000 ASCII characters in length.
- For tables that have **TimeSeries** columns, all columns must be included.

Replicate only between like data types. For example, do not replicate between the following combinations of data types:

- CHAR(40) to CHAR(20)
- INT to FLOAT

You can replicate between the following types with caution:

- SERIAL and INT
- BYTE and TEXT
- BLOB and CLOB

Note: The ERKEY shadow columns are not included in the participant definition if you use SELECT * in your participant modifier. To include the ERKEY shadow columns in the participant definition, use the **--erkey** option with the **cdr define replicate**, **cdr change replicate**, or **cdr remaster** commands.

Example 1: Defining update-anywhere participants

If you do not specify the participant type, Enterprise Replication defines the participant as update-anywhere by default. For example:

```
"db1@g_hawaii:informix.mfct" "select * from mfct" \
"db2@g_maui:informix.mfct" "select * from mfct"
```

Example 2: Defining a primary server

For example, in the following participant definition, the **P** indicates that in this replicate, **hawaii** is a primary server:

```
"P db1@g_hawaii:informix.mfct" "select * from mfct"
```

If any data in the selected columns changes, that changed data is sent to the secondary servers.

Example 3: Defining a server that only receives data

In the following example, the **R** indicates that in this replicate, **maui** is a server that only receives data:

```
"R db2@g_maui:informix.mfct" "select * from mfct"
```

The specified table and columns receive information that is sent from the primary server. Changes to those columns on **maui** are *not* replicated.

Example 4: Defining a data consolidation system with servers that only send data

To implement a data consolidation system, you can define one server to receive and consolidate the data and configure several other servers that only send data. In the following example, the **S** options indicate that the **rome**, **tokyo**, **perth**, and **ny** servers can only send data:

```
"db0@london:user.world_sales" "select * from world_sales"\
"S db1@rome:user1.sales_rome" "select * from sales_rome"\
"S db2@tokyo:user2.sales_tokyo" "select * from sales_tokyo"\
"S db3@perth:user3.sales_perth" "select * from sales_perth"\
"S db4@ny:user4.sales_ny" "select * from sales_ny"\
```

The central server, **london**, is a standard replication server without restrictions on sending or receiving data.

Related concepts:
Command Abbreviations
Option Abbreviations
Option Order
Long Command-Line Examples
Long Identifiers
Connect Option
Return Codes for the cdr Utility
Frequency Options
Primary-Target Replication System
Participant definitions
Data consolidation
Primary-Target Data Dissemination
Related reference:
cdr define replicate
cdr modify replicate
cdr change replicate
cdr define template
cdr swap shadow

Return Codes for the cdr Utility

If a **cdr** command encounters an error, the database server returns an error message and a return code value.

The message briefly describes the error. For information about interpreting the return code, use the **cdr finderr** command.

The following table lists the return codes.

Table 1. Return codes for the cdr utility

Return code	Error text	Explanation
0	Command successful.	
1	A connection does not yet exist for the given server.	A replication server involved in the command is not connected to the server that is running the command. This error code can be returned when a **cdr sync** or **cdr check** task cannot switch connections between task participants. **User action:** Establish connections between all necessary replication servers and rerun the command.
3	Table column undefined.	A column name listed in the SELECT statement of the replicate participant definition is not found in the table dictionary. This error code can be returned if a shadow column name is included in the SELECT statement of the replicate definition. **User action:** Correct the SELECT statement of the participant definition.
4	Incompatible server version.	A **cdr** command originating on a database server running an older version of attempted to run on a database server running a later version of . **User action:** Run the command from the replication server running the most recent version of .

Return code	Error text	Explanation
5	Unable to connect to server specified.	A replication server involved in the command is not available for one of the following reasons: • The server disconnected from the domain. • Replication is no longer active on the server. • The server is offline. • The **--connect** option was not used and the **INFORMIXDIR** environment variable for the current server is not set. This error code can be returned if one of the **cdr sync** or **cdr check** task participants cannot be accessed or if a task participant became inactive or went offline while a sync or check task is in progress. This error code can be returned if the user running the **cdr define replicate** or **cdr change replicate** command does not have Connect privilege on the database specified for the replicated table. **User action:** Check the status of all participating servers and rerun the command when all servers are active.
6	Database does not exist.	The database name specified for the replicate in the command does not exist. **User action:** Verify the spelling of the database names and that they exist on each participant and rerun the command. This error code can be returned if the **cdr view** command is run and the **sysadmin** database does not exist.
7	Database not logged.	The database specified for the replicate in the command is a non-logging database. Replicated databases must be logged. **User action:** Change the database logging mode to buffered logging and rerun the command.
8	Invalid or mismatched frequency attributes.	The value for the **--at** or **--every** option is not within the range of valid values or is formatted incorrectly. **User action:** Rerun the command with valid and correctly formatted frequency values.
9	A connection already exists for the given server.	This error code can be returned if the **cdr connect server** command is run for a server that already has an active connection.
10	Invalid replicate set state change.	The replicate set specified in the command is not in the appropriate state for the command. This error code is returned in the following situations: • The **cdr stop replicateset** command is run but all replicates in the replicate set are not active. • The **cdr start replicateset** command is run but all replicates in the replicate set are already active. • The **cdr suspend replicateset** command is run but all replicates in the replicate set are not active. • The **cdr resume replicateset** command is run but all replicates in the replicate set are already active. **User action:** Run the **cdr list replicateset** and **cdr list replicate** commands to see the status of each replicate.
11	Undefined replicate set.	The specified replicate set does not exist or the replicate set is empty. The replicate set name might be incorrectly specified in the command. **User action:** Rerun the command with the correct replicate set name, or add replicates to the replicate set and then rerun the command.
12	Replicate set name already in use.	The replicate set name specified in the command is already being used. Replicate set names must be unique in the domain. **User action:** Run the **cdr list replicateset** command to view a list of replicate set names and then rerun the original command with a different replicate set name.
13	Invalid idle time specification.	The value for the **--idle** option is not within the range of valid values or is formatted incorrectly. **User action:** Rerun the command with a valid and correctly formatted value.
14	Invalid operator or specifier.	Both the **--ignoredel y** and the **deletewins** options were used in the same command. These options cannot be used together. **User action:** Rerun the command with only one of these options.
15	Invalid length.	The ATS or RIS directory path specified in the command exceeds 256 characters. This error can be returned if the server group name exceeds 127 characters. **User action:** Rerun the command with a shorter directory path or server group name.
16	Replicate is not a member of the replicate set.	The replicate specified to be deleted from the replicate set is not a member of the replicate set. **User action:** Run the **cdr list replicateset** command for the replicate set to view a list of replicates in the replicate set and then rerun the original command with the correct replicate name.
17	Participants required for operation specified.	One or more of the participants necessary for this command were not specified. This error code is returned if the sync source server or the target server is not defined as a participant for the **cdr sync** or **cdr check** task. This error code is also returned if the target participant list is empty. **User action:** Rerun the command with the required participants.
18	Table does not contain primary key.	The table specified in the command does not have a replication key. This error is returned if the **cdr sync** or **cdr check** task cannot find the replication key for the table being repaired. **User action:** Add a primary key constraint or the ERKEY shadow columns to the table and rerun the command. If you have another unique index on the replicated table, you can specify to use the columns in that index as the replication key when you define the replicate.

Return code	Error text	Explanation
19	Table does not exist.	The table owner name specified in the command is not correct. This error is also returned if the table owner name is not specified for a table in an ANSI database. **User action:** Rerun the command with the correct table owner name.
20	Server already participating in replicate.	The participant specified in the command is already a participant in the replicate.
21	Command timed out	The command timed out while waiting for queue monitoring to complete. **User action:** Check the server and connection status using the **cdr list server** command and, if needed, rerun the command and specify a longer timeout period.
22	Primary key not contained in select clause.	The replicate participant SELECT statement did not include the replication key columns. **User action:** Rerun the command including the replication key columns in the participant SELECT statement.
25	Replicate already participating in a replicate set.	The replicate specified to be added to the replicate set is already a member of the replicate set. **User action:** Run the **cdr list replicateset** command to view a list of replicates in the replicate set.
26	Replicate set operation not permitted on replicate.	The replicate specified to be deleted from the replicate set does not have a valid name. **User action:** Run the **cdr list replicateset** command to view a list of replicates in the replicate set and then rerun the original command with the correct replicate name.
28	Replicate name already in use.	The replicate name specified in the command is already being used. Replicate names must be unique in the domain. **User action:** Run the **cdr list replicate** command to view a list of replicate names and then rerun the original command with a different replicate name.
29	Table does not exist .	The table name specified in the command does not exist. **User action:** Rerun the command with an existing table name.
30	Invalid replicate state change.	The replicate specified in the command is not in the appropriate state for the command. This error code is returned in the following situations: • The **cdr stop replicate** command is run but the replicate is not active. • The **cdr start replicate** command is run but the replicate is already active. • The **cdr suspend replicate** command is run but the replicate is not active. • The **cdr resume replicate** command is run but the replicate is already active. **User action:** Run the **cdr list replicate** command to see the status of the replicate.
31	Undefined replicate.	The replicate name cannot be found in Enterprise Replication catalog tables. The name of the replicate might be incorrectly specified in the command. **User action:** Rerun the command with the correct replicate name.
32	sbspace specified for the send/receive queue does not exist	The CDR_QDATA_SBSPACE configuration parameter is not set to a valid sbspace name. **User action:** Set the CDR_QDATA_SBSPACE configuration parameter to a valid sbspace name in the onconfig file.
33	Server not participant in replicate/replicate set.	The server name specified in the command is not a participant in the replicate or replicate set. This error is returned if a server name is not valid. **User action:** To see a list of all participants for each replicate, query the **syscdrpart** view in the **sysmaster** database.
35	Server not defined in sqlhosts.	The server group name specified in the command is not defined in the sqlhosts file specified by the **INFORMIXSQLHOSTS** environment variable. **User action:** Check the sqlhosts file for the correct spelling of the server group name, or, if necessary, update the sqlhosts file to add the server group, and then rerun the original command.
37	Undefined server.	The target participant cannot be found in the Enterprise Replication catalog tables. The name of the server might be incorrectly specified in the command. **User action:** Rerun the command with the correct server name.
38	SPL routine does not exist.	The SPL routine specified with the **--conflict** option does not exist on one or more participants. **User action:** Make sure that the SPL routine exists on all participants and rerun the command.
39	Invalid select syntax.	The SELECT statement included in the command is not valid or is missing from the command. **User action:** Rerun the command with the correct SELECT statement.
40	Unsupported SQL syntax (join, etc.).	The SELECT statement contains syntax that is not supported for replicate participants. Syntax such as subqueries in the WHERE clause or selecting from multiple tables with a JOIN clause is not supported. **User action:** Rerun the command with the correct SELECT statement.
41	GLS files required for data conversion are not installed.	The GLS files required for data conversion to or from UTF-8 are not installed. Code set conversion files are installed with the Client SDK and are in the **$INFORMIXDIR/gls/cv9** directory.
42	Invalid time range.	The time range does not have valid values or is formatted incorrectly. **User action:** Rerun the command with a valid and correctly formatted time range.
43	Participants required for specified operation.	The command did not include the required participant information. **User action:** Rerun the command with participant information.
44	Invalid name syntax.	The name of a replicate or server in the command is not valid, for example, the name might be too long. **User action:** Rerun the command with a valid name.
45	Invalid participant.	The participant syntax is not valid. **User action:** Rerun the command with a valid participant syntax.

Return code	Error text	Explanation
47	Invalid server.	A connection between the current server and the specified server is not allowed. This error code is returned if a server attempts to connect to a leaf server that has a different parent server. This error code is also returned if the server specified in the **cdr repair** command does not exist in the Enterprise Replication catalog.
48	Out of memory.	Enterprise Replication cannot allocate enough memory for this command.
49	Maximum number of replicates exceeded.	The maximum number of replicates that can be defined from a particular server is exceeded. **User action:** Rerun the command while connected to a peer replication server.
52	Server name already in use.	A replication server with this group ID exists. **User action:** Run the **cdr list server** command to see a list of all replication server names and group IDs
53	Duplicate server or replicate.	A replication server or replicate name is listed more than once in the command. This error code is returned if the sync source server is also specified as a sync target server or if the same server is listed multiple times as a sync target participant. This error code is returned if the same group name is defined more than once in the sqlhosts file. **User action:** Rerun the command specifying each server and replicate one time.
54	Invalid conflict rule specified.	The conflict resolution rule is not correctly specified. This error code is returned for the **cdr define replicate** or **cdr modify replicate** command under the following circumstances: • A stored procedure is specified as the conflict resolution rule but the table has user-defined data types or collection data types. • A secondary conflict resolution rule is specified that is not a stored procedure conflict resolution rule. • A secondary conflict resolution rule is specified but the primary conflict resolution rule is not time stamp or delete wins. This error code is returned if the **--timestamp** option is used in a **cdr check** command and the replicate specified in the command does not use the time stamp or the delete wins conflict resolution rule. This error code is also returned when the **cdr check** command includes the **--deletewins** option but the specified replicate does not use the delete wins conflict resolution rule. **User action:** Correct the conflict resolution rule issue and rerun the command.
55	Resolution scope not specified.	The conflict resolution scope (row or transaction) is required for ER to resolve conflicts between replicated transactions. Scope is not required if the conflict resolution rule is **ignore**, in which case ER does not attempt to resolve conflicts. **User action:** Rerun the command with a conflict resolution scope.
56	Shadow columns do not exist for table.	A conflict resolution rule requires the **cdrtime** and **cdrserver** shadow columns but those columns do not exist in the replicated table. **User action:** Alter the replicated table to add the shadow columns by using the ADD CRCOLS clause and rerun the original command.
57	Error creating delete table.	The delete table corresponding to the replicated table was not created. **User action:** Check the server message log file for additional error messages.
58	No conflict resolution rule specified.	A conflict resolution rule was not specified in the command. **User action:** Rerun the command with the **--conflict** option to specify a conflict resolution rule.
61	User does not have permission to issue command.	The user running this command does not have the DBSA privilege at one of the participants in the command. **User action:** Acquire the DBSA privilege on all participants and rerun the command, or rerun the command as a user that has the DBSA privilege at all participants.
62	Enterprise Replication not active.	The command cannot run because Enterprise Replication is not active on the server. **User action:** Run the **cdr list server** command to see the status of the server.
63	Enterprise Replication already active.	The command cannot make Enterprise Replication active because ER is already active on the server. **User action:** Run the **cdr list server** command to see the status of the server.
64	Remote/cyclic synchronization not allowed.	The command to define a replication server was attempted on a remote server. **User action:** Rerun the command on the server that is being defined.
65	Server identifier already in use.	The server group ID is not unique. **User action:** Rerun the command with a unique server group ID.
66	No upper time for prune error.	The ending date value for the error pruning range was not specified. **User action:** Rerun the command with a valid ending date.
67	Error not found for delete or update.	The error sequence number does not exist in the errors table. **User action:** Run the **cdr error** command to see a list of error sequence numbers and then rerun the command with an existing number.
68	Invalid participant mode.	The participant type value is not valid. **User action:** Rerun the command with a valid participant type.
69	Conflict mode for replicate not ignore or always apply.	One or more replicate participants specified in the command is defined as receive-only and must use a conflict resolution rule of **ignore** or **always**. **User action:** Rerun the command with the **--conflict** option set to **ignore** or **always**.
70	Connect/disconnect to/from same server.	The command attempted to connect the local server to itself. **User action:** Rerun the command with a different server name.

Return code	Error text	Explanation
72	Cannot delete server with children.	The command did not delete the hub server because the hub server still has child servers. **User action:** Delete the child servers and then delete the hub server.
75	Request denied on limited server.	The command is not allowed on leaf servers. It is also not allowed on replication servers that are disabled. **User action:** If the server is in disabled mode, wait until the server is active and rerun the command.
77	Could not drop the Enterprise Replication database.	The **syscdr** database was not deleted because a client is accessing it. **User action:** Wait for the client to unlock the **syscdr** database and then rerun the command. If necessary, use the **--force** option to drop the **syscdr** database if Enterprise Replication was partially deleted.
78	Invalid ATS directory.	The ATS directory path specified in the command was not valid for one of the following reasons: • The path does not exist. • The path is not a directory. • The path is /dev/null (UNIX) or NUL (Windows). **User action:** Rerun the command with a valid ATS directory path.
79	Invalid RIS directory.	The RIS directory path specified in the command was not valid for one of the following reasons: • The path does not exist. • The path is not a directory. • The path is /dev/null (UNIX) or NUL (Windows). **User action:** Rerun the command with a valid RIS directory path.
80	Invalid conflict resolution change.	The conflict resolution rule of a replicate cannot be changed to **ignore** or from **ignore**.
84	No sync server.	A synchronization server must be specified if the replication server being defined is a non-root or leaf server. The first server in a replication domain must be a root server. **User action:** Rerun the command with the **--sync** option.
85	Incorrect participant flags.	The participant type or owner option included in the command was not valid. **User action:** Rerun the command with valid participant options.
86	Conflicting leaf server flags.	The **--nonroot** and **--leaf** options cannot be used together. **User action:** Rerun the command with only one of the options.
90	CDR connection to server lost, id *group_id*, name *groupname*. Reason: System clocks off by %d seconds.	The system clock times on the servers differ by more than 900 seconds.
91	Invalid server state change.	The server is already in the state indicated by the command. This error code can be returned if the **cdr suspend server** command is run on a server that is suspended or if the **cdr resume server** command is run on a server that is active. **User action:** Run the **cdr list server** command to see the status of the server.
92	CDR is already defined.	Enterprise Replication is already defined on this server.
93	Enterprise Replication is currently initializing.	Enterprise Replication cannot be stopped on the server because replication is in the process of being initialized. **User action:** Run the **cdr list server** command to see the status of the server. Rerun the command when replication is active.
94	Enterprise Replication is currently shutting down.	Enterprise Replication cannot be stopped on the server because replication is already in the process of being stopped. **User action:** Run the **cdr list server** command to see the status of the server. If necessary, rerun the command.
99	Invalid options or arguments passed to command.	One or more of the options included with this command are not valid options. **User action:** Rerun the command with valid options.
100	Fatal server error.	The command was not completed because of an unrecoverable error condition.
101	This feature of Enterprise Replication is not yet supported.	One of the participants included with this command is running a version of that does not support this command. **User action:** Rerun the command with valid options.
102	Root server cannot sync with non root or leaf servers.	The synchronization server must be a root server. The **--sync** option cannot specify a non-root or leaf server. **User action:** Rerun the command specifying a root server with the **--sync** option.
103	Invalid server to connect.	A non-root server can connect only to its parent or children servers. **User action:** Rerun the command specifying to connect to the parent or a child server.
105	UDR needed for replication was not found.	A user-defined type listed in the SELECT statement of the participant definition does not have one or more of the **streamread()**, **streamwrite()**, or **compare()** support routines. **User action:** Create the required routines for the user-defined type and rerun the command.
106	Setup necessary for UDR invocation could not be completed.	The set-up process necessary to run the **streamread()**, **streamwrite()**, or **compare()** routine for a user-defined type included in the participant definition failed. **User action:** Check to be sure that the required routines for the user-defined type exist. Create them if necessary and rerun the command.
107	Sbspace specified for the send/receive queue does not exist.	The sbspace specified for the CDR_QDATA_SBSPACE configuration parameter is not a valid name or does not exist. **User action:** Correct the value of the CDR_QDATA_SBSPACE configuration parameter or create the sbspace and rerun the command.

Return code	Error text	Explanation
110	Data types with out of row or multi-representational data are not allowed in a replicate where clause.	A replicate participant WHERE clause cannot include a data type that has out-of-row data, such as, a collection data type, a user-defined type, or a large object type. **User action:** Remove the column with out-of-row data from the participant WHERE clause and rerun the command.
111	Cannot have Full Rows off and use stored procedure conflict resolution.	The stored procedure conflict resolution rule requires full row replication. **User action:** Rerun the command without the **--fullrow=n** option.
112	The replicate set command could only be partially executed. Please run **cdr list replset ReplSetName** to check results.	The command was successful on some, but not all, of the replicates in the replicate set. **User action:** Run the **cdr list replicate** command to see the status of each replicate and run the appropriate command on each of the remaining replicates.
113	Exclusive Replset violation.	The specified replicate is a member of an exclusive replicate set, which requires this operation to be performed for the replicate set instead of for individual replicates. **User action:** Run the equivalent command for the replicate set.
115	The syscdr database is missing.	The **syscdr** database cannot be opened. **User action:** Check server message log file for any additional error messages. If you received this error code after running the **cdr delete server --force** command, no action is required on the server being deleted. Run the **cdr delete server** command to delete that server on all peer replication servers in the domain. If you receive this error after running the **cdr start** command, make sure that Enterprise Replication is defined on the local server, and if necessary, define it by running the **cdr define server** command.
116	Dbspace indicated by CDR_DBSPACE is invalid.	The dbspace specified as the value of the CDR_DBSPACE configuration parameter does not exist. **User action:** Correct the value of the CDR_DBSPACE configuration parameter or create the dbspace and rerun the command.
117	Enterprise Replication operation attempted on HDR secondary server.	Enterprise Replication commands are not valid on high-availability secondary servers. **User action:** Rerun the command on a high-availability primary server.
118	SQLHOSTS file has multiple entries either at group ID or group name.	There are multiple group definitions for the same group name in the sqlhosts file. **User action:** Update the sqlhosts file to make all group entries unique.
119	SQLHOSTS file has a problem with (g=) or (i=) option.	The group name specified in the command is not found in the sqlhosts file. **User action:** Rerun the command with a valid group name or update the sqlhosts file and then rerun the command.
120	Cannot execute this command while ER is active.	Enterprise Replication cannot be deleted on this server because replication is still active. **User action:** Run the **cdr stop** command and then rerun the **cdr delete server --force** command.
121	Master participant not found.	The replication server that is specified as the master server in the command does not exist or is not a participant in the specified replicate. **User action:** Rerun the command with the correct master server name.
122	Attempt to perform invalid operation including shadow replicates.	The replicate specified in the command has shadow replicates, which prevent the command from completing. **User action:** Run the **cdr list replicate** command to see shadow replicate information. Wait for the shadow replicate to be deleted and then rerun the original command. If you are deleting the replicate, delete the shadow replicate and then rerun the original command.
123	Attempt to include an invalid participant in a shadow replicate.	The command attempted to add a participant to a shadow replicate that does not exist in the primary replicate. **User action:** Rerun the command with a valid participant.
124	Invalid command passed to cdrcmd function.	An argument that is not valid was passed to an internal routine. **User action:** Contact Software Support.
125	An error occurred concerning a mastered replicate.	The server specified as the master server in the command is not included as a participant in the replicate. This error code is returned if the mastered dictionary verification fails when adding a participant to a mastered replicate. This error code is returned if Enterprise Replication encounters an internal error during master replicate definition. **User action:** Rerun the command with the master server included in the participant list or check the table dictionary.
126	Invalid template participant.	The same table name was specified more than once in the **cdr define template** command, or a participant name in the **cdr realize template** command is not valid. **User action:** Rerun the command with unique table names or with valid participant names.
127	Template name already in use.	A replication template with this name exists. **User action:** Rerun the command with a unique template name.
128	Undefined template.	The template name specified in the command does not exist. **User action:** Rerun the command with an existing template name.
129	Cannot delete specified replset as it is a template.	The replicate set specified in the command is a part of a template and cannot be deleted with this command. **User action:** Run the **cdr delete template** command to delete a template.
131	Sync server not specified.	The synchronization server specified in the command must be the same server that was specified in the **cdr define repair** command. **User action:** Rerun the command with the correct synchronization server.

Return code	Error text	Explanation
132	Invalid sync server specified. Server is not yet defined in ER topology.	The synchronization server specified in the command is not a replication server. **User action:** Rerun the command with an existing replication server as the synchronization server.
134	Cannot lock the replicated table in exclusive mode. For more information see message log file.	The command cannot obtain an exclusive lock on the table to set alter mode. **User action:** See the online message log file for other errors.
135	Replicate/table is not in alter mode.	The table specified in the command is not in alter mode and therefore alter mode cannot be turned off.
136	Snoopy sub-component is down.	Alter mode cannot be set because the log capture thread was not active.
137	Mismatch between local table dictionary and master dictionary.	The master dictionary does not match the local participant dictionary. **User action:** Check the replicated table definitions on all participants.
138	Replicates not found for table. For more information see message log file.	Alter mode was not turned off because the replicate definitions for the specified table cannot be found. **User action:** Check the spelling of the table name and rerun the command.
139	Mismatch in replicate names or states. Primary and shadow replicate states must match. See the message log file for more information.	The primary and shadow replicates are not in the same state. **User action:** Run the **cdr list replicate** command to see the replicate state. When the primary and shadow replicates have the same state, rerun the original command.
140	Primary and shadow replicate participant verification failure.	The primary and shadow replicate information does not match.
141	Table is already in alter mode. For more information see message log file.	Alter mode cannot be turned on because the table is already in alter mode.
142	Classic replicates (no mastered dictionary) defined on the table. See message log file for more information.	One or more non-mastered replicates are defined on the specified table. Alter mode requires mastered replicates.
146	Resynchronize error, job name is already in use.	The job name must be unique. **User action:** Rerun the command with a unique job name.
147	Resynchronize error, specified replicate is a shadow repl.	The replicate specified in the command is a shadow replicate. The operation cannot be performed on a shadow replicate. **User action:** Run the **cdr list replicate** command to see a list of replicate names and rerun the original command with a primary replicate.
148	Only either participant list or target server can be given for a define repair command.	Both the target server name and a participant list were included in the command. **User action:** Rerun the command with either a target server name or a participant list.
151	Resynch job can be started or stopped only at the source server.	This command must be run from the server specified as the synchronization data source. **User action:** Rerun the command while connected to the synchronization data source server.
154	The replicate being repaired must be in active state.	The replicate specified in the command cannot be repaired because it is not active. **User action:** Run the **cdr list replicate** command to see the states of replicates.
156	Cannot perform auto remastering process. Replicate is not defined with column name verification option (–name=y). Perform manual remastering process.	Automatic remastering is not possible for the specified replicate. **User action:** Manually remaster the replicate. For instructions, see Remastering replicates without name verification.
157	CDR: Cannot verify/block grouper evaluation blocking condition.	The specified table cannot be set in alter mode because the grouper component is not active. **User action:** Run the **onstat -g grp** and **onstat -g ddr** commands to check the status of the grouper and log capture.
158	CDR: Cannot unblock grouper evaluation.	Alter mode cannot be turned off for the table because the grouper component is not active. **User action:** Run the **onstat -g grp** and **onstat -g ddr** commands to check the status of the grouper and log capture.
159	CDR: Grouper evaluation was already blocked in the same transaction. Commit the previous alter statement then re-execute the current alter statement.	More than one alter statement for replicated tables was included in a single transaction. **User action:** Rerun each alter statement in its own transaction.
160	The specified table was not found in the database. The table specified is either a view or an internally created cdr system table and replicate cannot be defined on views and internally created cdr system tables.	A table name specified in the command was not found or is not a type of table that can be replicated. **User action:** Rerun the command with valid table names.
161	Specified file to read table participants *filename* could not be opened. Please check. Template could not be defined.	The file name specified in the command does not exist. **User action:** If necessary, create the file. Rerun the command with the correct file path and name for the table list.
162	CDR: Local group name not defined in ATS/RIS file.	The ATS or RIS file content is not in the correct format. The file might be corrupted.
163	Error detected while checking replicate attributes on the given table.	The specified table cannot be set to alter mode because the table does not have any master replicates defined. Alter mode requires master replicates. **User action:** Run the **cdr list replicate** command to see the replicates that are defined for the table.
164	Cannot repair - ATS/RIS repair failed.	The ATS or RIS file content is not in the correct format. The file might be corrupted.

Return code	Error text	Explanation
165	Cannot suspend replicate/replset because of dependent repair jobs.	The replicate or replicate set cannot be suspended until the active repair jobs are complete. **User action:** Wait for the repair jobs to complete. Run the **cdr list replicate** command to see if the shadow replicates associated with the repair jobs still exist. After the shadow replicates are automatically deleted, rerun the original command.
166	Replicate set does not have any replicates.	The replicate set specified in the commands does not contain replicates. This error code is also returned when no replicates are found for a **cdr check** repair task when the **--allrepl** option is used. **User action:** Run the **cdr list replicateset** command for the replicate set to see its replicates.
167	Enterprise Replication is not supported in Express Edition server.	Enterprise Replication commands cannot be run on servers running Express Edition.
168	The specified table is actually a view, replicate definition on view is not supported.	Replicates cannot be defined on views. **User action:** Rerun the command specifying standard table names.
169	Cannot realize an empty template/	The template cannot be instantiated because it does not contain any replicates. **User action:** Run the **cdr list template** command to see if the template contains replicates.
170	Template is not yet defined that does not have any replicates.	The template cannot be instantiated because it is not defined. **User action:** Run the **cdr list template** command to see the names of defined templates.
171	Classic replicates do not support --verify (-v) and/or --autocreate (-u) options.	The **--verify** and **--autocreate** options are valid only for master replicates. **User action:** Verify the replicate definition by running the **cdr list replicate** command.
172	Checksum libraries not installed.	Enterprise Replication cannot find the checksum function for the **cdr check replicate** or **cdr check replicateset** command. This error can occur if a replication server is running a version of that does not support the **cdr check replicate** or **cdr check replicateset** command. This error can also occur if the custom checksum function that is specified by the **--checksum** option is not installed and registered on all replication servers. **User action:** If the replication server is running version 10.00, make sure that the checksum routines are registered. On version 10.00, checksum routines must be registered manually. If you specified a custom checksum function, make sure that it is installed and registered on all replication servers.
173	External Sync shutdown requested.	The synchronization task is not active. This error code is returned when Enterprise Replication is being shut down on a replication server participating in a synchronization task started by the **cdr sync** or **cdr check**command. **User action:** Run the **cdr list server** or **cdr view servers** command to see the status of the participating server and when all servers are active, rerun the original command.
174	External Sync abort required.	The synchronization or repair task did not complete in the timeout period. This error can occur if the replicate being synchronized or the shadow replicate that was created to resynchronize the data is not active at all the participants specified in the command. This error code is also returned when the **cdr check replicate** or **cdr check replicateset** command is run with the **--enable** option and the target server cannot be enabled and repaired in the timeout period. The timeout period is 128 seconds or the value you set with the **--timeout** option. **User action:** Run the **cdr list replicate** command to check the replicate status. If all participants are active, try running the command again. If the server was being enabled, run the **cdr list server** command to check the server status. If all participants are active, try running the command again with an increased timeout value.
175	Sync has received a request to stop.	The synchronization or check command was stopped.
176	Sync attempted on replicate which is not active.	The synchronization or check command was stopped because one of the replicates specified is not active.
178	WARNING: Replicate is not in sync.	The replicate is not in sync. This error can be returned after running **cdr check replicate** or **cdr check replicateset**. This error can be returned after running **cdr check replicate** or **cdr check replicateset** with the **--repair** option if there are pending transactions that are not yet applied or if transactions were aborted. **User action:** If you receive this error after running a consistency check, repair the data. For more information, see Resynchronizing data among replication servers. If you receive this error code after repairing data, look for ATS or RIS files at target participants. If you see ATS or RIS files, look at the SQL and ISAM error code for the failures and if necessary repair the transactions by using the **cdr repair** command. If there are no ATS or RIS files at the target participants, rerun the original command with the **--inprogress** option to control how long check task rechecks inconsistent rows that might be in process of being applied at target servers.
181	Value specified cannot be set in-memory, for more information see message log file.	The specified configuration parameter was not modified for the current session. **User action:** For more information, see the server online message log file.
182	Warning: Value specified was adjusted before setting it in-memory, for more information see message log file.	The value of the configuration parameter specified in the command was adjusted and then the configuration parameter was reset for the current session. **User action:** For more information, see the server online message log file.

Return code	Error text	Explanation
183	Operation not supported for the specified onconfig variable.	The specified configuration parameter cannot be dynamically updated while the server is running. **User action:** Edit the onconfig file and then shut down and restart the server.
184	onconfig text is specified in wrong format.	The value specified for the configuration parameter is not valid. **User action:** For more information, see the server online message log file.
185	Specified variable is an unsupported or unknown ER onconfig or CDR_ENV variable.	The specified configuration parameter or environment variable is not valid in this command. **User action:** Check the spelling of the configuration parameter or environment variable.
186	Value of onconfig variable cannot be changed when ER is defined.	The specified configuration parameter cannot be changed after Enterprise Replication is initialized. **User action:** For more information, see the server online message log file.
187	Value of onconfig variable cannot be changed when HDR is defined.	The specified configuration parameter cannot be changed while the server is participating in a high-availability cluster.
188	WARNING: The onconfig variable is not modified as the specified value is same as stored in the memory.	The value specified for the configuration parameter is the same as its current value for the session.
189	Replicate cannot be defined or modified since the participant table is protected using Label Based Access Control.	The table specified in the command is using label-based access control (LBAC), which is not supported with Enterprise Replication. **User action:** Rerun the command with a different table name, or remove LBAC from the table and then rerun the command.
190	Code sets specified by CLIENT_LOCALE and DB_LOCALE must be identical.	The ATS or RIS file repair operation requires that the **CLIENT_LOCALE** and **DB_LOCALE** environment variables be set to the same value. **User action:** Reset the value of one of the environment variables to that it matches the other and then rerun the original command.
191	Cannot determine connection server ID for server.	The command cannot obtain the group ID for the server being connected to.
192	Unable to find or connect to a **syscdr** database at a non-leaf server.	The repair command canot find a root server from which to obtain the Enterprise Replication catalog information.
193	SQL failure due to server resource limitations.	An SQL statement failed with memory or lock resource-related error codes.
194	SQL failure due to loss of network connection to server.	An SQL query failed with a network error.
195	SQL failure.	This error code is returned when a command fails due to an SQL error code other than SQL resource limitations-related error codes.
196	Encountered an SQL error.	The command was stopped because an SQL statement failed.
200	Unexpected Internal Error with cdr check or cdr sync.	An internal UDR routine execution might have returned an unexpected error. **User action:** Look at the additional error messages printed on the screen to get more details about this error.
201	Sync/Check encountered an unexpected column type.	The data type of one of the columns being synchronized or checked cannot be resolved for data comparison.
202	Source and Target do not have the same data type.	Corresponding columns on the source server and the target server have different data types.
203	Data for row or column not found.	Enterprise Replication cannot display the column value of a mismatched column on the screen.
204	Table could not be found.	The table that is being synchronized or repaired is not found on one of the participants. This error code is also returned if the participant being deleted cannot be found in the Enterprise Replication catalog tables.
205	Undefined server group.	The server group specified in the command was not found in the Enterprise Replication catalog tables. **User action:** Rerun the command with an existing server group name.
206	Template not realized at sync data source.	The template cannot be realized on the specified servers because it is not yet realized on the synchronization server. **User action:** Rerun the **cdr realize template** command specifying the synchronization source server as a participant.
207	Template already realized at one or more of requested servers.	One or more of the participants specified in the command already has the template instantiated on it. **User action:** Run the **cdr list replicate** command to check the status of the participants and then rerun the original command with the correct list of participants.
208	Server unknown at remote server.	Information about the local server is not available at the remote server.
209	A byte sequence that is not a valid character in the specified locale was encountered	One or more characters in a name specified in the command is not valid.
210	Parameter passed to command (or internally, routine) is invalid.	An argument specified in the command does not have a valid value.
211	Command is too large to be executed as a background task.	The command specified as a background task exceeded 2048 bytes. **User action:** Rerun the command without the **--background** option.
212	Sync/Check subtask aborted.	One of the tasks that was being performed in parallel was stopped. **User action:** Check the command output to determine which task was stopped.

Return code	Error text	Explanation
213	WARNING: set is not in sync.	At least one of the replicates in the specified replicate set is not in sync. This error can be returned after running **cdr check replicateset**. This error can be returned after running **cdr check replicateset** with the **--repair** option if there are pending transactions that are not yet applied or if transactions were aborted. **User action:** If you receive this error after running a consistency check, repair the data. For more information, see Resynchronizing data among replication servers. If you receive this error code after repairing data, look for ATS or RIS files at target participants. If you see ATS or RIS files, look at the SQL and ISAM error code for the failures and if necessary repair the transactions by using the **cdr repair** command. If there are no ATS or RIS files at the target participants, rerun the original command with the **--inprogress** option to control how long the check task rechecks inconsistent rows that might be in process of being applied at target servers.
214	ER: The logical log replay position is not valid. Restart ER with the **cdr cleanstart** command, and then synchronize the data with the **cdr check --repair** command.	Enterprise Replication cannot start because the logical log replay position is not valid. **User action:** Run the **cdr cleanstart** command and then the **cdr check replicateset** command with the **--repair** option.
215	Command failed -- The specified table is an external table. You cannot include an external table in a replicate.	Tables created with the CREATE EXTERNAL TABLE statement cannot be included in a replicate.
217	Error with Quality of Data command.	This error can be returned after running the **cdr define qod** command if the quality of data master server is already defined. This error can be returned after running the **cdr start qod** command or the **cdr stop qod** command if the quality of data master server is not defined or the command was run from a different server. **User action:** If the quality of data master does not exist, run the **cdr define qod** command and then rerun the original command. If the command was run on a different server, rerun the command from the quality of data master server, as indicated in the error message.
220	A node included in the list is not valid.	The server group specified in the grid command was not found in the Enterprise Replication catalog tables. **User action:** Rerun the command with an existing server group name.
221	The grid name is not unique.	Grid names must be unique among grids and among replicate sets. **User action:** Run the **cdr list grid** command to see existing grid names and run the **cdr list replicateset** command to see existing replicate set names. Rerun the original command with a unique grid name.
222	The grid does not exist.	The grid name specified in the command is not the name of an existing grid. **User action:** Run the **cdr list grid** command to see existing grid names. Rerun the original command with an existing grid name.
223	grid enable user failed	The user name specified in the command does not exist. **User action:** Rerun the command with an existing user name.
224	grid enable node failed	The server name specified in the command is not the name of an existing replication server. **User action:** Rerun the command with an existing replication server name.
225	sec2er failure	The **cdr start sec2er** command failed. **User action:** Following the instructions in the command output to perform all necessary prerequisites.
227	Region Command Failed	The **cdr define region** or **cdr delete region** command failed. **User action:** Review the specific error message and make the appropriate corrections to the command.
228	Grid Table Command Failed.	The **cdr change gridtable** command failed. **User action:** Review the specific error message and make the appropriate corrections to the command.

Related concepts:
Command Abbreviations
Option Abbreviations
Option Order
Long Command-Line Examples
Long Identifiers
Connect Option
Frequency Options
Related reference:
Participant and participant modifier
cdr finderr
cdr check queue

Frequency Options

You can specify the interval between replications or the time of day when replication occurs for a replicate.

```
Frequency Options
```

```
   .- --immed----------.
|--+-------------------+---------------------------------------|
   +- --every=interval-+
   '- --at=time--------'
```

Element	Purpose	Restrictions
interval	Time interval for replication	The smallest interval in minutes, in one of the following formats: • The number of minutes, as an integer value 1 - 1966020, inclusive. • The number of hours and minutes separated by a colon. The minimum value is 0:01. The maximum value is 32767:59
time	Specific time for replication	Time is given as a 24-hour clock.

The following table describes the frequency options.

Long Form	Short Form	Meaning
--immed	**-i**	Default. Replication occurs immediately.
--every=	**-e**	Replication occurs immediately and repeats at the frequency that is specified by interval.
--at=	**-a**	Replication occurs at the specified day and time.

Usage

The frequency of replication is a property of a replicate. You can set the frequency of replication for a replicate when you define it or modify it. You can reset the frequency of all replicates in a replicate set when you define or modify a replicate set or define a template. For non-exclusive replicate sets, you can update the frequency of individual replicates separately.

If you do not specify a time, replication occurs immediately.

Important: When you use time-based replication by including the **--every** or the **--at** option, replicated transactions are split into multiple transactions and referential integrity is not supported. If you want to replicate data intermittently, you can specify the **--immed** and then disconnect the servers until you want to replicate the data.

Intervals

The **--every=***interval* option specifies the interval between actions. The interval of time between replications can be either of the following formats:

- The number of minutes
  To specify the number of minutes, specify an integer value greater than 0. For example, `-e 60` indicates 60 minutes between replications.

  If you specify the interval of time between replications in minutes, the longest interval is 1966020.

- The number of hours and minutes
  To specify hours and minutes, give the number of hours, followed by a colon, and then the number of minutes. For example, `-e 5:45` indicates 5 hours and 45 minutes between replications.

If you specify the length of time in hours and minutes, the longest interval is 32767:59.

Time of Day

Enterprise Replication always gives the time of day in 24-hour time. For example, 19:30 is 7:30 P.M. Enterprise Replication always gives time as the local time, unless the **TZ** environment variable is set. However, Enterprise Replication stores times in the global catalog in Greenwich Mean Time (GMT).

The **--at=***time* option specifies the day on which replication occurs. The string *time* can have the following formats:

- Day of week
  To specify a specific day of the week, give either the long or short form of the day, followed by a period and then the time. For example, `--at=Sunday.18:40` or `-a Sun.18:40` specifies that replication occurs every Sunday at 6:40 P.M.

  The day of the week is given in the locale of the client. For example, with a French locale, you might have `--at=Lundi.3:30` or `-a Lun.3:30`. The time and day are in the time zone of the server.

- Day of month
  To specify a specific day in the month, give the date, followed by a period, and then the time. For example, `1.3:00` specifies that replication occurs at 3:00 A.M. on the first day of every month.

  The special character `L` represents the last day of the month. For example, `L.17:00` is 5:00 P.M. on the last day of the month.

- Daily
  To specify that replication occurs each day, give only the time. For example, 4:40 specifies that replication occurs every day at 4:40 A.M.

Related concepts:
Command Abbreviations
Option Abbreviations
Option Order
Long Command-Line Examples
Long Identifiers
Connect Option
Return Codes for the cdr Utility

Related reference:
Participant and participant modifier
cdr change replicateset
cdr define replicate
cdr define replicateset
cdr define template
cdr modify replicate
cdr modify replicateset

cdr add onconfig

The **cdr add onconfig** command adds one or more values to a configuration parameter in the ONCONFIG file.

Syntax

```
>>-cdr add onconfig--+-------------------------+--------------->
                     |                    (1)  |
                     '-| Connect Option   |-----'

>--"--parameter name--value--"--------------------------------><
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter to set.	You can add values to the following Enterprise Replication configuration parameters: • CDR_LOG_LAG_ACTION • CDR_LOG_STAGING_MAXSIZE • CDR_QDATA_SBSPACE • CDR_SUPPRESS_ATSRISWARN • ENCRYPT_MAC • ENCRYPT_MACFILE • CDR_ENV: ◦ CDRSITES_731 ◦ CDRSITES_92X ◦ CDRSITES_10X	
value	The value of the configuration parameter.	Must be a valid value for the configuration parameter.	Follows the syntax rules for the specific configuration parameter.

Usage

Use the **cdr add onconfig** command to add one or more values to an Enterprise Replication configuration parameter while replication is active. The ONCONFIG file is updated. You can set environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example adds an sbspace to the existing list of sbspaces for holding spooled transaction row data:

```
cdr add onconfig "CDR_QDATA_SBSPACE sbspace_11"
```

The following example adds the cdrIDs for two version 7.x servers to the existing list of servers:

```
cdr add onconfig "CDR_ENV CDRSITES_731=1,3"
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Dynamically Modifying Configuration Parameters for a Replication Server
Related reference:
cdr change onconfig
cdr remove onconfig

cdr alter

The **cdr alter** command puts the specified tables in alter mode.

Syntax

```
>>-cdr alter--+--------------------------+--+- --on--+---------->
              |                     (1) |  '- --off-'
              '-| Connect Option   |-----'

   .----------------------.
   V                      |
>----database:owner.table-+-------------------------------------><
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
database	The name of the database that contains the table	The database server must be registered with Enterprise Replications.	Long Identifiers
owner	User ID of the owner of the table		Long Identifiers
table	Specifies the name of the table to put in alter mode	The table must be an actual table. It cannot be a synonym or a view.	Long Identifiers

The following table describes the options to **cdr alter**.

Long Form	Short Form	Meaning
--on	**-o**	Sets alter mode on.
--off	**-f**	Unsets alter mode.

Usage

Use this command to place a table in or out of alter mode. Use alter mode when you need to alter an attached fragment to the table or you want to perform other alter operations manually.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example puts **table1** and **table2** in alter mode:

```
cdr alter --on db1:owner1.table1 db2:owner2.table2
```

Related concepts:
Alter, rename, or truncate operations during replication
SQL statements and replication
Enterprise Replication Server administrator
Related reference:
cdr swap shadow
cdr remaster

cdr autoconfig serv

The **cdr autoconfig serv** command can autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can automatically configure replication.

Syntax

```
>>-cdr autoconfig serv--+--------------------------+------------>
                        |                     (1) |
                        '-| Connect Option   |-----'

>--+-----------------------------------------------+--------------><
   '-+-| Source options |---------------------+-'
     +-| Target options |---------------------+
     '-| Source options |--| Target options |-'

Source options

|-- --sourcehost--host-- --sourceport--port---------------------|

Target options
```

```
|-- --targethost--host-- --targetport--port--------------------|
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions
host	The name of a database server host.	
port	The port number that is used for communication	

The following table describes the options to **cdr autoconfig serv**.

Long Form	Short Form	Meaning
--sourcehost	**-H**	The host of the database server that is sending autoconfiguration information. If **--sourcehost** and **--sourceport** are not specified, the database server where the command is run is considered the source database server.
--sourceport	**-P**	The port that is used by the database server that is sending autoconfiguration information.
--targethost	**-h**	The host of the database server that is receiving autoconfiguration information.
--targetport	**-p**	The port that is used by the database server that is receiving autoconfiguration information.

Usage

Run the **cdr autoconfig serv** command to autoconfigure connectivity for servers in a high-availability cluster or Enterprise Replication domain, and to autoconfigure replication if you are adding database servers to an Enterprise Replication domain. The CDR_AUTO_DISCOVER configuration parameter must be set to `1` on all database servers that are participating in an Enterprise Replication domain or high-availability cluster, before you can run the **cdr autoconfig serv** command. A newly installed database severs that is added to an Enterprise Replication domain through the **cdr autoconfig serv** command must have a configured storage pool.

If the source server is already configured for Enterprise Replication, the command performs the following actions:

1. The source server propagates its trusted-host file to target server.
2. The target server adds entries for itself and all other replication servers to its sqlhosts file.
3. The source server updates its sqlhost file with entries for the target server.
4. Each replication server updates its sqlhost file and trusted-host file with entries for the target server.
5. The target server sets its CDR_DBSPACE configuration parameter and creates the dbspace that is required for Enterprise Replication.
6. The target server sets its CDR_QDATA_SBSPACE configuration parameter and creates the sbspace that is required for Enterprise Replication.
7. The aborted transactions spooling (ATS) file directory $INFORMIXDIR/tmp/ats_*dbservername* is created on the target server.
8. The row information spooling (RIS) file directory $INFORMIXDIR/tmp/ris_*dbservername* is created on the target server.
9. Replication of the domain information in the **syscdr** catalog to the target server starts.

If the source server is not configured for Enterprise Replication, the command performs the additional actions:

1. The source server adds entries for itself to its sqlhosts file.
2. The source server sets its CDR_DBSPACE configuration parameter and creates the dbspace that is required for Enterprise Replication.
3. The source server sets its CDR_QDATA_SBSPACE configuration parameter and creates the sbspace that is required for Enterprise Replication.
4. The aborted transactions spooling (ATS) file directory $INFORMIXDIR/tmp/ats_*dbservername* is created on the source server.
5. The row information spooling (RIS) file directory $INFORMIXDIR/tmp/ris_*dbservername* is created on the source server.

The following restrictions apply to the **cdr autoconfig serv** command:

- All replication servers must be active, or the **cdr autoconfig serv** command fails.
- Do not run the **cdr autoconfig serv** command if you have configured trusted-host information, manually, rather than through running the **admin()** or **task()** function with the **cdr add trustedhost** argument.
- Do not run the **cdr autoconfig serv** command if your replication servers have secure ports configured.
- The **cdr autoconfig serv** command does not copy hosts.equiv information to the trusted-host file that is set by the REMOTE_SERVER_CFG configuration parameter. Run the **admin()** or **task()** function with the **cdr add trustedhost** argument if you must add information from the hosts.equiv file to the trusted-host file that is set by the REMOTE_SERVER_CFG configuration parameter.

Database servers are configured serially. Parallel configuration is not supported.

You can run this command from within an SQL statement by using the SQL administration API.

Example 1: Define Enterprise Replication on a database server

For this example, you have one database server that is not configured for Enterprise Replication:

- **server_1** on **host_1**

The following command is run on **server_1**.

```
cdr autoconfig serv
```

The command defines Enterprise Replication on **server_1**.

Example 2: Configure connectivity and ER between two stand-alone servers by using source syntax

For this example, you have two stand-alone database servers:

- **server_1** on **host_1**
- **server_2** on **host_2**

The following command is run on **server_1**:

```
cdr autoconfig serv -c server_2 --sourcehost host_1 --sourceport 9000
```

The command performs the following actions:

1. The command connects to **server_2**.
2. Enterprise Replication is defined on **server_1**.
3. Enterprise Replication is defined on **server_2**.
4. **server_1** replicates the domain data to **server_2**.

Example 3: Configure connectivity and ER between two stand-alone servers using target syntax

For this example, you have two stand-alone database servers:

- **server_1** on **host_1**
- **server_2** on **host_2**

The following command is run on **server_2**:

```
cdr autoconfig serv -c server_1 --targethost host_2 –targetport 9002
```

The command performs the following actions:

1. The command connects to **server_1**.
2. Enterprise Replication is defined on **server_1**.
3. Enterprise Replication is defined on **server_2**.
4. **server_1** replicates the domain data to **server_2**.

Example 4: Configure connectivity and ER between two stand-alone servers

For this example, you have three stand-alone database servers:

- **server_1** on **host_1**
- **server_2** on **host_2**
- **server_3** on **host_3**

The following commands are run on **server_1**:

```
cdr autoconfig serv --targethost hos_t2 –targetport 9002
cdr autoconfig serv --targethost host_3 –targetport 9003
```

The commands perform the following actions:

1. The first command connects to **server_1**.
2. Enterprise Replication is defined on **server_1**.
3. Enterprise Replication is defined on **server_2**.
4. **server_1** replicates its data to **server_2**.
5. The second command connects to **server_1**.
6. Enterprise Replication is defined on **server_3**.
7. **server_1** replicates the domain data to **server_3**.

Related concepts:
Creating sqlhost group entries for replication servers
Related tasks:
Preparing the Network Environment
Related reference:
cdr autoconfig serv
CDR_AUTO_DISCOVER configuration parameter
Related information:
cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API)
cdr add trustedhost argument: Add trusted hosts (SQL administration API)
Trusted-host information
Client/server communication

cdr change grid

The **cdr change grid** command adds or deletes replication servers to or from a grid.

Syntax

```
>>-cdr change grid--+-------------------------+--grid_name----->
                   |                    (1)  |
                   '-| Connect Option   |-----'

                  .--------------.
                  V              |
>--+- --add----+----server_group-+------------------------------><
   '- --delete-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers
server_group	Name of a database server group to add to, or remove from, the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers

The following table describes the **cdr change grid** options.

Long Form	Short Form	Meaning
--add	**-a**	Add the specified replication servers to the grid.
--delete	**-d**	Delete the specified replication servers from the grid.

Usage

Use the **cdr change grid** command to add a new replication server to an existing grid or to remove a replication server from an existing grid.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility.

Examples

The following example adds two servers to a grid named **grid_1**:

```
cdr change grid grid_1 --add gserv3 gserv4
```

The following example removes a server from a grid named **grid_1**:

```
cdr change grid grid_1 --delete gserv1
```

Related concepts:
Grid maintenance
Enterprise Replication Server administrator
Related tasks:
Adding a replication server to a grid by cloning
Adding a replication server to a grid by running cdr change grid
Related reference:
cdr define grid
cdr list grid

cdr change gridtable

The **cdr change gridtable** command has multiple uses. It can verify that specific tables can be used in gird queries and then add the tables to a list of verified tables, and it can delete tables from the list of verified tables.

Syntax

```
>>-cdr change gridtable--+-------------------------+----------->
                         |                    (1)  |
                         '-| Connect Option   |-----'

>-- --grid--=--grid_name-- --database--=--database-------------->

>--+- --add----+--+- --all----+--------------------------------><
   '- --delete-'  | .-------. |
                  | V       | |
                  '---table-+-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
database	The name of the database.	Must be in a server that is in the specified grid.	Long Identifiers
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers
table	The name of the table.	The table cannot be a synonym or a view.	Long Identifiers

The following table describes the **cdr change gridtable** options.

Long Form	Short Form	Meaning
--add	**-a**	Add the specified tables to the grid.
--all	**-A**	Specifies to add or delete all the tables in the database.
--database=	**-D**	Specifies the database in which the tables are located.
--delete	**-d**	Delete the specified tables from the grid.
--grid=	**-g**	Specifies the grid to which to add or delete tables.

Usage

Use the **cdr change gridtable** command with the **--add** option to add one or more tables to an existing grid. You add a table to a grid when you want to include the table in grid queries. System tables are automatically included in the grid. When you add a table to a grid, the **cdr change gridtable** command ensures that every table with that name has the same schema on every grid server. Every table must have the same columns, column names, and data types. The specified database must use the same locale on every grid server.

Use the **cdr change gridtable** command with the **--delete** option to remove one or more tables from an existing grid. The specified tables cannot be included in grid queries.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 228.

For information about these error codes, see Return Codes for the cdr Utility.

Examples

The following example adds all tables in the **stores** database to the grid named **grid1**:

```
cdr change gridtable --grid=grid1 --database=stores --add --all
```

The following example removes the table named **customer** in the **stores** database from the grid named **grid1**:

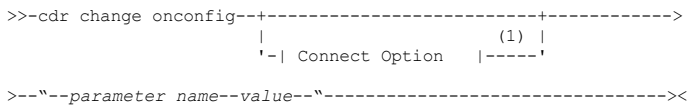
```
cdr change gridtable --grid=grid1 --database=stores --delete=customer
```

Related concepts:
Grid queries
Related tasks:
Defining tables for grid queries
Related reference:
cdr remaster gridtable
Related information:
GRID clause

cdr change onconfig

The **cdr change onconfig** command replaces the existing value of a configuration parameter with a new value in the ONCONFIG file.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter to change.	None. All Enterprise Replication configuration parameters and environment variables can be changed with this command.	
value	The value of the configuration parameter.	Must be a valid value for the configuration parameter.	Follows syntax rules for the specific configuration parameter.

Usage

Use the **cdr change onconfig** command to replace the existing value of an Enterprise Replication configuration parameter with a new value in the ONCONFIG file. You can set Enterprise Replication environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

Suppose the CDR_SUPPRESS_ATSRISWARN configuration parameter is set to suppress the generation of error and warning messages 1, 2, and 10, so that it appears in the ONCONFIG file as: CDR_SUPPRESS_ATSRISWARN 1,2,10. The following command changes the suppressed error and warning messages to 2, 3, 4, 5, and 7:

```
cdr change onconfig "CDR_SUPPRESS_ATSRISWARN 2-5,7"
```

Suppose the CDR_RMSCALEFACT environment variable is set to the value of 4. The following example sets the number of data sync threads started for each CPU VP to 3:

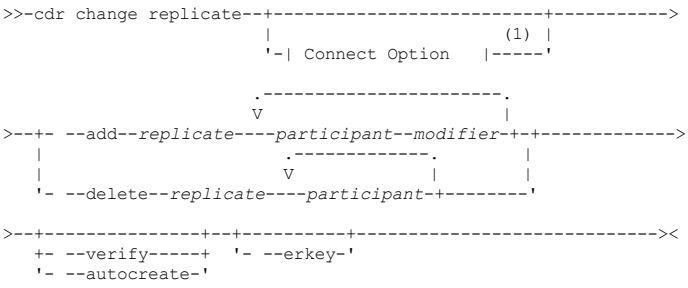
```
cdr change onconfig "CDR_ENV CDR_RMSCALEFACT=3"
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Dynamically Modifying Configuration Parameters for a Replication Server
Related reference:
cdr add onconfig
cdr remove onconfig

cdr change replicate

The **cdr change replicate** command modifies an existing replicate by adding or deleting one or more participants.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		Participant and participant modifier
participant	Specifies the database server and table for replication.	The participant must exist.	Participant and participant modifier
replicate	Name of the replicate to change.	The replicate must exist.	Long Identifiers

The following table describes the options to **cdr change replicate**.

Long Form	Short Form	Meaning
--add	**-a**	Adds participants to a replicate.

Long Form	Short Form	Meaning
--autocreate	**-u**	For use with master replicates only. Specifies that if the tables in the master replicate definition do not exist in the databases on the target servers, then they are created automatically. However, the table cannot contain columns with user-defined data types. The tables are created in the same dbspace as the database. Note: Tables that are created with the **--autocreate** option do not automatically include non-replication key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with the **--autocreate** option require the use of these objects you must explicitly create the objects by hand.
--delete	**-d**	Removes participants from a replicate.
--erkey	**-K**	Includes the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, in the replicate definition, if the table being replicated has the ERKEY shadow columns. The ERKEY shadow columns are used as the replication key.
--verify	**-v**	For use with master replicates only. Specifies that the **cdr change template** command verifies the database, tables, and column data types against the master replicate definition on all listed servers

Usage

Use this command to add or delete a participant from a replicate. You can define a replicate that has zero or one participants, but to be useful, a replicate must have at least two participants. You cannot start and stop replicates that have no participants. All participants for the replicate must be online and the **cdr** utility must be able to connect to each participant.

Important: Enterprise Replication adds the participant to the replicate in an inactive state, regardless of the state of the replicate. To activate the new participant, run **cdr start replicate** with the name of the server group. See cdr start replicate.
When you run the **cdr change replicate** command, an event alarm with a class ID of 65 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example 1: Add two participants

The following example adds two participants to the replicate named **repl_1**: **db1@server1:antonio.table** with the modifier `select * from table1`, and **db2@server2:carlo.table2** with the modifier `select * from table2`:

```
cdr change repl -a repl_1 \
   "db1@server1:antonio.table1" "select * from table1" \
   "db2@server2:carlo.table2" "select * from table2"
```

Example 2: Remove two participants

The following example removes the same two participants from replicate **repl_1**:

```
cdr change repl -d repl_1 \
   "db1@server1:antonio.table1" \
   "db2@server2:carlo.table2"
```

Example 3: Add a participant that includes ERKEY shadow columns

The following example adds a participant and includes the ERKEY shadow columns from the table **table1**:

```
cdr change repl -a repl_1 --erkey\
   "db1@server1:antonio.table1" "select * from table1"
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Preparing tables without primary keys
Related reference:
cdr define replicate
cdr delete replicate
cdr list replicate
cdr modify replicate
cdr resume replicate
cdr start replicate
cdr stop replicate
cdr suspend replicate
Enterprise Replication Event Alarms
Participant and participant modifier

cdr change replicateset

The **cdr change replicateset** command changes a replicate set by adding or deleting replicates.

Syntax

```
>>-cdr change replicateset--+-------------------------+-------->
                            |                    (1)  |
                            '-| Connect Option   |-----'

                            .-----------.
                            V           |
>--+- --add----+--repl_set----replicate-+---------------------><
   '- --delete-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set to change.	The replicate set must exist.	Long Identifiers
replicate	Name of the replicates to add to or delete from the set.	The replicates must exist.	Long Identifiers

The following table describes the options to **cdr change replicateset**

Long Form	Short Form	Meaning
--add	**-a**	Add replicates to a replicate set.
--delete	**-d**	Remove replicates from a replicate set.

Usage

Use this command to add replicates to a replicate set or to remove replicates from an exclusive or non-exclusive replicate set:

- If you add a replicate to an exclusive replicate set, Enterprise Replication changes the existing state and replication frequency settings of the replicate to the current properties of the exclusive replicate set.
  If you remove a replicate from an exclusive replicate set, the replicate retains the properties of the replicate set at the time of removal (not the state the replicate was in when it joined the exclusive replicate set).

  When you add or remove a replicate from an exclusive replicate set that is suspended or that is defined with a frequency interval, Enterprise Replication transmits all the data in the queue for the replicates in the replicate set up to the point when you added or removed the replicate.
- If you add or remove a replicate to a non-exclusive replicate set, the replicate retains its individual state and replication frequency settings.

Use this command to add or remove replicates from a grid replicate set. You can only add replicates that were created outside of a grid environment to a grid replicate set if the following conditions are met:

- The participant servers must be the same as the servers in the grid.
- The replicated table schema must be the same among all participants.
- The entire replicated table is replicated. Using a SELECT statement in the participant definition that does not include all the columns in the table or includes a WHERE clause is not allowed.
- The replicate must not belong to an exclusive replicate set.
- The replicate must not include **TimeSeries** columns.

When you run the **cdr change replicateset** command, an event alarm with a class ID of 66 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example adds the replicates **house** and **barn** to replicate set **building_set**:

```
cdr change replicateset --add building_set house barn
```

The following example removes the replicates **teepee** and **wigwam** from replicate set **favorite_set**:

```
cdr change replset --delete favorite_set teepee wigwam
```

Related concepts:
Frequency Options
Enterprise Replication Server administrator
Related tasks:
Suspending a Replicate Set
Adding an existing replicate to a grid replicate set by using cdr change replicateset
Related reference:
cdr define replicate
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset

cdr change shardCollection

The **cdr change shardCollection** command changes the sharding definition that determines which database servers are part of the shard cluster.

Syntax

Notes:

1. See Connect Option.

Element	Description	Restrictions
ER_group	The ER-group name of a database server that receives sharded data.	Must be the ER-group name of an existing database server.
definition_name	The name of the sharding definition that is modified.	Must be the name of an existing definition.
expression	The WHERE-clause expression that is used to select rows or documents by shard key or shard column value.	
partitions	The number of hashing partitions on each shard server.	Must be a positive integer.
REMAINDER	Specifies the database server that receives rows or documents with shard key values that are not selected by other expressions.	Use only if the sharding definition uses an expression for distributing data.

The following table describes the **cdr change shardCollection** options.

Long Form	Short Form	Description
--add	**-a**	Adds a database server to a sharding definition.
--drop	**-d**	Removes a database server from a sharding definition. If the database server that is removed uses an expression for sharding, the expression is not included in the **cdr change shardCollection** command.
--partitions=	**-p**	Changes the number of hashing partitions on each shard server when the sharding strategy is consistent hashing. The more hashing partitions, the more evenly the data is distributed among shard servers. However, if you specify more than 10 hashing partitions, the resulting SQL statement to create the sharded table might fail because it exceeds the maximum character limit for an SQL statement.
--replace	**-r**	Replaces a sharding definition when the sharding strategy is expression.

Usage

The **cdr change shardCollection** command can change the database servers that are part of a shard cluster. You cannot use the **cdr change shardCollection** command to change the sharding strategy of a shard cluster. To change the sharding strategy, you must remove the shard cluster and then re-create it with a different sharding strategy. However, you can change certain properties of the following sharding strategies:

- Consistent hash: You can change the number of hashing partitions on each shard server.
- Expression: You can change the expression that is used for distributing data.

The **cdr change shardCollection** command creates a new sharding definition, moves existing data to appropriate shard servers, and then removes the original sharding definition.

To delete a shard server from Enterprise Replication, remove the database server from its shard cluster by running the **cdr change shardCollection** command and then run the **cdr delete server** command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 39, 99, 196, 229.

For information about these error codes, see Return Codes for the cdr Utility.

Example: Adding a database server to a sharding definition that uses a hash algorithm

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers_1
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
```

The following command adds **g_shard_server_D** to **collection_1**:

```
cdr change shardCollection collection_1 --add g_shard_server_D
```

A new sharding definition is created for **collection_1**. All data is redistributed to the database servers in the new sharding definition. The old sharding definition is deleted.

Example: Changing the number of hashing partitions on shard servers

For this example, you have a sharding definition with a consistent hashing strategy that is created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1
   g_shard_server_2
   g_shard_server_3
```

The following command changes the number of hashing partitions on each shard server to 4:

```
cdr change shardCollection collection_1 --partitions=4
```

The data on each shard server is redistributed into four partitions. A small amount of data might be moved between shard servers.

Example: Adding multiple database servers to a sharding definition that uses an expression

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_2 db_2:joe.clients
   --type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_A "IN ('TX','OK')"
   g_shard_server_B "IN ('NY','NJ')"
   g_shard_server_C "IN ('AL','GA')"
   g_shard_server_D  REMAINDER
```

The following command adds the shard servers **g_shard_server_E** and **g_shard_server_F** to **collection_2**:

```
cdr change shardCollection collection_2 --add
   g_shard_server_E "IN ('CA','AZ')"
   g_shard_server_F "IN ('WA','ID')"
```

A new sharding definition is created for **collection_2**. The appropriate data from **g_shard_server_D** is redistributed to the new shard servers. The old sharding definition is deleted.

Example: Removing a database server from a sharding definition

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_3 db_3:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
   g_shard_server_D
```

The following command removes **g_shard_server_B** from **collection_3**:

```
cdr change shardCollection collection_3 --drop g_shard_server_B
```

A new sharding definition is created for **collection_3**. The data is removed from **g_shard_server_B**. All the data is redistributed to the database servers in the new sharding definition. The old sharding definition is deleted.

Example: Replacing shard servers

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_4 db_4:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A
   g_shard_server_B
   g_shard_server_C
   g_shard_server_D
```

The following command changes the shard servers for **collection_4** to **g_shard_server_A**, **g_shard_server_C**, **g_shard_server_E**, and **g_shard_server_F**:

```
cdr change shardCollection collection_4 --replace
   g_shard_server_A
   g_shard_server_C
   g_shard_server_E
   g_shard_server_F
```

A new sharding definition is created for **collection_4**. The data is removed from **g_shard_server_B** and **g_shard_server_D**. All the data is redistributed to the database servers in the new sharding definition. The old sharding definition is deleted.

Example: Replacing a sharding definition that uses an expression

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_5 db_5:joe.clients
   -t delete -k unit_number -s expression -v version
   g_shard_server_A "BETWEEN 0 and 100"
   g_shard_server_B "BETWEEN 101 and 200"
   g_shard_server_C "BETWEEN 201 and 300"
   g_shard_server_D  REMAINDER
```

The following command changes the shard servers and the expression for **collection_5**:

```
cdr change shardCollection collection_5 -r
   g_shard_server_E "BETWEEN 0 and 100"
   g_shard_server_F "BETWEEN 101 and 200"
   g_shard_server_G "BETWEEN 201 and 200"
   g_shard_server_C "BETWEEN 301 and 400"
   g_shard_server_D  REMAINDER
```

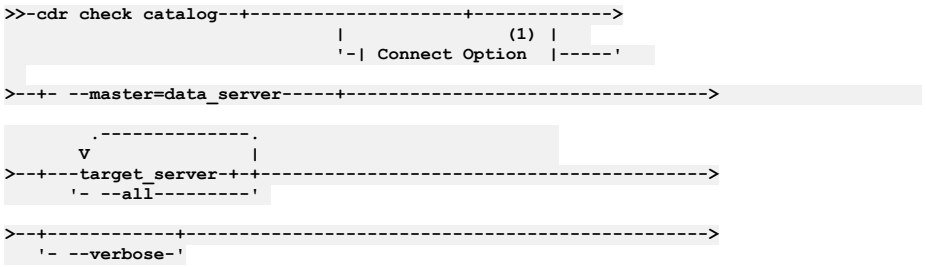
A new sharding definition is created for **collection_5**. The data is removed from **g_shard_server_A** and **g_shard_server_B**. All the data is redistributed to the appropriate database servers in the new sharding definition. The old sharding definition is deleted.

Related concepts:
Shard cluster management and monitoring
Related reference:
cdr define shardCollection
cdr delete shardCollection
cdr list shardCollection
Related information:
Enabling sharding for JSON or relational data
Changing the definition for a shard cluster
onstat -g shard command: Print information about the shard cache

cdr check catalog

The **cdr check catalog** command compares the metadata information related to servers, replicates and replicate sets on replication servers for any inconsistency.

Syntax



Note:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
target_server	Name of a database server group to check.	Must be the name of an existing database server group in the sqlhosts file.	Long identifiers

The following table describes the options to **cdr check catalog**.

Long Form	Short Form	Meaning

Long Form	Short Form	Meaning
--all	**-a**	Specifies that master server metadata info is compared to metadata info on all servers in ER domain.
--master=	**-m**	Specifies the database server to use as the reference copy of the data.
--verbose	**-v**	Specifies that the consistency report shows all comparisons.

Usage

Use **cdr check catalog** command to compare the metadata information related to servers, replicates and replicate sets on replication servers for any inconsistency. If you include the **--verbose** option, the report lists every comparison for metadata between the master server and target servers. If the servers specified for **--connect** or **--master** options are leaf servers, parent servers are used instead.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful. If the command is not successful, one of the following error codes is returned: 1, 5, 21, 37, 48, 53, 61, 62, 99, 121, 193, 194, 195, 205. For information about these error codes, see Return Codes for the cdr Utility.

Examples

The following command generates a consistency report comparing the metadata on the master server **g_serv1** with the metadata on the server **g_serv2**:

```
cdr check catalog --master=g_serv1 g_serv2
```

The summary consistency report shows that the metadata is consistent:

```
Verifying server definitions...
Server definitions...OK

Verifying replicate definitions...
Replicate definitions...OK

Verifying replicate participant definitions...
Replicate participant definitions...OK

Verifying replicate participants...
Replicate participants...OK

Verifying replicate set definitions...
Replicate set definitions...OK

Verifying replicate set participants...
Replicate set participants...OK
```

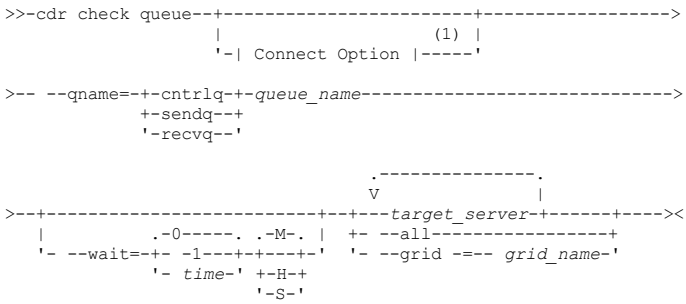
This report indicates that the metadata is consistent on these servers.

Related reference:
cdr check replicate
cdr check replicateset

cdr check queue

Use the **cdr check queue** command to check the consistency of Enterprise Replication metadata, and to check the consistency of user data before running critical tasks in the Enterprise Replication domain. The command returns successfully when all of the commands that were queued when **cdr check queue** was run are complete.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid	Must be the name of an existing grid.	Long Identifiers
target_server	Name of a database server group on which to check the queue		

The following table describes the **cdr check queue** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the Enterprise Replication are checked
--grid	**-g**	Specifies the grid name
--qname	**-q**	Specifies the name of the queue to monitor: • **cntrlq** = Control queue • **sendq** = Send queue • **recvq** = Receive queue
--wait	**-w**	Specifies the amount of time to wait for queues to complete before returning. Minutes are used if the time unit is not specified. **-1** = Wait until all queued elements are complete **0** (default) = Do not wait for queued elements to complete; return immediately *Positive integer* = Number of hours, minutes, or seconds to wait, depending on the time unit specified: • **H** or **h** = Hours • **M** or **m** = Minutes (default) • **S** or **s** = Seconds

Usage

The **cdr check queue** command is used to monitor control, send, and receive queues on one or more Enterprise Replication servers and can optionally wait for queues to empty before returning.

The Enterprise Replication queues are checked at the time that the **cdr check queue** command runs. The time is displayed in the command output. For control and receive queues, any messages queued after the command runs are not included in the output. For the send queue, any transactions committed after the **cdr check queue** command runs are not included in the output.

If a leaf server name is specified with the **--connect** option, the system connects to the parent server to read information from the **syscdr** database.

Only a DBSA can run the **cdr check queue** command. With a non-root installation, the user who installs the server is the equivalent of the DBSA, unless the user delegates DBSA privileges to a different user.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 17, 21, 48, 62, 94, 99, 100, 196, 222.

Example 1: Control queue report for all servers

The following command waits up to 10 seconds for the control queues on all replication servers to complete before generating a report.

```
cdr check queue -q cntrlq  -w 10s  -a
```

The queue report for the previous command might be:

```
Checking cntrlq queue status for server g_madras ...
cntrlq queue status for g_madras as of Mon Dec  5 12:03:19 2011:      COMPLETE
Checking cntrlq queue status for server g_delhi ...
cntrlq queue status for g_delhi as of Mon Dec  5 12:03:19 2011:       COMPLETE
Checking cntrlq queue status for server g_bombay ...
cntrlq queue status for g_bombay as of Mon Dec  5 12:03:19 2011:      COMPLETE
```

This report indicates that all of the queue items in the control queue at the time the **cdr check queue** command was issued are complete.

Example 2: Send queue report for all servers

The following command waits up to 10 seconds for the send queues on all replication servers to complete before generating a report.

```
cdr check queue -q  sendq  -w 10s  -a
```

The queue report for the previous command might be:

```
Checking sendq queue status for server g_madras ...
Checking sendq queue status for server g_delhi ...
sendq queue status for g_delhi as of Mon Dec  5 12:04:00 2011:       COMPLETE
```

```
sendq queue status for g_madras as of Mon Dec  5 12:04:00 2011:      COMPLETE
Checking sendq queue status for server g_bombay ...
sendq queue status for g_bombay as of Mon Dec  5 12:04:01 2011:      COMPLETE
```

This report indicates that all of the queue items in the send queue at the time the **cdr check queue** command was issued are complete.

Example 3: Send queue report that shows timeout

The following command waits up to 10 seconds for the send queues on all replication servers to complete before generating a report.

```
cdr check queue -q  sendq  -w 10s  -a
```

The queue report for the previous command might be:

```
Checking sendq queue status for server g_madras ...
sendq queue status for g_madras as of Mon Dec  5 12:04:54 2011:      COMPLETE
Checking sendq queue status for server g_delhi ...
sendq queue status for g_delhi as of Mon Dec  5 12:04:54 2011:     INCOMPLETE
  Operation timed out.
command failed -- Command timed out. (21)
```

This report indicates that the send queue for server **g_delhi** had commands that did not complete before the timeout period of 10 seconds elapsed.

Related concepts:
Monitor and troubleshooting Enterprise Replication
Return Codes for the cdr Utility
Enterprise Replication Server administrator

Related tasks:
Deleting a Replication Server
Deleting a Replicate
Deleting a Replicate Set

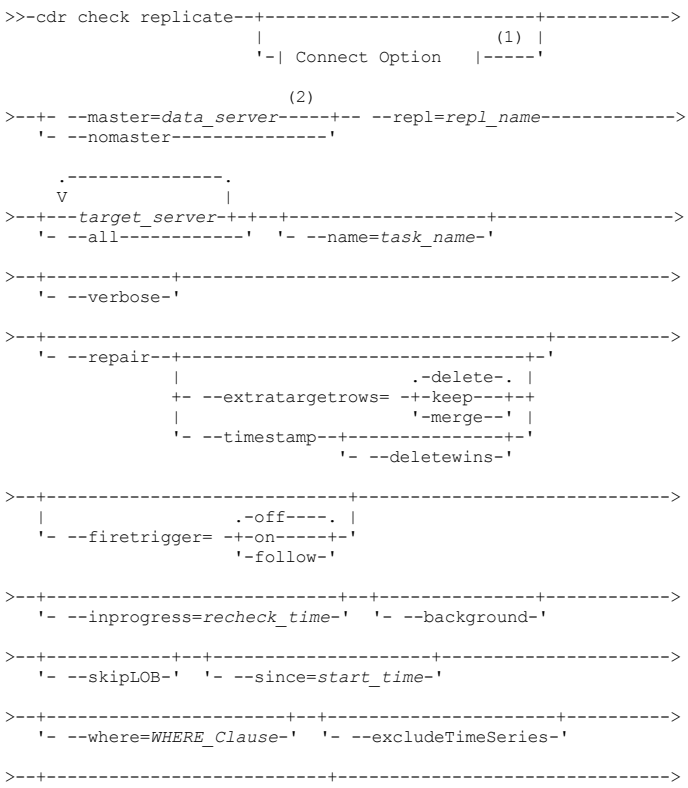
Related reference:
cdr delete replicate
cdr delete replicateset
cdr delete server
cdr delete grid

---

cdr check replicate

The **cdr check replicate** command compares the data on replication servers to create a report that lists data inconsistencies and can optionally repair the inconsistent data within a replicate.

Syntax


>>-cdr check replicate--+-------------------------+------------>
| (1) |
'-| Connect Option |-----'
(2)
>--+- --master=data_server-----+-- --repl=repl_name------------>
'- --nomaster---------------'
.---------------.
V |
>--+---target_server-+-+--+-------------------+----------------->
'- --all------------' '- --name=task_name-'
>--+------------+-->
'- --verbose-'
>--+---+---------->
'- --repair--+---------------------------------+-'
| .-delete-. |
+- --extratargetrows= -+-keep---+-+
| '-merge--' |
'- --timestamp--+---------------+-'
'- --deletewins-'
>--+-----------------------------+------------------------------->
| .-off----. |
'- --firetrigger= -+-on-----+-'
'-follow-'
>--+---------------------------+--+--------------+------------>
'- --inprogress=recheck_time-' '- --background-'
>--+------------+--+--------------------+---------------------->
'- --skipLOB-' '- --since=start_time-'
>--+----------------------+--+---------------------+---------->
'- --where=WHERE_Clause-' '- --excludeTimeSeries-'
>--+---------------------------+------------------------------->
```
```

```
   '- --ignoreHiddenTSElements-'

>--+-------------------------------+-------------------------><
   '- --checksum=checksum_function-'
```

Notes:

1. See Connect Option.
2. Omit if you include the **--timestamp** option.

Element	Purpose	Restrictions	Syntax
checksum_function	Name of the checksum function to use during consistency checking.	The function must be installed and registered on all replication servers.	Long Identifiers
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
recheck_time	The number of seconds to spend rechecking transactions that might be listed as inconsistent because they are not yet applied on the target server.	Must be a positive integer.	
repl_name	Name of the replicate to check.	Must be an existing replicate.	Long Identifiers
start_time	The time from which to check updated rows.	Can have one the following formats: • *number***M** = Include rows updated in the last specified number of minutes. • *number***H** = Include rows updated in the last specified number of hours. • *number***D** = Include rows updated in the last specified number of days. • *number***W** = Include rows updated in the last specified number of weeks. • "*YYYY-MM-DD hh:mm:ss*" = Include rows updated since this time stamp.	The time stamp format follows the convention of the DBTIME environment variable.
target_server	Name of a database server group to check.	Must be the name of an existing database server group in the sqlhosts file.	Long Identifiers
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers
WHERE_Clause	Clause that specifies a subset of table rows to be checked.	You cannot include a **TimeSeries** column in the WHERE clause.	WHERE clause syntax

The following table describes the options for the **cdr check replicate** command.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are checked.
--background	**-B**	Specifies that the operation is run in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name that is specified by the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--checksum=		Specifies the name of an existing checksum function to use during consistency checking. By default, the checksum function that is provided with the database server is run.
--deletewins	**-d**	Specifies that the replicate uses the delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--excludeTimeSeries		Specifies to prevent the checking of time series data.
--extratargetrows=	**-e**	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns. Note: When **cdr check replicate** is used with **-extratargetrows** (or - e) option for SEND-ONLY replicate, server displays the following warning and continue the operation: `WARNING: Extra row option is not applicable to Send-Only participant. '--extratargetrows' option is ignored` Note: When **cdr check replicate** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted: `Error: Receive only participant '%s' can not be a master node in data synchronization task`

Long Form	Short Form	Meaning
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while data is synchronizing: • **off**: (default) do not fire triggers at target servers during synchronization • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--inprogress=	**-i**	Specifies to spend more than the default time to recheck inconsistent rows that might be in the process of being applied on target servers. If the **--inprogress=** option is not set, inconsistent rows are rechecked for up to five seconds.
--master=	**-m**	Specifies the database server to use as the reference copy of the data. You cannot use the **--master** option with the **--timestamp** option.
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--nomaster	**-N**	Specifies that the replicate is configured as a data consolidation system in which the multiple primary servers only send data and the single target server only receives data.
--repair	**-R**	Specifies that rows that are found to be inconsistent are repaired.
--repl=	**-r**	Specifies the name of the replicate to check.
--since=	**-S**	Specifies the time from which to check updated rows. The replicate must be using the time stamp or delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--skipLOB	**-L**	Specifies that large objects are not checked.
--timestamp	**-t**	Specifies to repair inconsistent rows based on the latest time stamp among all the participants. The replicate must use the time stamp or delete wins conflict resolution rule. You cannot use the **--master** option with the **--timestamp** option. You cannot use this option for replicates that include **TimeSeries** columns.
--verbose	**-v**	Specifies that the consistency report shows specific inconsistent values.
--where=	**-w**	Specifies what data to check with a WHERE clause. You cannot include a **TimeSeries** column in the WHERE clause.

Usage

Use the **cdr check replicate** command to check the consistency of data between multiple database servers for a specific replicate. The **cdr check replicate** command compares all rows on all specified database servers against the data in the reference server and produces a consistency report. If you include the **--verbose** option, the report lists every inconsistent row value; otherwise, the report summarizes inconsistent rows.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

If you want to monitor the progress of the check operation, include the **--name** option and specify a name for the progress report task. Then, run the **cdr stats check** command and specify the progress report task name.

Depending on the state of the data in your database when you run the **cdr check** command, the system might also run an UPDATE STATISTICS command.

If replicated transactions are active when the **cdr check replicate** command is running, the consistency report might include rows that are temporarily inconsistent until those transactions are applied at the target server. By default, the **cdr check replicate** command rechecks inconsistent rows for up to five seconds after the initial check is completed. If you find your transaction latency is longer than five seconds, you can extend the recheck time period by using the **--inprogress** option to specify a longer interval. After the initial recheck, inconsistent transactions are rechecked until there are no inconsistent transactions or the number of seconds specified by the **--inprogress** option elapses. In general, set the recheck time to a little longer than your average transaction latency because if repairing inconsistencies causes spooling in the send queue, transaction latency might increase during a repair. View your transaction latency with the **cdr view apply** command.

You can improve the performance of consistency checks by limiting the amount of data that is checked by using one or more of the following options:

- Check from a specific time with the **--since** option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.
- Check a subset of the data with the **--where** option. For example, if you have a corrupted table fragment on a server, you can specify to check only the data in that fragment.
- Skip the checking of large objects with the **--skipLOB** option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.

You can run a consistency check as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the **--name** option so that you can monitor the check with the **cdr stats check** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database. If you use the **--background** option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If you have large tables, you can speed consistency checking by indexing the **ifx_replcheck** shadow column.

If your replication system is configured for data consolidation and the primary servers include the **S** option in their participant definitions, you must include the **--nomaster** option.

If you include the **--repair** option, the **cdr check replicate** command repairs inconsistent rows so that they match the rows on the reference server. The **cdr check replicate** command uses direct synchronization as a foreground process when repairing inconsistent rows. The **cdr check replicate** command with the**--repair** option does the following tasks:

1. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
2. Performs an index scan on the replication key index at both the source server and the target server to create a checksum and identify inconsistent rows.
3. Replicates inconsistent rows from the source server to the target server by doing a dummy update of the source server, which might result in increased logging activity. Rows are not replicated to participants that include the **S** option in the participant definition because those participants only send data.
4. Runs a check to determine whether any rows remain inconsistent. Rows can be temporarily inconsistent if not all transactions are complete on the target server.
5. If any rows are inconsistent, reruns the check for up to five seconds, or for up to the number of seconds specified by the **--inprogress** option.
6. Deletes the shadow replicate.
7. Displays the consistency report.

To repair replicate sets based on the latest time stamps among the participants instead of based on a master server, use the **--repair** option with the **--timestamp** option. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option. A time stamp repair evaluates extra and mismatched rows according to the rules of the time stamp or delete wins conflict resolution rules. The reference server in a time stamp repair is the server with the lowest replication key.

The following table describes the columns of the consistency report.

Table 1. Consistency Report Description

Column name	Description
Node	The name of the replication server.
Rows	The number of rows that are checked in the participant. If you included the **--since** or **--where** options, this number indicates the number of rows that fit the filter conditions. The number of rows that are checked with the **--since** option might be different on different servers, because of replication latency. Some rows might be checked on a source server to verify target server rows even if those rows on the source server did not originally fit the filter conditions.
Elements	For replicates that include **TimeSeries** columns the Elements column is shown instead of the Rows column. The Elements column shows the number of time series elements that are checked in the participant.
Extra	The number of rows on the target server that do not exist on the reference server. For the reference server, this number is always 0.
Missing	The number of rows on the reference server that do not exist on the target server. For the reference server, this number is always 0.
Mismatch	The number of rows on the target server that are not consistent with the corresponding rows on the reference server. For the reference server, this number is always 0.
Total Mismatch	For replicates that include **TimeSeries** columns the Total Mismatch column is shown instead of the Mismatch column. The Total Mismatch column shows the number of rows on the target server that are not consistent with the corresponding rows on the reference server. If the number in this column is greater than the number in the TmSr Rltd Mismatch column, the additional rows are inconsistent in a way that does not involve a **TimeSeries** column. For the reference server, this number is always 0.
TmSr Rltd Mismatch	For replicates that include **TimeSeries** columns the TmSr Rltd Mismatch column shows the number rows on the target server that do not have the same time series properties as the corresponding rows on the reference server because of time series properties. The following time series properties are checked: • Whether the **TimeSeries** column is NULL • The origin of the time series • The calendar definition • Whether the time series is regular or irregular • The time series instance ID • The time series threshold For the reference server, this number is always 0.
Other Mismatch	For replicates that include **TimeSeries** columns the Other Mismatch column shows the number of elements that are in mismatched rows on the target server. For the reference server, this number is always 0.
Processed	The number of rows that are processed to correct inconsistent rows. The number of processed rows on the reference server is equal to the number of mismatched rows plus missing rows on the target servers. The number of processed rows for a target server is usually equal to the number of extra rows it has. If a row has child rows, then the number of processed rows can be greater than the number of extra rows because the child rows must be deleted as well. If the **--extratargetrows** option is set to **keep**, then extra rows are not deleted from the target and those rows are not added to the Processed column. If the **--extratargetrows** option is set to **merge**, then those rows are replicated to the reference server and are listed in the Processed column for the target server. For a time stamp repair, the time stamp or delete wins conflict resolution rule is used to determine how to process the row.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 17, 18, 31, 37, 48, 53, 54, 61, 75, 99, 101, 121, 172, 174, 178, 193, 194, 195, 200, 203, 204.

For information about these error codes, see Return Codes for the cdr Utility

Example 1: Summary consistency report

The following command generates a consistency report for a replicate named **repl1**, comparing the data on the server **serv2** with the data on the server **serv1**:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2
```

The summary consistency report shows that the servers are consistent:

```
                ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 52         0         0         0         0
g_serv2                 52         0         0         0         0

              ------   Table scan for repl1 end   ---------
```

This report indicates that the replicate is consistent on these servers.

Example 2: Summary consistency report with repair

The following command generates a consistency report and repairs inconsistent rows on all servers for a replicate named **repl1**:

```
cdr check replicate --master g_serv1 --repl=repl1 --all --repair
```

The consistency report shows that the target server has extra rows:

```
                ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         2
g_serv2                 67         2         2         0         2
g_serv3                 67         0         0         0         0

Validation of repaired rows failed.
WARNING: replicate is not in sync

              ------   Table scan for repl1 end   ---------
```

This report indicates that **g_serv2** has two extra rows and is missing two rows. Two rows were processed on **g_serv1** to replicate the missing rows to **g_serv2**. Also, two rows were processed on **g_serv2** to delete the extra rows. Because the **--extratargetrows** option was not specified, the default behavior of deleting rows on the target servers that are not on the reference server occurred.
In this example, not all repaired rows were validated. Some rows might be still in the process of being applied on the target servers. Using the **--inprogress** option to extend the time of the validation check after the repair might prevent validation failures.

Example 3: Verbose consistency report with repair

The following command generates a verbose consistency report, creates a progress report task, and repairs inconsistent rows on all servers for a replicate named **repl1**:

```
cdr check replicate --master=g_srv1 --replicate=repl1 --all --name=task1 \
 --verbose --repair
```

The verbose consistency report shows details of the repaired rows:

```
             ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Creating Shadow Repl sync_20104_1310721_1219952381
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_srv1                 424         0         0         0        11
g_srv2                 416         3        11         0         3

The repair operation completed. Validating the repaired rows ...
Validation failed for the following rows:

row missing on <g_srv2>
key:c1:424
------------------------------------------------------------------
row missing on <g_srv2>
key:c1:425
------------------------------------------------------------------
row missing on <g_srv2>
key:c1:426
------------------------------------------------------------------
marking completed on g_srv1 status 0

              ------   Table scan for repl1 end   ---------
```

This report indicates that the first check found three extra rows and 11 missing rows on the server **g_srv2**. After the repair operation and subsequent recheck, three rows were still missing on **g_srv2**. The progress report information can be accessed with the **cdr stats check task1** command.

Example 4: Repeating verbose consistency report without repair

The following command generates a verbose consistency report for a replicate named **repl1**, comparing the data on the server **serv2** with the data on the server **serv1**, and rechecks inconsistent rows for up to 20 seconds:

```
cdr check replicate --master g_serv1 --repl=repl_1 g_serv2 --all \
--verbose --inprogress=20
```

The verbose consistency report shows details for the inconsistent rows:

```
               ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
data mismatch between g_serv1 and g_serv2
item_num:1
order_num:1011
          lname
g_serv1   Pauly
g_serv2   Pauli
--------------------------------------------------
row missing on g_serv2
item_num:1
order_num:1014
--------------------------------------------------
row missing on g_serv2
item_num:2
order_num:1014
--------------------------------------------------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         0
g_serv2                 65         0         2         1         0

WARNING: replicate is not in sync

               ------   Table scan for repl1 end   ---------
```

This report indicates that there is one inconsistent row on **g_serv2**. The replication key for that row is the combination of the **item_num** column and the **order_num** column. The row that is inconsistent is the one that has the item number `1` and the order number `1011`. There are two rows that are missing on **g_serv2**, each identified by its replication key value.

Example 5: Summary consistency report with time filter

The following command generates a summary consistency report for the data that was updated in the last five minutes:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2 --since=5M
```

The consistency report shows that the servers are consistent:

```
            ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                  2         0         0         0         0
g_serv2                  2         0         0         0         0

            ------   Table scan for repl1 end   ---------
```

Only two rows were checked on each server (the Rows column) because only two rows were updated in the last five minutes.

Example 6: Consistency check and repair with time filter

The following command generates a summary consistency report for the data that was updated since July 4, 2008 at 12:30:00 local time:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2 \
--since="2008-07-04 12:30:00"
```

Example 7: Summary consistency report and repair with data filters

The following command generates a consistency report and repairs the data where the **region** column equals `East`:

```
cdr check replicate --master=g_serv1 --repl=repl1 --repair g_serv2 \
--where="region = 'East'"
```

Example 8: Repair inconsistencies based on time stamp

The following command repairs inconsistencies based on the most recent time stamps for the **repl1** replicate on all replication servers:

```
cdr check replicate --repl=repl1 --all --repair --timestamp
```

The master server is not specified because the **--timestamp** option is used.

The consistency report shows that the three servers are not consistent:

```
            ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         4        10
g_serv2                 67         0         2         3         0
g_serv3                 67         0         5         0         4
```

```
WARNING: replicate is not in sync

          ------   Table scan for repl1 end   ---------
```

The value in the Extra column is always 0. In this example, seven rows are replicated from the **g_serv1** server to fix missing rows. The **g_serv1** server also replicated three rows to fix mismatched rows on the **g_serv2** server. The **g_serv3** server replicated four rows to resolve mismatched rows on the **g_serv1** server.

Example 9: Check a replicate that includes a TimeSeries column

The following command checks the replicate named **repl2**, which includes a **TimeSeries** column:

```
cdr check replicate --repl=repl2 --master=g_3 --all
```

The following consistency report shows that the source server, **g_3**, has more elements than the target server, **g_4**:

```
           ------   Table scan for repl2 start  --------

                                                 Total    TmSr Rltd
Node                    Rows     Extra   Missing  Mismatch  Mismatch Processed
----------------- ---------- ---------- ---------- ---------- ---------- ----------
g_3                        1          0          0          0          0          0
g_4                        1          0          0          0          0          0

TimeSeries Column: raw_reads
                                                              Other
Node               Elements     Extra   Missing  Mismatch  Mismatch Processed
----------------- ---------- ---------- ---------- ---------- ---------- ----------
g_3                        2          0          0          0          0          0
g_4                        1          0          1          0          0          0

WARNING: replicate is not in sync
               ------   Table scan for repl2 end   ---------
```

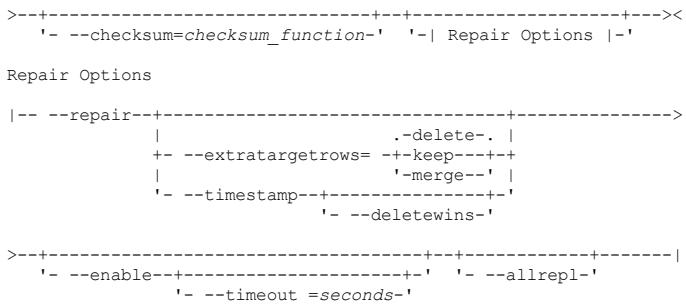
Related concepts:
Interpreting the Consistency Report
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related tasks:
Checking Consistency and Repairing Inconsistent Rows
Indexing the ifx_replcheck Column
Increase the speed of consistency checking
Related reference:
cdr sync replicate
cdr check replicateset
cdr stats check

---

cdr check replicateset

The **cdr check replicateset** command compares the data on replication servers to create a report listing data inconsistencies. Optionally you can use the command to repair the inconsistent data within a replicate.

Syntax



Notes:

1. See Connect Option.
2. Omit if you include the **--timestamp** option.
3. Omit if you include the **--allrepl** option.

Element	Purpose	Restrictions	Syntax
checksum_function	Name of the checksum function to use during consistency checking.	The function must be installed and registered on all replication servers.	Long Identifiers
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
number_processes	The number of parallel processes to use for the command.	The maximum number of processes Enterprise Replication can use is equal to the number of replicates in the replicate set.	
recheck_time	The number of seconds to spend rechecking transactions that might be listed as inconsistent because they are not yet applied on the target server.	Must be a positive integer.	
repl_set	Name of the replicate set. Can be the name of a derived replicate set.		Long Identifiers
seconds	The number of seconds to wait for a disabled replication server to be recognized as active by other replication servers in the domain and how long to wait for control messages queued at peer servers to be applied at newly-enabled server.	Must be an integer value from 0 to 60.	
start_time	The time from which to check updated rows.	Can have one the following formats: • *number***M** = Include rows updated in the last specified number of minutes. • *number***H** = Include rows updated in the last specified number of hours. • *number***D** = Include rows updated in the last specified number of days. • *number***W** = Include rows updated in the last specified number of weeks. • "*YYYY-MM-DD hh*:*mm*:ss" = Include rows updated since this time stamp.	The time stamp format follows the convention of the DBTIME environment variable.
target_server	Name of a database server group to check.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers

The following table describes the **cdr check replicateset** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are checked.
--allrepl	**-A**	Specifies that all replicates, whether they are in a replicate set or not, are repaired. You cannot use the **--replset** option with the **--allrepl** option.
--background	**-B**	Specifies that the operation is run in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name that is specified by the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--checksum=		Specifies the name of an existing checksum function to use during consistency checking. By default, the checksum function that is provided with the database server is run.
--enable	**-E**	Enables replication on the target server if it was disabled by the **cdr disable server** command.

Long Form	Short Form	Meaning
--deletewins	**-d**	Specifies that the replicate uses the delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--excludeTimeSeries		Specifies to prevent the checking of time series data.
--extratargetrows=	**-e**	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns. Note: When **cdr check replicate set** is used with **-extratargetrows** (or -e) option for SEND-ONLY replicate, server displays the following warning and continue the operation: `WARNING: Extra row option isn't applicable to Send-Only participant. '--extratargetrows' option is ignored` Note: When **cdr check replicate set** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted: `Error: Receive only participant '%s' can not be a master node in data synchronization task`
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while data is synchronizing: • **off**: (default) do not fire triggers at target servers during synchronization • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--inprogress=	**-i**	Specifies to spend more than the default time to recheck inconsistent rows that might be in the process of being applied on target servers. If the **--inprogress=** option is not set, inconsistent rows are rechecked for up to five seconds.
--master=	**-m**	Specifies the database server to use as the reference copy of the data. You cannot use the **--master** option with the **--timestamp** option.
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--nomaster	**-N**	Specifies that the replicate is configured as a data consolidation system in which the multiple primary servers only send data and the single target server only receives data.
--process=	**-p**	Specifies to run the command in parallel, using the specified number of processes. At most, Enterprise Replication can use one process for each replicate in the replicate set. If you specify more processes than replicates, the extra processes are not used. Not all replicates can be processed in parallel. For example, if replicates have referential integrity rules, the replicates with the parent tables must be processed before the replicates with the child tables.
--repair	**-R**	Specifies that rows that are found to be inconsistent are repaired.
--replset	**-s**	Specifies the name of the replicate set to check. You cannot use the **--replset** option with the **--allrepl** option.
--skipLOB	**-L**	Specifies that large objects are not checked.
--since=	**-S**	Specifies the time from which to check updated rows. The replicate must be using the time stamp or delete wins conflict resolution rule. You cannot use this option for replicates that include **TimeSeries** columns.
--timeout=	**-w**	Specifies the time to wait for a disabled server to be enabled.
--timestamp	**-t**	Specifies to repair inconsistent rows based on the latest time stamp among all the participants. The replicate must use the time stamp or delete wins conflict resolution rule. You cannot use the **--master** option with the **--timestamp** option. You cannot use this option for replicates that include **TimeSeries** columns.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr check replicateset** command to check the consistency of data between multiple database servers for a replicate set. The **cdr check replicateset** command compares all rows on all specified database servers against the data in the reference server and produces a consistency report. If you include the **--verbose** option, the report lists every inconsistent value; otherwise, the report summarizes inconsistent rows.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

If you want to monitor the progress of the check operation, include the **--name** option and specify a name for the progress report task. Then, run the **cdr stats check** command and specify the progress report task name.

Depending on the state of the data in your database when you run the **cdr check** command, the system might also run an UPDATE STATISTICS command.

If replicated transactions are active when the **cdr check replicateset** command is running, the consistency report might include rows that are temporarily inconsistent until those transactions are applied at the target server. By default, the **cdr check replicateset** command rechecks inconsistent rows for up to five seconds after the initial check is completed. If you find your transaction latency is longer than five seconds, you can extend the recheck time period by using the **--inprogress** option to specify a longer interval. After the initial recheck, inconsistent transactions are rechecked until there are no inconsistent transactions or the number of seconds specified by the **--inprogress** option elapses. In general, set the recheck time to a little longer than your average transaction latency because if repairing inconsistencies causes spooling in the send queue, transaction latency might increase during a repair. View your transaction latency with the **cdr view apply** command.

You can improve the performance of consistency checks by limiting the amount of data that is checked by using one or more of the following options:

- Skip the checking of large objects with the **--skipLOB** option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.
- Check from a specific time with the **--since** option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.

You can significantly improve the performance of checking a replicate set by checking the member replicates in parallel. You specify the number of parallel processes with the **--process** option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

You can run a consistency check as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the **--name** option so that you can monitor the check with the **cdr stats check** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database. If you use the **--background** option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If you have large tables, you can speed consistency checking by indexing the **ifx_replcheck** shadow column.

If your replication system is configured for data consolidation and the primary servers include the **S** option in their participant definitions, you must include the **--nomaster** option.

The **cdr check replicateset** command repairs inconsistent rows so that they match the rows on the reference server. During a repair of inconsistent rows, the **cdr check replicateset** command uses direct synchronization as a foreground process when repairing inconsistent rows. The **cdr check replicateset** command with the **--repair** option performs the following tasks:

1. Determines the order in which to repair tables if they have referential relationships.
2. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
3. Performs an index scan on the replication key index at both the source server and the target server to create a checksum and identify inconsistent rows.
4. Replicates inconsistent rows from the source server to the target server by doing a dummy update of the source server, which might result in increased logging activity. Rows are not replicated to participants that include the **S** option in the participant definition because those participants only send data.
5. Runs a check to determine whether any rows remain inconsistent. Rows can be temporarily inconsistent if not all transactions are complete on the target server.
6. If any rows are inconsistent, reruns the check for up to five seconds, or for up to the number of seconds specified by the **--inprogress** option.
7. Deletes the shadow replicate.
8. Repeats steps 2 through 7 for each replicate in the replicate set.
9. Displays the consistency report.

If you have disabled a server with the **cdr disable server** command, you can enable it and synchronize it by using the **--enable** option with the **--repair** option. You can optionally specify a timeout period with the **--timeout** option.

To repair all replicates, use the **--allrepl** option with the **--repair** option.

To repair replicate sets based on the latest time stamps among the participants instead of based on a master server, use the **--repair** option with the **--timestamp** option. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option. A time stamp repair evaluates extra and mismatched rows according to the rules of the time stamp or delete wins conflict resolution rules.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 11, 17, 18, 31, 37, 48, 53, 54, 61, 75, 99, 101, 121, 166, 172, 174, 193, 194, 195, 200, 203, 204, 213.

For information about these error codes, see Return Codes for the cdr Utility

Example 1: Generate a consistency report

The following command uses two processes to generate a consistency report for each of the two replicates in the set in parallel for a replicate set named **replset1**, comparing the data on the server **serv2** with the data on the server **serv1**:

```
cdr check replicateset --master=g_serv1 --replset=replset_1 g_serv2 \
--process=2
```

The summary consistency report for the previous command might be:

```
Jan 17 2010 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- ---------- ---------- ---------- ---------- ----------
g_serv1                 52          0          0          0          0
g_serv2                 52          0          0          0          0

Jan 17 2010 15:46:55 ------   Table scan for repl1 end   ---------
```

```
Jan 17 2010 15:46:46 ------   Table scan for repl2 start  --------

------   Statistics for repl2 ------
Node                   Rows     Extra   Missing  Mismatch Processed
---------------- --------- --------- --------- --------- ---------
g_serv1                  48         0         0         0         0
g_serv2                  48         0         0         0         0

Jan 17 2010 15:47:05 ------   Table scan for repl2 end   ---------
```

This report indicates that the replicate set is consistent on these servers.

The consistency report for replicate sets shows a series of consistency reports for individual replicates that has the same format as the reports run with the **cdr check replicate** command.

Example 2: Enable and synchronize a replication server

The following command enables a replication server named **g_serv2** and repairs inconsistencies by time stamp on all of its replicate sets:

```
cdr check replicateset  --repair --enable\
--timestamp --allrepl g_serv2
```

The master server is not specified because the **--timestamp** option is used. The replicate set name is not specified because the **--allrepl** option is used.

Example 3: Repair inconsistencies based on time stamp

The following command repairs inconsistencies based on the most recent time stamps for all replicate on all replication servers:

```
cdr check replicateset --all --repair --timestamp --allrepl
```

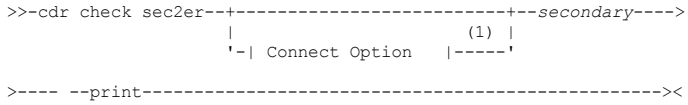
Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related tasks:
Altering multiple tables in a replicate set
Checking Consistency and Repairing Inconsistent Rows
Indexing the ifx_replcheck Column
Increase the speed of consistency checking
Repairing inconsistencies while enabling a replication server
Related reference:
cdr sync replicateset
cdr check replicate
cdr stats check
cdr disable server

---

cdr check sec2er

The **cdr check sec2er** command determines whether a high availability cluster can be converted to replication servers.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
secondary	Name of the secondary server in the cluster.		Long Identifiers

The following table describes the **cdr check sec2er** option.

Long Form	Short Form	Meaning
--print	**-p**	Shows the commands that would be run by the **cdr start sec2er** command during a conversion.

Usage

You must run the **cdr check sec2er** command from a primary server in a cluster with a high-availability data replication secondary or a remote stand-alone secondary server. The output of the **cdr check sec2er** command can show warning messages and error messages:

- Warning messages indicate possible problems for replication after the conversion. You can solve these problems after converting the cluster to replication servers.
- Error messages indicate problems preventing the conversion to replication server. You must solve all error conditions before you run the **cdr start sec2er** command to convert a cluster to replication servers.

Use the **--print** option to display the commands that are run during a conversion.

Depending on the state of the data in your database when you run the **cdr check** command, the system might also run an UPDATE STATISTICS command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error code is returned: 225.

For information about these error codes, see Return Codes for the cdr Utility.

Examples

The following example checks if a cluster consisting of a primary server named **priserv** and a secondary server named **secserv** can be converted to replication servers:

```
cdr check sec2er -c priserv secserv
```

The following output of the **cdr check sec2er** command indicates that conversion would be successful, but that several issues should be addressed either before or after conversion:

```
WARNING:CDR_SERIAL value on priserv can cause collisions.
WARNING:Dbspace is becoming full.
WARNING:Using the same values for CDR_SERIAL can cause collisions.

Secondary conversion to ER is possible.
Errors:0000   Warnings:0003
```

The following output of the **cdr check sec2er** command indicates that conversion will not be successful until the CDR_QDATA_SBSPACE configuration parameter is set in the onconfig file on both the primary and the secondary servers:

```
WARNING:CDR_SERIAL value on priserv can cause collisions.
WARNING:Dbspace is becoming full.
WARNING:Using the same values for CDR_SERIAL can cause collisions.
ERROR:ER sbspace not correctly set up (CDR_QDATA_SBSPACE).

Secondary conversion to ER is not possible.
Errors:0001   Warnings:0003
```

The following output of the **cdr check sec2er** command indicates that conversion will not be successful until the sqlhosts files on both the primary and the secondary servers are correctly configured for Enterprise Replication:

```
WARNING:CDR_SERIAL value on serv1 can cause collisions.
ERROR:Server priserv and server secserv belong to the same group.
WARNING:Dbspace  is becoming full.
ERROR:Server priserv and server secserv belong to the same group.
WARNING:Using the same values for CDR_SERIAL can cause collisions.
FATAL:SQLHOSTS is not set up correctly for ER.
ERROR:SQLHOSTS is not set up correctly for ER.
ERROR:ER sbspace not correctly set up (CDR_QDATA_SBSPACE).

Secondary conversion to ER is not possible.
Errors:0004   Warnings:0003
```

The following example shows the output of the **--print** option, which describes the commands that will be run when the **cdr start sec2er** command is run on the **priserv** server. The servers are defined as replication servers. Any tables that do not have a primary key are altered to add ERKEY shadow columns. A replicate is created and started for each user table on the **priserv** server.

```
$cdr check sec2er --print serv2
Secondary conversion to ER is possible.

Errors:0000   Warnings:0000
--
--  Define ER for the first time
--
cdr define serv -c cdr1 -I cdr1

--
--  Creating Replication Key
--
dbaccess - - <<EOF
database stores_demo;
alter table 'mpruet'.classes add ERKEY;
EOF

--
--  Define the replicates
--
--
--  Defining Replicates for Database stores_demo
--
cdr define repl --connect=cdr1 sec2er_1_1282611664_call_type --master=cdr1 \
   --conflict=always --scope=row \
```

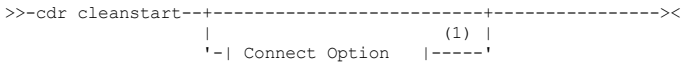
```
   "stores_demo@cdr1:'mpruet'.call_type" \
         "select * from 'mpruet'.call_type"
cdr start repl --connect=cdr1 sec2er_1_1282611664_call_type

cdr define repl --connect=cdr1 sec2er_4_1282611664_cust_calls --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.cust_calls" \
         "select * from 'mpruet'.cust_calls"
cdr start repl --connect=cdr1 sec2er_4_1282611664_cust_calls

cdr define repl --connect=cdr1 sec2er_5_1282611664_customer --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.customer" \
         "select * from 'mpruet'.customer"
cdr start repl --connect=cdr1 sec2er_5_1282611664_customer

cdr define repl --connect=cdr1 sec2er_3_1282611664_classes --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.classes" \
         "select * from 'mpruet'.classes"
cdr start repl --connect=cdr1 sec2er_3_1282611664_classes
--
--  Starting RSS to ER conversion
--
--
--  WARNING:
--
--  DDL statements will not be automatically propagated to the ER server
--  after converting the secondary server into an ER server.  If you
--  create or alter any objects, such as databases, tables, indexes, and
--  so on, you must manually propagate those changes to the ER node and
--  change any replication rules affecting those objects.
--
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr start sec2er
Example of creating a new replication domain by cloning

cdr cleanstart

The **cdr cleanstart** command starts an Enterprise Replication server with empty queues.

Syntax



Notes:

1. See Connect Option.

Usage

The **cdr cleanstart** command starts an Enterprise Replication server, but first empties replication queues of pending transactions. Use this command if synchronizing the server using the **cdr sync** command would be quicker than letting the queues process normally.

If an Enterprise Replication server was restored from a backup, but the restore did not include all log files from the replay position, or the system was not restored to the current log file, advance the log file unique ID past the latest log file unique ID prior to the restore, and then run the **cdr cleanstart** command followed by the **cdr sync** command to synchronize the server.

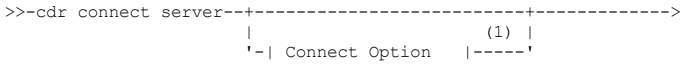
You can run this command from within an SQL statement by using the SQL administration API.

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr start

cdr connect server

The **cdr connect server** command reestablishes a connection to a database server that has been disconnected with a **cdr disconnect server** command.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of database server group to resume.	The database server group must be defined for replication and be disconnected.	Long Identifiers

Usage

When you run the **cdr connect server** command, an event alarm with a class ID of 53 is generated, if that event alarm is enabled.

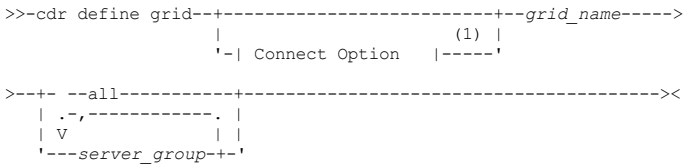
You can run this command from within an SQL statement by using the SQL administration API.

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr define server
cdr delete server
cdr disconnect server
cdr list server
cdr modify server
cdr resume server
cdr suspend server
Enterprise Replication Event Alarms

cdr define grid

The **cdr define grid** command creates a named grid of replication servers to simply administration.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be unique among grid names and replicate set names.	Long Identifiers
server_group	Name of a database server group to add to the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers

The following table describes the **cdr define grid** option.

Long Form	Short Form	Meaning
--all	**-a**	Include all replication servers in the domain.

Usage

You must run the **cdr define grid** command from a replication server that is a member of an Enterprise Replication domain.

Use the **--all** to include all replication servers in the domain in the grid.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 221.

For information on these error codes, see Return Codes for the cdr Utility.

Examples

The following example defines a grid named **grid1** and adds two replication servers to it:

```
cdr define grid grid1 gserv1, gserv2
```

The following example defines a grid named **grid1** and adds all replication servers in the current domain to it:

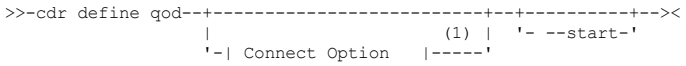
```
cdr define grid grid1 --all
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Creating a grid
Related reference:
cdr change grid
cdr list grid
cdr delete grid

cdr define qod

The **cdr define qod** command defines a master server for monitoring the quality of data (QOD) for replication servers.

Syntax



Notes:

1. See Connect Option.

The following table describes the **cdr define qod** option.

Long Form	Short Form	Meaning
--start	**-s**	Specifies to start quality of data monitoring.

Usage

If Connection Manager service-level agreements (SLAs) use a apply-failure or transaction-latency policy, the Connection Manager uses QOD information to decide where to route client connection requests.

Quality of data information is used for the following SLA policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.

You can start monitoring by including the **--start** option or by running the **cdr start qod** command after the **cdr define qod** command.

If the monitoring of data quality is already enabled, running the **cdr define qod** command changes the master server.

You must run the **cdr define qod** command from a root server.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 217.

For information on error codes, see Return Codes for the cdr Utility.

Example 1: Defining a master server

The following command defines **server_1** as the master server for quality of data monitoring:

```
cdr define qod server_1
```

After you have defined **server_1** as the master server, you must run **cdr define qod** on **server_1** to begin quality of data monitoring.

Example 2: Defining a master server and starting quality of data monitoring

The following command connects to **server_2**, defines **server_2** as the master server, and then starts quality of data monitoring:

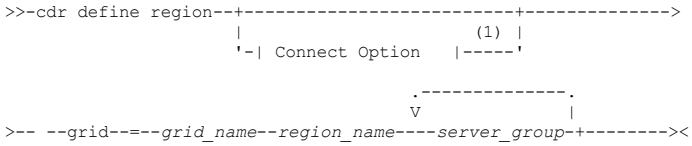
```
cdr define qod -c server_2 --start
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr start qod
cdr stop qod
cdr reset qod
Related information:
SLA Connection Manager configuration parameter

cdr define region

The **cdr define region** command creates a region that contains a subset of the servers in a grid.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of a defined grid.	Long Identifiers
region_name	Name of the region.	Must be unique among region names, grid names, and replicate set names.	Long Identifiers
server_group	Name of a database server group to add to the region.	Must be the name of a defined grid server.	Long Identifiers

The following table describes the **cdr define region** option.

Long Form	Short Form	Meaning
--grid	**-g**	Specifies the grid that contains the servers to include in the region.

Usage

Use the **cdr define region** command to define a region of a grid. You can use a region name in a grid query to limit the servers on which the query is run. Regions contain servers from a single grid. You can define multiple regions for the same grid. Regions can overlap or be contained by another region. You can create an unlimited number of regions in a grid.

You cannot change a region. If you want to add or remove a grid server from a region, delete and re-create the region. Delete the region by running the **cdr delete region** command. Re-create the region with a different set of grid servers by running the **cdr define region** command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 227.

For information about these error codes, see Return Codes for the cdr Utility.

Examples

The following command creates a region that contains two servers and is named **northwest**. The region is created in the grid named **mygrid1**.

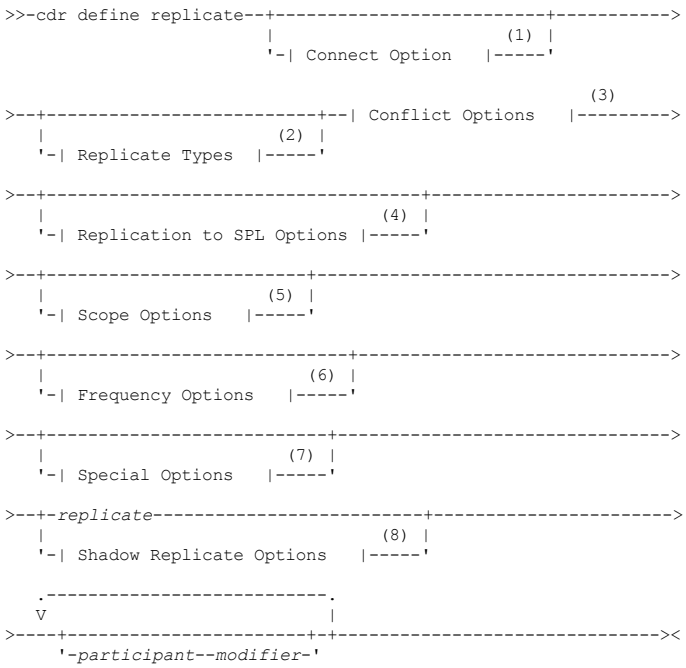
```
cdr define region --grid=mygrid1 northwest server_or server_wa
```

Related concepts:
Grid queries
Related tasks:
Defining tables for grid queries
Related reference:
ifx_grid_connect() procedure
cdr define region
cdr change gridtable
cdr remaster gridtable
ifx_node_id() function
ifx_node_name() function
Related information:
SELECT_GRID session environment option
SELECT_GRID_ALL session environment option
GRID clause

cdr define replicate

The **cdr define replicate** command defines a replicate on the specified replication servers.

Syntax



Notes:

1. See Connect Option.
2. See Replicate Types.
3. See Conflict Options.
4. See Replication to SPL routine.
5. See Scope Options.
6. See Frequency Options.
7. See Special Options.
8. See Shadow Replicate Options.

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		Participant and participant modifier
participant	Name of a participant in the replication.	The participant must exist.	Participant and participant modifier
replicate	Name of the new replicate.	The replicate name must be unique.	Long Identifiers

Usage

All servers that are specified as participants for the replicate must be online and the **cdr** utility must be able to connect to each participant.

To be useful, a replicate must include at least two participants. You can define a replicate that has one or no participant, but before you can use that replicate, you must use the **cdr change replicate** command to add more participants. You cannot start and stop replicates that have no participants.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

When you define a replicate, the replicate does not begin until you explicitly change its state to active by running the **cdr start replicate** command.
Important: Do not create more than one replicate definition for each row and column combination to replicate. If the participant is the same, Enterprise Replication attempts to insert duplicate values during replication.
The maximum number of replicates that you can define as participants on a particular replication server is 32767.

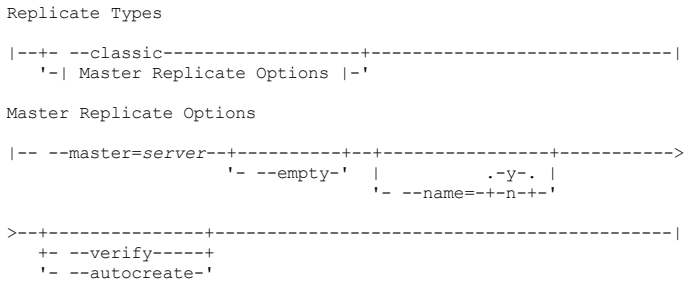
You can run this command from within an SQL statement by using the SQL administration API.

Replicate Types

By default, replicates are master replicates. If you do not specify a master server, the master replicate is based on the first participant. A master replicate uses saved dictionary information about the attributes of replicated columns to verify that participants conform to the specified schema. You must specify at least one participant when you create a master replicate. All participants that are specified are verified when the **cdr define replicate** or **cdr change replicate** command is run. If any participant does not conform to the master definition, the command fails and that local participant is disabled. If a participant you specify does not contain the master replicate table, Enterprise Replication automatically creates the table on the participant, based on the master replicate dictionary information. All database servers that have master replicates must be able to establish a direct connection with the master replicate database server.

When you create a master replicate and do not include a participant modifier, the database server internally generates a participant modifier with SELECT statement that lists each column name in the table. The database server requires the individual column names to verify the schema. If the length of the SELECT statement exceeds 15 000 ASCII characters, replicate creation fails. If your column names are too long, you can create a classic replicate, which has a generated participant modifier of SELECT *.

If you do not want to verify the schema, create a classic replicate. For example, if you want to create a data consolidation system in which one server only receives data from other servers that only send data, create a classic replicate by including the **--classic** option.



Element	Purpose	Restrictions	Syntax
server	Name of the database server group from which to base the master replicate definition.	The name must be the name of a database server group.	Long Identifiers

The following table describes the replicate type options.

Long Form	Short Form	Meaning
--autocreate	**-u**	Specifies that if the tables in the master replicate definition do not exist in the databases on the target servers, they are created automatically. However, the tables cannot contain columns with user-defined data types. The tables are created in the same dbspace as the database. Note: Tables that are created with the **--autocreate** option do not automatically include indexes that are not based on the replication key, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with the **--autocreate** option require the use of these objects you must manually create those objects. You cannot use this option for replicates that include **TimeSeries** columns.
--classic		Specifies that the replicate being created is a classic replicate.
--empty	**-t**	Specifies that the participant on the server that is specified with the **--master** option is used as the basis of the master replicate, but is not added to the replicate.
--master=	**-M**	Specifies that the replicate being created is a master replicate. If you omit this option, the master replicate is based on the first participant.
--name=	**-n**	Specifies whether the master replicate has column name verification in addition to column data type verification. Valid values are: • **--name=y** = Default. Column names are verified to be the same on all participants. • **--name=n** = Column names are not verified and discrepancies can exist.
--verify	**-v**	Specifies that the **cdr define replicate** command verifies the database, tables, and column data types against the master replicate definition on all listed servers.

Replication to SPL routine

At target participant, ‘replication to SPL routine’ type replicate definition causes SPL routine to be executed instead of applying data to target table. Target participant for “replication to SPL routine” replicate definition can be configured to be same as source database, different database on the same server, or remote peer Enterprise Replication server. “Replication to SPL routine” replicate definition does not enforce the requirement to have primary key, unique index or ER key on the replicated table.
Note: Even though data is applied to stored procedure routine, target table definition must exist.
For more information see, Replication to SPL routine

```
|--+--splname=spl_routine_name----+-------+---------------------+--------->
   |                              |       |                .-y-. |
'--'--jsonsplname=spl_routine_name-'       '- -- cascaderepl=-+-n-+-'
```

Long Form	Meaning
--splname	Stored procedure routine name to apply data to. SPL routine must exist at all participants. Column list for SPL routine extracted from replicate participant select statement column projection list.
--jsonsplname	Stored procedure routine name to apply data to. SPL routine and table definition must exist at all participants. Input argument for SPL routine must be a JSON document. --jsonsplname option is mutually exclusive to --splname option.
--cascaderepl	Enable cascade replication. Required if replication to SPL needs to be executed for the data applied through Enterprise Replication.

--splname option stored procedure argument list:

- Optype char(1) – operation type. Values include
  - I – Insert
  - U – Update
  - D – Delete
- Soucre_id integer – Source server id. Same as group id.
- Committime integer – Transaction commit time.
- Txnid bigint – Transaction id.
- Before value column list.
- After value column list.

Note: Column list for SPL routine extracted from select statement projection list

Conflict Options

The **--conflict** options specify how Enterprise Replication resolves data conflicts at the database server.

```
Conflict Options

|-- --conflict=-------------------------------------------------->

>--+-always-----------------------------------------+-----------|
   +-ignore---------------------------------------+
   +-SPL_routine--+-------------+------------------+
   |              '- --optimize-'                  |
   +-timestamp--+--------------------------------+-+
   |            '-,--SPL_routine--+-------------+-' |
   |                              '- --optimize-'   |
   '-deletewins-------------------------------------'
```

Element	Purpose	Restrictions	Syntax
SPL_routine	SPL routine for conflict resolution	The SPL routine must exist.	Long Identifiers

The following table describes the **--conflict** options.

Long Form	Short Form	Meaning
--conflict=	**-C**	Specifies the rule that is used for conflict resolution. • Use the **always** option if you do not want Enterprise Replication to resolve conflicts, but you do want replicated changes to be applied even if the operations are not the same on the source and target servers. Use the always-apply conflict resolution rule only with a primary-target replication system. If you use always-apply with an update-anywhere replication system, your data might become inconsistent. You must use the always-apply rule if your replicate includes **TimeSeries** data types. • Use the **ignore** option if you do not want Enterprise Replication to resolve conflicts. • Use the **timestamp** option to have the row or transaction with the most recent time stamp take precedence in a conflict. • Use the **deletewins** option to have the row or transaction with a DELETE operation, or otherwise with the most recent time stamp, take precedence in a conflict. The delete wins conflict resolution rule prevents upserts. The action that Enterprise Replication takes depends on the scope.

Long Form	Short Form	Meaning
--optimize	**-O**	Specifies that the SPL routine is optimized. An optimized SPL routine is called only when a collision is detected and the row to be replicated fails to meet one of the following two conditions: • It is from the same database server that last updated the local row on the target table. • It has a time stamp greater than or equal to that of the local row. When this option is not present, Enterprise Replication always calls the SPL routine that is defined for the replicate when a conflict is detected.

Scope Options

The **--scope** options specify the scope of Enterprise Replication conflict resolution.

Scope Options

The following table describes the **--scope** option.

Long Form	Short Form	Meaning
--scope=	**-S**	Specifies the scope that is used when Enterprise Replication encounters a problem with data or a conflict occurs. • **--scope=row** = Evaluate one row at a time and apply the replicated rows that win the conflict resolution with the target rows. • **--scope=transaction** = Default. Apply the entire transaction if the replicated transaction wins the conflict resolution. When you specify the scope, you can abbreviate **transaction** to **tra**.

Special Options

Special Options

Element	Purpose	Restrictions	Syntax
column_name	The name of a column that is included in a unique index or constraint	The column must exist.	Long Identifiers

The following table describes the special options to the **cdr define replicate** command.

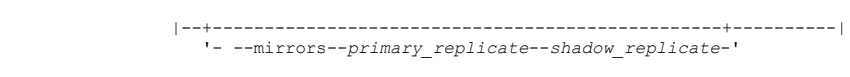
Long Form	Short Form	Meaning
--alwaysRepLOBS=		Specifies whether columns that contain unchanged large objects are included in replicated rows: • **--alwaysRepLOBS=n**: Default. Columns that contain unchanged large objects are not replicated. • **--alwaysRepLOBS=y**: Columns that contain large objects are always included in replicated rows.

Long Form	Short Form	Meaning
--anyUniqueKey	**-U**	Specifies that the replication key is detected automatically from the following sources in the follow order: • A primary key that is defined on the table • ERKEY shadow columns that are included in the table • Any unique key or unique constraint that is defined on the table The replicate must be a strictly mastered replicate. If the table includes ERKEY shadow columns, those columns are included in the participant definition only if the table does not have a primary key.
--ats	**-A**	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database.
--erkey	**-K**	Adds the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, to the participant definition, if the table includes the ERKEY shadow columns. An index that is created on the ERKEY shadow columns is used as the replication key, unless the **--key** option is included.
--firetrigger	**-T**	Specifies that the rows that the replicate inserts fire triggers at the destination.
--floatieee	**-I**	Transfers replicated floating-point numbers in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format. Use this option for all new replicate definitions.
--floatcanon	**-F**	Transfers replicated floating-point numbers in machine-independent decimal representation. This format is portable, but can lose accuracy. This format is provided for compatibility with earlier versions only; use **--floatieee** for all new replicate definitions.
--fullrow=	**-f**	Specifies whether to replicate full rows or only the changed columns: • **--fullrow=y** = Default. Indicates to replicate the full row and to enable upserts. If you also specify **deletewins** as the conflict resolution rule, upserts are disabled. • **--fullrow=n** = Indicates to replicate only changed columns and disable upserts.
--ignoredel=	**-D**	Specifies whether to retain deleted rows on other nodes: • **--ignoredel=y** = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify **deletewins** as the conflict resolution rule. • **--ignoredel=n** = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--key=	**-k**	Specifies the columns that are included in an existing unique index or unique constraint to use as the replication key. All the columns that are included in the unique index or constraint must be listed, in the same order as the columns are listed in the index or constraint definition. The replicate must be a strictly mastered replicate. The unique index or constraint that the **--key** option specifies is used as the replication key even if the table has an existing primary key or ERKEY columns.
--ris	**-R**	Activates row-information spooling for replicate row data that fails conflict resolution or encounters replication order problems.
--serial	**-s**	Specifies that replicated transactions for the replicate are applied serially instead of in parallel.
--UTF8=	None	Specifies whether to enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets. • **--UTF8=y** Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data that is contained within opaque data types. You cannot use **--UTF8=y** for replicates that contain **TimeSeries** data types, user-defined data types, or DataBlade module data types. • **–UTF8=n** Indicates that code set conversion is ignored.

Shadow Replicate Options

A shadow replicate is a copy of an existing, or primary, replicate. You must create a shadow replicate to manually remaster of a replicate that is defined with the **-n** option. After you create the shadow replicate, the next step in manual remastering is to switch the primary replicate and the shadow replicate by running the **cdr swap shadow** command.

Shadow Replicate Options



Element	Purpose	Restrictions	Syntax

Element	Purpose	Restrictions	Syntax
primary_replicate	Name of the replicate on which to base the shadow replicate.	The replicate must exist. The replicate name must be unique.	Long Identifiers
shadow_replicate	Name of the shadow replicate to create.	The replicate name must be unique.	Long Identifiers

The following table describes the shadow replicate option to **cdr define replicate**.

Long Form	Short Form	Meaning
--mirrors	**-m**	Specifies that the replicate created is a shadow replicate based on an existing primary replicate.

Example 1: Define a replicate with two participants

The following example defines a replicate with two participants:

```
cdr define repl --conflict=timestamp,sp1 \
--scope=tran --ats --fullrow=n --floatieee newrepl \
"db1@iowa:antonio.table1" "select * from table1" \
"db2@utah:carlo.table2" "select * from table2"
```

Line 1 of the example specifies a primary conflict resolution rule of timestamp. If the primary rule fails, the SPL routine **sp1** is run to resolve the conflict. Because no database server is specified here (or on any later line), the command connects to the database server named in the INFORMIXSERVER environment variable.

Line 2 specifies that the replicate has a transaction scope for conflict resolution scope and enables aborted transaction spooling. Enterprise Replication replicates only the rows that changed and uses the IEEE floating point format to send floating-point numbers across dissimilar platforms. The final item specifies the name of the replicate, **newrepl**.

Line 3 defines the first participant, **"db1@iowa:antonio.table1"**, with the SELECT statement **"select * from table1"**.

Line 4 defines a second participant, **"db2@utah:carlo.table2"**, with the SELECT statement **"select * from table2"**.

Example 2: Define a replicate with a frequency of every five hours

This example is the same as the preceding example with the following exceptions:

- Line 1 instructs Enterprise Replication to use the global catalog on database server **ohio**.
- Line 2 specifies that the data is replicated every five hours.

```
cdr def repl -c ohio -C timestamp,sp1 \
-S tran -A -e 5:00 -I newrepl \
"db1@iowa:antonio.table1" "select * from table1" \
"db2@utah:carlo.table2" "select * from table2"
```

Example 3: Define a master replicate

The following example defines a master replicate:

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl \
"db1@iowa:antonio.table1" "select * from table1"
```

Line 1 instructs Enterprise Replication to create a master replicate based on the replicate information from the database server **iowa**. Line 2 specifies the delete wins conflict resolution rule, a transaction scope, and that the name of the replicate is **newrepl**. Line 3 specifies the table and columns included in the master replicate.

Example 4: Define a master replicate and create a table on a participant

This example is the same as the previous example except that it specifies a second participant in Line 4. The second participant (**utah**) does not have the table **table1** specified in its participant and modifier syntax. The **-u** option specifies to create the table **table1** on the **utah** server.

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl -u\
"db1@iowa:antonio.table1" "select * from table1 \
"db2@utah:carlo.table1" "select * from table1"
```

Example 5: Define a replicate with the ERKEY shadow columns

This example defines a master replicate similar to the one in example 3, and includes the ERKEY shadow columns for the replication key.

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl --erkey\
"db1@iowa:antonio.table1" "select * from table1"
```

Example 6: Define a data consolidation system

This example defines a replicate for a data consolidation system in which one target server receives replicated data from four primary servers.

```
cdr def repl -c london \
sales -C always\
"db0@london:user.world_sales" "select * from world_sales"\
"S db1@rome:user1.sales_rome" "select * from sales_rome"\
"S db2@tokyo:user2.sales_tokyo" "select * from sales_tokyo"\
"S db3@perth:user3.sales_perth" "select * from sales_perth"\
"S db4@ny:user4.sales_ny" "select * from sales_ny"\
```

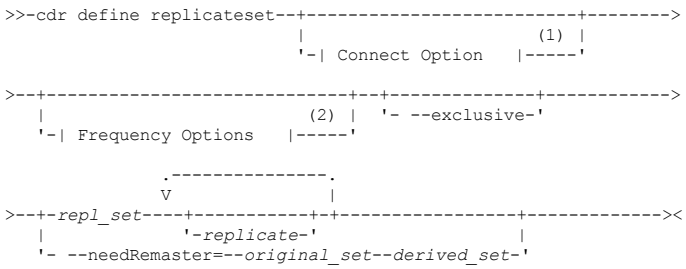
The **S** options in the participant definitions indicate that the **rome**, **tokyo**, **perth**, and **ny** servers can only send replicated data to the **london** server.

Related concepts:
Replicate types
Conflict Resolution
Conflict Resolution Scope
Failed Transaction (ATS and RIS) Files
Frequency Options
Preparing for Role Separation (UNIX)
Replication of TimeSeries data types
Enterprise Replication Server administrator
Related tasks:
Enabling ATS and RIS File Generation
Specifying Conflict Resolution Rules and Scope
Enabling Triggers
Using the IEEE Floating Point or Canonical Format
Defining Replication Servers
Setting Up Failed Transaction Logging
Preparing tables without primary keys
Related reference:
cdr change replicate
cdr change replicateset
cdr delete replicate
cdr list replicate
cdr modify replicate
cdr resume replicate
cdr start replicate
cdr stop replicate
cdr suspend replicate
cdr swap shadow
cdr define template
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset
Replicate only changed columns
Participant and participant modifier

cdr define replicateset

The **cdr define replicateset** command defines a replicate set on all the servers that are included as participants in the replicates. A replicate set is a collection of several replicates to be managed together.

Syntax



Notes:

1. See Connect Option.
2. See Frequency Options.

Element	Purpose	Restrictions	Syntax
derived_set	Name of a temporary replicate set to create that contains only replicates that must be remastered.	The name must be unique and cannot be the same as a replicate name.	Long Identifiers
original_set	Name of an existing replicate set that contains replicates that must be remastered.	The replicate set must exist.	Long Identifiers
repl_set	Name of replicate set to create.	The name must be unique and cannot be the same as a replicate name.	Long Identifiers

Element	Purpose	Restrictions	Syntax
replicate	Name of a replicate to be included in the replicate set.	The replicate must exist.	Long Identifiers

The following table describes the options to the **cdr define replicateset** command.

Long Form	Short Form	Meaning
--exclusive	**-X**	Creates an exclusive replicate set. Replicates that belong to this replicate set cannot belong to any other replicate sets.
--needRemaster=	**-n**	Creates a derived replicate set that contains replicates that have schema changes and must be remastered, and any classic replicates. All classic replicates are converted to master replicates regardless of whether they have schema changes.

Usage

All servers that are specified as participants for all the specified replicates must be online and the **cdr** utility must be able to connect to each participant.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

Any valid replicate can be defined as part of a replicate set. A replicate can belong to more than one non-exclusive replicate set, but to only one exclusive replicate set.

When you create an exclusive replicate set, the state is initially set to active.

To create an exclusive replicate set and make it active

1. Create an empty replicate set.
2. Stop the replicate set.
3. Add replicates to the replicate set.
4. Set the state of the replicate set to active by running **cdr start replicateset**.

Because individual replicates in a non-exclusive replicate set can have different states, the non-exclusive replicate set itself has no state. You cannot change whether a replicate set is exclusive or not.

If you change the schema of multiple replicated tables for replicates that belong to the same replicate set, you can create a derived replicate set so that you can remaster all the replicates with one command. Use the **--needRemaster** option to specify the existing replicate set and the name of the derived replicate set. Then run the **cdr remaster replicateset** command.

You can run this command from within an SQL statement by using the SQL administration API.

Example: Define a non-exclusive replicate set

The following command connects to the default server and defines the non-exclusive replicate set **accounts_set** with replicates **repl1**, **repl2**, and **repl3**:

```
cdr def replset accounts_set repl1 repl2 repl3
```

Example: Define an exclusive replicate set

The following command connects to the server **olive** and defines the exclusive replicate set **market_set** with replicates **basil** and **thyme**:

```
cdr def replset --connect=olive --exclusive market_set basil thyme
```

Example: Define a derived replicate set

The following command defines a derived replicate set named **derived_accounts** that is based on the replicate set **accounts_set**:

```
cdr define replicateset --needRemaster=accounts_set derived_accounts
```

Related concepts:
Define replicate sets
Frequency Options
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related tasks:
Altering multiple tables in a replicate set
Exclusive Replicate Sets
Related reference:
cdr change replicateset
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset

cdr define server

The **cdr define server** command defines a replication server in an Enterprise Replication domain. You can add a replication server to an existing domain or create a new domain.

Syntax

```
>>-cdr define server--+-------------------------+-------------->
                     |                   (1) |
                     '-| Connect Option   |-----'

>--+---------------------+---------------------------------------->
   '-| Dynamic Options |-'

>--+-------------------------------------+-- --init-------------->
   '- --sync=sync_server-+------------+-'
                         +- --nonroot-+
                         '- --leaf----'

>--server_group-------------------------------------------------><
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to add to an Enterprise Replication domain.	Must be the name of an existing database server group in the sqlhosts information.	
sync_server	Name of a replication server that is a member of the domain into which you are adding a server.	Must be an existing replication server. The server must be online.	Long Identifiers

The following table describes the options to **cdr define server**.

Long Form	Short Form	Meaning
--init	**-I**	Adds *server_group* to the replication domain.
--leaf	**-L**	Defines the server as a leaf server in an existing domain. The server that is specified by the **--sync** option becomes the parent of the leaf server.
--nonroot	**-N**	Defines the server as a nonroot server in an existing domain. The server that is specified by the **--sync** option becomes the parent of the nonroot server.
--sync=	**-S**	Adds a server to the existing domain of which the *sync_server* is a member. Uses the global catalog on *sync_server* as the template for the global catalog on the new replication server, *server_group*. For Hierarchical Routing topologies, Enterprise Replication also uses the *sync_server* as the parent of the new server in the current topology.

Dynamic Options

Use the dynamic options to modify the default behavior of **cdr define server**. You can change these options with the **cdr modify server** command while replication is active.

```
Options

|--+-----------------+--+----------------+--+---------------------------+--+-----------------+--|
   '- --ats=ats_dir -'  '- --ris=ris_dir-'  |                   .-text-. |  '- --idle=timeout-'
                                            '- --atsrisformat=--+-xml--+-'
                                                                '-both-'
```

Element	Purpose	Restrictions	Syntax
ats_dir	Name of the directory for Aborted Transaction Spooling files. The default is /tmp.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of /dev/null (UNIX) or NUL (Windows) prevents ATS file generation.	Follows naming conventions on your operating system
ris_dir	Name of the directory for Row Information Spooling files. The default is /tmp.	Must be a full path name. The path for the directory can be no longer than 256 characters. A value of /dev/null (UNIX) or NUL (Windows) prevents RIS file generation.	Follows naming conventions on your operating system
timeout	Idle timeout for this replication server.	Default value of 0 indicates no timeout. Must be an integer number of minutes. The maximum value is 32767.	Integer

The following table describes the options to **cdr define server** that you can change with the **cdr modify server** command while replication is active.

Long Form	Short Form	Meaning

Long Form	Short Form	Meaning
--ats=	**-A**	Specifies the directory to store aborted transaction spooling files for replicate transactions that fail to be applied.
--atsrisformat=	**-X**	Specifies the format of ATS and RIS files: • **text** indicates that ATS and RIS files are generated in standard text format. • **xml** indicates that ATS and RIS files are generated in XML format. • **both** indicates that ATS and RIS files are generated in both standard text format and XML format. If you omit the **--atsrisformat=** option, ATS and RIS files are created in text format.
--ris=	**-R**	Specifies the directory to store row information spooling files for replicate row data that fails conflict resolution or encounters replication-order problems.
--idle=	**-i**	The default value is `0`. Set the number of minutes after which an inactive connection is closed after *timeout* minutes. If timeout is `0`, the connection does not time out.

Usage

Run the **cdr define server** command on the database server that you want to define as a replication server. To create the replication server in an existing domain, specify a synchronization server that belongs to that domain with the **--sync=** option. To create a replication server in a new domain, omit the **--sync=** option. The **cdr define server** command creates the Enterprise Replication global catalog on the specified server.

If the CDR_QDATA_SBSPACE and CDR_DBSPACE configuration parameters are not set and the database server has a storage pool with sufficient space, the **cdr define server** command automatically performs the following tasks:

- Creates an sbspace and a dbspace from chunks from the storage pool. The spaces have the same characteristics and storage-pool behavior as other spaces created from the storage pool. The storage pool must have at least 500 MB of free space for the sbspace and 200 MB of free space for the dbspace. These spaces must be composed of chunks of size 100 MB or greater.
- Sets the values of the CDR_QDATA_SBSPACE and CDR_DBSPACE configuration parameters to the space names both in memory and in the onconfig file.
- Shows the names of the spaces that are created.

You can run this command from within an SQL statement by using the SQL administration API.

The maximum number of replication servers that you can define is 32767.

Examples

The following example defines the first database server in a replication domain. The command specifies the following actions:

- Connect to the database server **stan**.
- Initialize Enterprise Replication.
- Set the /cdr/ats directory for generated ATS files.
- Set the /cdr/ris directory for generated RIS files.
- Set the format of ATS and RIS files to text.

```
cdr define server --connect=stan \
--ats=/cdr/ats --ris=/cdr/ris \
--atsrisformat=text --init g_stan
```

The following example adds a database server to the replication domain that was created in the previous example. The command specifies the following actions:

- Connect to the database server **oliver**.
- Initialize Enterprise Replication.
- Synchronize the catalogs on database server **oliver** with the catalogs on database server **stan**.
- Set the /cdr/ats directory for generated ATS files.
- Specify that RIS files are not generated.
- Set the file format of ATS files to XML.

```
cdr define server -c oliver \
-A /cdr/ats -R /dev/null -X xml \
-S g_stan -I g_oliver
```

Related concepts:
Monitor and troubleshooting Enterprise Replication
Enterprise Replication Server administrator
Related tasks:
Enabling ATS and RIS File Generation
Disabling ATS and RIS File Generation
Defining Replication Servers
Customizing the Replication Server Definition
Related reference:
cdr connect server
cdr delete server
cdr disconnect server
cdr list server
cdr modify server
cdr resume server
cdr suspend server
cdr realize template

cdr define shardCollection

The **cdr define shardCollection** command creates a sharding definition for distributing a table or collection across multiple shard servers.

Syntax

Element	Description	Restrictions
collection	The name of the collection that is distributed across database servers.	Must be the name of an existing collection.
column	The name of a table column.	Must be the name of an existing column.
database	The name of the database that contains the table or collection that is distributed across database servers.	Must be the name of an existing database.
definition_name	The name of the sharding definition that is created.	
ER_group	The ER-group name of a database server that receives sharded data.	Must be the ER-group name of an existing database server.
expression	The WHERE-clause expression that is used to select rows or documents by shard key value.	
field	The name of a collection field.	Must be the name of an existing field.
REMAINDER	Specifies the database server that receives rows or documents with shard key value that is not selected by the other expressions.	
partitions	The number of hashing partitions to create on each shard server.	Must be a positive integer.
table	The name of the table that is distributed across database servers.	Must be the name of an existing table.
user	The owner of the table or collection that is distributed across database servers.	Must be the name of an existing user.

The following table describes the **cdr define shardCollection** parameters.

Long Form	Short Form	Description
--key=	**-k**	Defines the shard key on all database servers. Possible values are: • A column name • An expression All database servers in a shard cluster must use the same column or expression as the shard key.
--partitions=	**-p**	Specifies the number of hashing partitions to create on each shard server when the sharding strategy is consistent hashing. Default is 3. The more hashing partitions, the more evenly the data is distributed among shard servers. However, if you specify more than 10 hashing partitions, the resulting SQL statement to create the sharded table might fail because it exceeds the maximum character limit for an SQL statement.

Long Form	Short Form	Description
--strategy=	**-s**	Specifies the method for determining which database server an inserted row or document is distributed to. Possible values are: • **expression**: The expression that is defined in the server statement is used. • **chash**: A consistent hash algorithm is used. When you add or remove a shard server, the consistent hashing algorithm redistributes a fraction of the data. • **hash**: A hash algorithm is used. When you add or remove a shard server, the hashing algorithm redistributes all the data.
--type=	**-t**	Specifies action on the shard server where a row or document was inserted: • **delete** (default): The row or document is deleted from the source shard server after it is replicated to the target shard server. If you do not set `--versionCol=`*column*, changes made to rows and documents can be lost during the replication process. • **keep**: The row or document is not deleted on the source shard server after the row or document is replicated to the source shard serverso that two copies of the data exist in the shard cluster. • **informational**: Data is not replicated. You can run sharded queries but the data is not sharded during loading. You must load the data on the appropriate shard server according to the sharding definition. • **informational_noer**: You can use with shard edge servers without having to define Enterprise Replication at edge servers.
--versionCol=	**-v**	When `--type=delete` is specified in the sharding definition, Enterprise Replication must verify that a source row or document was not updated before it can delete the row or document on the shard server. Possible values are: • A table column name • A field name If `--type=delete` is set in the sharding definition, but `--versionCol=`*column* is not, changes made to rows and documents can be lost during the replication process. This parameter is required if any rows have out-of-row data, such as data stored in smart large object, or if collections have BSON documents that have sizes larger than 4 KB.

Usage

Use the **cdr define shardCollection** command to create a sharding definition for distributing a table or document across multiple shard servers. The replicates that are created as part of the **cdr define shard** command are mastered and use always apply and row scope. You cannot specify that triggers fire.

Multiple sharding definitions are not allowed on the same table or collection.

You cannot manually define an Enterprise Replication replicate for a table that is sharded.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 3, 18, 39, 52, 83, 99, 125, 196, 215, 229.

For information about these error codes, see Return Codes for the cdr Utility.

Example: Creating a sharding definition that uses a consistent hash algorithm

The following example creates a sharding definition that is named **collection_1**. Rows that are inserted on any of the shard servers are distributed, based on a consistent hash algorithm, to the appropriate shard server. Enterprise Replication must verify that a replicated row or document was not updated before the row or document can be deleted on the source server. The **b** column in the **customers** table that is owned by user **john** is the shard key. Each shard server has three hashing partitions.

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1 g_shard_server_2 g_shard_server_3
```

The partition range for each shard server is calculated based on the server group name. The data is distributed according to the following sharding definition:

```
g_shard_server_1     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 4019 and 5469)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5719 and 6123)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2113 and 2652)
g_shard_server_2     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 6124 and 7415)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5470 and 5718)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 7416 and 7873)
g_shard_server_3     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2653 and 3950)
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) >= 7874
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) < 2113
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 3951 and 40
```

Example: Creating a sharding definition that uses a hash algorithm

The following example creates a sharding definition that is named **collection_1**. Rows that are inserted on any of the shard servers are distributed, based on a hash algorithm, to the appropriate shard server. Enterprise Replication must verify that a replicated row or document was not updated before the row or document can be deleted on the source server. The **state** column in the **customers** table that is owned by user **john** is the shard key.

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=state --strategy=hash --versionCol=version
   g_shard_server_A g_shard_server_B g_shard_server_C g_shard_server_D
```

Example: Creating a sharding definition that uses an IN expression

The following example creates a sharding definition that is named **collection_2**. The **state** column in the **clients** table that is owned by user **joe** is the shard key. Rows that are inserted on any of the shard servers are distributed, based on the defined expression, to the appropriate shard server. Replication acknowledgment must verify that a replicated row or document was not updated before the row or document can be deleted on the source shard server.

```
cdr define shardCollection collection_2 db_2:joe.clients
--type=delete --key=state --strategy=expression --versionCol=version
   g_shard_server_A "IN ('TX','OK')"
   g_shard_server_B "IN ('NY','NJ')"
   g_shard_server_C "IN ('AL','GA')"
   g_shard_server_D  REMAINDER
```

In the previous example:

- Inserted rows that have a value of `AL` in the **state** column are sent to **g_shard_server_C**.
- Inserted rows that have a value of `NJ` in the **state** column are sent to **g_shard_server_B**.
- Inserted rows that have a value of `CA` in the **state** column are sent to **g_shard_server_D**.

Example: Creating a sharding definition that uses a BETWEEN expression

The following example creates a definition that is named **collection_3**. The **age** column in the **users** table that is owned by user **charles** is the shard key. Rows that are inserted on any of the shard servers are distributed, based on the defined expression, to the appropriate shard server. Replication acknowledgment must verify that a replicated row or document was not updated before the row or document can be deleted on the source shard server.

```
cdr define shardCollection collection_3 db_3:charles.users
--type=delete --key=age --strategy=expression --versionCol=version
   g_shard_server_A "BETWEEN 0 and 20"
   g_shard_server_B "BETWEEN 21 and 62"
   g_shard_server_C "BETWEEN 63 and 100"
   g_shard_server_D  REMAINDER
```

In the previous example:

- Inserted rows that have a value of `35` in the **age** column are sent to **g_shard_server_B**.
- Inserted rows that have a value of `102` in the **age** column are sent to **g_shard_server_D**.
- Inserted rows that have a value of `15` in the **age** column are sent to **g_shard_server_A**.

Example: Creating a sharding definition that defines a shard key by function

The following example creates a sharding definition that is named **collection_4**. The **COLOR** shard key in the **cars** collection that was owned by user **mike** is the shard key. Documents that are inserted on any of the shard servers are distributed, based on the defined expression, to the appropriate shard server.

```
cdr define shardCollection collection_4 db_4:mike.cars
   -t delete -k "bson_value_lvarchar(data,'COLOR')" -s expression -v version
   g_shard_server_E "IN ('blue','green')"
   g_shard_server_F "IN ('black','white')"
   g_shard_server_G "IN ('brown','gray')"
   g_shard_server_H "IN ('red','yellow')"
   g_shard_server_I  REMAINDER
```

In the previous example:

- Inserted documents that have a value of `yellow` in the **COLOR** key are sent to **g_shard_server_H**.
- Inserted documents that have a value of `blue` in the **COLOR** key are sent to **g_shard_server_E**.
- Inserted documents that have a value of `pink` in the **COLOR** key are sent to **g_shard_server_I**.

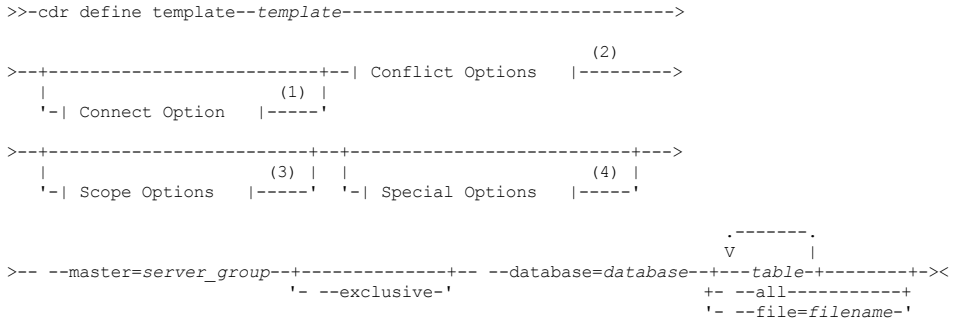
Related tasks:
Creating a shard cluster
Related reference:
cdr change shardCollection
cdr delete shardCollection
cdr list shardCollection
Related information:
Enabling sharding for JSON or relational data
Creating a shard cluster by running the addShard command in the MongoDB shell
Creating a shard cluster by running the addShard command through db.runCommand in the MongoDB shell
Creating a shard-cluster definition that uses a hash algorithm for distributing data across database servers
onstat -g shard command: Print information about the shard cache

cdr define template

The **cdr define template** command creates a template for replicates and a replicate set.

Because templates define replicates, many of the syntax options for the **cdr define template** command are the same as for the **cdr define replicate** command.

Syntax



Notes:

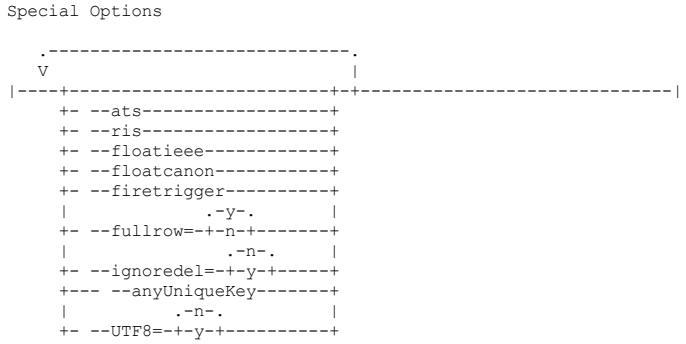
1. See Connect Option.
2. See Conflict Options.
3. See Scope Options.
4. See Special Options.

Element	Purpose	Restrictions	Syntax
template	Name of the template to create.	The template name must be unique and cannot be the same as a replicate or replicate set name.	Long Identifiers
database	Name of the database that is used to define the template attributes.	The database server must be registered with Enterprise Replication.	Long Identifiers
table	Name of the table to be included in the template.	The table must be an actual table. It cannot be a synonym or a view. For ANSI databases, you must specify *owner.tablename*.	Long Identifiers
filename	The directory and file name of the file that contains a list of tables to be included in the template.	Must be a full path name and file name. The path and file name can be no longer than 256 bytes. Within the file, the table names can be separated by a space or placed on different lines.	Follows naming conventions on your operating system.
server_group	Name of a database server group to declare for Enterprise Replication.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers

The following table describes the options to **cdr define template**.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all tables in the database are included in the template.
--database=	**-d**	Specifies which database the template is based on. If no tables or table list file name are listed after this option, then all tables in the database are included in the template.
--exclusive	**-X**	Creates an exclusive replicate set. The state of the replicate set is inactive until you apply the template. This option is required if you have referential integrity constraints on a table.
--file=	**-f**	Specifies the path and file name of a file that lists the tables to be included in the template. The file must contain only table names, either separated by spaces or each on its own line.
--master=	**-M**	Specifies the server that contains the database to be used as the basis of the template. If this option is not specified, then the server that is specified in the connect option is used.

Special Options



```
|                  .-n-. |
'- --alwaysRepLOBs=-+-y-+-'
```

The following table describes the special options to the **cdr define template** command.

Long Form	Short Form	Meaning
--alwaysRepLOBS=		Specifies whether columns that contain unchanged large objects are included in replicated rows: • **--alwaysRepLOBS=n**: Default. Columns that contain unchanged large objects are not replicated. • **--alwaysRepLOBS=y**: Columns that contain large objects are always included in replicated rows.
--anyUniqueKey	**-U**	Specifies that the replication key is detected automatically from the following sources in the follow order: • A primary key that is defined on the table • ERKEY shadow columns that are included in the table • Any unique key or unique constraint that is defined on the table The replicate must be a strictly mastered replicate. If the table includes ERKEY shadow columns, those columns are included in the participant definition only if the table does not have a primary key.
--ats	**-A**	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database.
--firetrigger	**-T**	Specifies that the rows that the replicate inserts fire triggers at the destination.
--floatieee	**-I**	Transfers replicated floating-point numbers in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format. Use this option for all new replicate definitions.
--floatcanon	**-F**	Transfers replicated floating-point numbers in machine-independent decimal representation. This format is portable, but can lose accuracy. This format is provided for compatibility with earlier versions only; use **--floatieee** for all new replicate definitions.
--fullrow=	**-f**	Specifies whether to replicate full rows or only the changed columns: • **--fullrow=y** = Default. Indicates to replicate the full row and to enable upserts. If you also specify **deletewins** as the conflict resolution rule, upserts are disabled. • **--fullrow=n** = Indicates to replicate only changed columns and disable upserts.
--ignoredel=	**-D**	Specifies whether to retain deleted rows on other nodes: • **--ignoredel=y** = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify **deletewins** as the conflict resolution rule. • **--ignoredel=n** = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--ris	**-R**	Activates row-information spooling for replicate row data that fails conflict resolution or encounters replication order problems.
--UTF8=	None	Specifies whether to enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets. • **--UTF8=y** Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data that is contained within opaque data types. You cannot use **--UTF8=y** for replicates that contain **TimeSeries** data types, user-defined data types, or DataBlade module data types. • **–UTF8=n** Indicates that code set conversion is ignored.

Usage

A template consists of schema information about a database, a group of tables, column attributes, and the replication keys that identify rows. A template defines a group of master replicates and a replicate set. Templates are an alternative to using the **cdr define replicate** and **cdr start replicate** commands for each table and manually combining the replicates into a replicate set by using the **cdr define replicateset** command.

The replicate set can be exclusive or non-exclusive. Specify that the replicate set is exclusive if you have referential constraints that are placed on the replicated columns. If you create an exclusive replicate set using a template, you do not stop the replicate set to add replicates. The **cdr define template** command performs this task automatically.

If your tables include the ERKEY shadow columns, they are automatically added to replicate definition when you define a template. The **--erkey** option is not needed with the **cdr define template** command.

You cannot specify an SPL routine for conflict resolution when you define a template.

After you define a template by running the **cdr define template** command, use the **cdr realize template** command to apply the template to your Enterprise Replication database servers.

You cannot update a template. To modify a template, you must delete it and then re-create it with the **cdr define template** command.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example illustrates the **cdr define template** command:

```
cdr define template tem1 -c detroit\
-C timestamp -S tran \
--master=chicago\
--database=new_cars table1 table2 table3
```

Line 1 of the example specifies a template name of **tem1** and that the connection is made to the server **detroit**. Line 2 specifies a conflict-resolution rule of **timestamp** and a transaction scope for conflict resolution. Line 3 specifies that the master replicate information is obtained from the server **chicago**. Line 4 specifies to use the **new_cars** database on the **chicago** server and to include only the tables **table1**, **table2**, and **table3**.
The next example is the same as the first except that it has additional options and uses a file instead of a list of tables:

```
cdr define template tem1 -c detroit\
-C timestamp -S tran --master=chicago\
--ignoredel=y\
--database=new_cars --file=tabfile.txt
```

Line 3 indicates that delete operations are not replicated. Retaining deleted rows on target servers is useful for consolidation models.
Line 4 specifies a file name for a file that contains a list of tables to include in the template. The tabfile.txt file has the following contents:

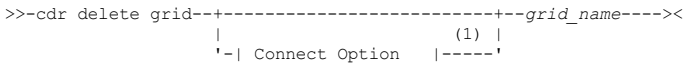
```
table1
table2
table3
table4
```

Related concepts:
Replicate types
Frequency Options
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related tasks:
Exclusive Replicate Sets
Preparing tables without primary keys
Creating replicated tables through a grid
Related reference:
cdr list template
cdr realize template
cdr delete template
cdr define replicate
Participant and participant modifier

cdr delete grid

The **cdr delete grid** command deletes the specified grid.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers

Usage

Use the **cdr enable grid** command to delete an existing grid.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 222.

For information about these error codes, see Return Codes for the cdr Utility.

Examples

The following example deletes the grid named **grid1**:

```
cdr delete grid grid1
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
ifx_grid_connect() procedure
cdr define grid
cdr check queue

cdr delete region

The **cdr delete region** command deletes a region from a grid.

Syntax

```
>>-cdr delete region--+--------------------------+-------------->
                     |                    (1)   |
                     '-| Connect Option    |-----'

>-- --region--=--region_name-----------------------------------><
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
region_name	Name of the region.	Must be the name of a defined region.	Long Identifiers

The following table describes the **cdr delete region** option.

Long Form	Short Form	Meaning
--region	**-r**	Specifies the region to delete.

Usage

Use the **cdr delete region** command to remove a region from a grid. Delete a region that is no longer valid, for example, if you remove a grid server that is included in the region from the grid. If you need to change the list of grid servers in a region, you delete the region and then re-create it with the new server list.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 227.

For information about these error codes, see Return Codes for the cdr Utility.

Examples

The following command deletes the region that is named **northwest**:

```
cdr delete region --region=northwest
```

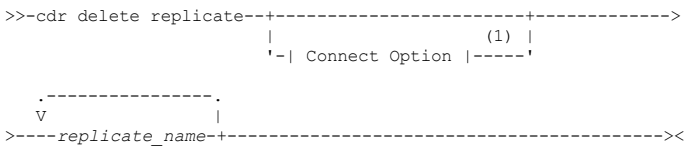
Related concepts:
Grid queries
Related tasks:
Defining tables for grid queries
Related reference:
ifx_grid_connect() procedure
cdr delete region
cdr change gridtable
cdr remaster gridtable
ifx_node_id() function
ifx_node_name() function
Related information:
SELECT_GRID session environment option

SELECT_GRID_ALL session environment option
GRID clause

cdr delete replicate

The **cdr delete replicate** command deletes a replicate.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
replicate_name	Name of the replicate to delete.		Long Identifiers

Usage

The **cdr delete replicate** command deletes the replicate *repl_name* from the global catalog. All replication data for the replicate is purged from the send queue at all participating database servers. You can run this command from within an SQL statement by using the SQL administration API.

Important: If you are creating a replication server to replace the one you deleted, use the **cdr check queue --qname=ctrlq** command to make sure that the delete operation propagated to all the servers.
When you run the **cdr delete replicate** command, an event alarm with a class ID of 68 is generated, if that event alarm is enabled.

Example 1: Deleting a single replicate

The following command connects to the default database server specified by the INFORMIXSERVER environment variable and deletes the replicate **reynolds**:

```
cdr delete replicate reynolds
```

Example 2: Deleting multiple replicates

The following command connects to database server **hoek** and deletes the replicates **reynolds** and **stimpson**:

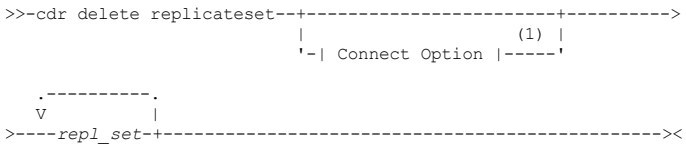
```
cdr del rep -c hoek reynolds stimpson
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Deleting a Replicate
Related reference:
cdr change replicate
cdr define replicate
cdr list replicate
cdr modify replicate
cdr resume replicate
cdr start replicate
cdr stop replicate
cdr suspend replicate
cdr check queue
Enterprise Replication Event Alarms

cdr delete replicateset

The **cdr delete replicateset** command deletes an exclusive or non-exclusive replicate set from the global catalog.

Syntax



Notes:

1. See Connect Option.

Element	Description	Restrictions	Syntax
repl_set	Name of replicate set to delete. Can be the name of a derived replicate set.		Long Identifiers

Usage

The **cdr delete replicateset** command deletes the exclusive or non-exclusive replicate set *repl_set* from the global catalog.

The **cdr delete replicateset** command does not affect the replicates or associated data. When a replicate set is deleted, the individual replicates within the replicate set are unchanged.

Attention: Do not delete time-based exclusive replicate sets. Doing so might result in inconsistent data.
Important: If you are creating a replicate set to replace the one you deleted, use the **cdr check queue -qname=ctrlq** command to make sure that the delete operation propagated to all the servers.
When you run the **cdr delete replicateset** command, an event alarm with a class ID of 69 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Example 1: Deleting a single replicate set

The following example connects to the database server specified by the INFORMIXSERVER environment variable and deletes the replicate set **accounts_set**:

```
cdr delete replset accounts_set
```

Example 2: Deleting multiple replicate sets

The following example connects to database server **hoek** and deletes the replicate sets **accounts1_set** and **accounts2_set**:

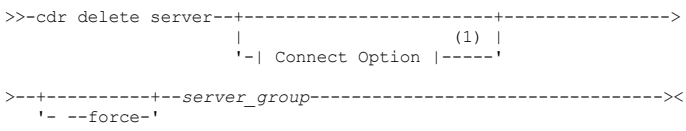
```
cdr del replset -c hoek accounts1_set accounts2_set
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Altering multiple tables in a replicate set
Deleting a Replicate Set
Related reference:
cdr change replicateset
cdr define replicateset
cdr define replicate
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset
cdr check queue
Enterprise Replication Event Alarms

cdr delete server

The **cdr delete server** disables a database server from participating in Enterprise Replication.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	A database server's `group` entry in its INFORMIXSQLHOSTS file and the global catalog.		Long Identifiers

The following table describes the option to the **cdr delete server** command.

Long Form	Short Form	Purpose
--force	**-f**	Remove an inactive Enterprise Replication server from the global catalog. You must use this option for standard servers that were converted from high-availability cluster servers.

Usage

The **cdr delete server** command disables a database server from to participating in Enterprise Replication. Use the **--force** option to disable an inactive replication server or to remove Enterprise Replication from a standard server that has been converted from a high-availability cluster server by the **oninit -d standard** command. You cannot delete a server that has non-root or leaf children under it. You must delete the children of a server before deleting the parent server.

The **cdr delete server** command generates event alarms with class IDs of 67 and 71, if the alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

To remove a server from an Enterprise Replication domain, run the **cdr delete server** command two times:

1. Run the command on the database server being removed, to disable it. This command does not propagate to other database servers in the domain.
2. Run the same command on a different server within the Enterprise Replication domain. This removes the disabled database server from the domain.

To remove an entire Enterprise Replication domain, run the **cdr delete server** command once on each replication server. The **cdr delete server** command performs the following tasks on each server:

1. Drops the Enterprise Replication connection to other hosts in the domain. A 25582 error and an operating system error might be printed to the online log.
2. Removes Enterprise Replication information, including delete tables and shadow columns.
3. Shuts down Enterprise Replication, if it is running.
4. Drops the local copy of the global catalog.

When you run the **cdr delete server** command on a different root server within an Enterprise Replication domain, the command performs the following tasks:

1. Deletes the command-specified database server from the global catalogs of all other servers in the domain.
2. Removes the command-specified database server from all participating replicates.
3. Purges all replication data destined for the command-specified database server from the send queues of all other servers in the domain.

Important: If you are creating a replication server to replace the one you deleted, use the **cdr check queue --qname=ctrlq** command to make sure that the delete operation propagated to all the servers.

Example 1: Removing a single database server from the domain

This example removes the database server **g_italy** from the Enterprise Replication environment. The commands are issued from **g_usa**:

```
cdr delete server -c italy g_italy
cdr delete server -c usa g_italy
```

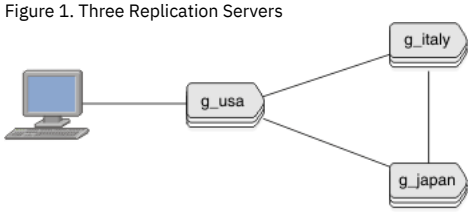
The first command connects to database server **g_italy** and disables Enterprise Replication by removing the **syscdr** database and removing or stopping other Enterprise Replication components.

The second command performs the following actions:

- Removes **g_italy** from the **g_usa** global catalog
- Drops the connection between **g_usa** and**g_italy**
- Removes **g_italy** from all participating replicates
- Purges the replication data destined for **g_italy** from send queues
- Broadcasts the delete command to all the other database servers in the Enterprise Replication domain so that the other servers can perform the same actions

Example 2: Removing the whole domain

The following illustration shows a replication environment with three replication servers, **g_usa**, **g_italy**, and **g_japan**.
Figure 1. Three Replication Servers



To remove Enterprise Replication from every server in the domain, issue the **cdr delete server** command while connecting to each server. For example, from the computer containing the **g_usa** replication server, run these commands to remove Enterprise Replication and eliminate the domain:

```
cdr delete server -c italy g_italy
cdr delete server -c japan g_japan
cdr delete server g_usa
```

Example 3: Removing Enterprise Replication from a high-availability server

In this example, the replication server group **g_usa** contains two servers that participate in a high-availability cluster: a primary (**usa_p**) and a secondary (**usa_s**). After **usa_s** is converted to a stand-alone server, the following command removes Enterprise Replication from it:

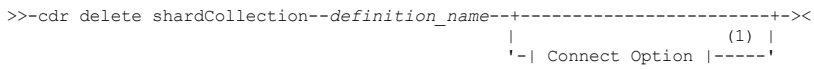
```
cdr delete server -c usa_s -f g_usa
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Failover for High-availability clusters in an Enterprise Replication environment
Deleting a Replication Server
Related reference:
cdr connect server
cdr define server
cdr disconnect server
cdr list server
cdr modify server
cdr resume server
cdr suspend server
cdr check queue
Enterprise Replication Event Alarms

cdr delete shardCollection

The **cdr delete shardCollection** command deletes a sharding definition, and then stops data sharding.

Syntax



Notes:

1. See Connect Option.

Element	Description	Restrictions
definition_name	The name of the sharding definition that is used for distributing data across multiple database servers.	Must be the name of an existing definition.

Usage

Use the **cdr delete shardCollection** command to delete a sharding definition, and then stop data sharding.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 99, 196, 229.

For information about these error codes, see Return Codes for the cdr Utility.

Example

For this example, you have a sharding definition that was created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers_1
   --type=delete --key=col2 --strategy=hash --versionCol=version
   shard_server_A
   shard_server_B
   shard_server_C
```

The following example deletes **collection_1**, and stops the sharding of table **customers_1**:

```
cdr delete shardCollection collection_1
```

Related concepts:
Shard cluster management and monitoring
Related reference:
cdr define shardCollection
cdr change shardCollection
cdr list shardCollection
Related information:

onstat -g shard command: Print information about the shard cache
Enabling sharding for JSON or relational data

cdr delete template

The **cdr delete template** command deletes a template from the replication domain. It also deletes any underlying replicate sets associated with the template (these will exist if the template has been realized). No replicates are deleted.

Syntax

```
>>-cdr delete template--+-------------------------+--template-><
                       |                    (1)  |
                       '-| Connect Option   |-----'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
template	Name of the template to delete.	The template must exist.	Long Identifiers

Usage

Use the **cdr delete template** command to delete the template definition and the replicate set realized from the template. Any replicates created by realizing the template to a database server are unaffected by this command.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following command deletes the template and replicate set **tem1**:

```
cdr delete template tem1
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr define template
cdr realize template
Enterprise Replication Event Alarms

cdr disable grid

The **cdr disable grid** command removes the authorization to run grid routines from users or servers.

Syntax

```
>>-cdr disable grid--+-------------------------+--------------->
                    |                    (1)  |
                    '-| Connect Option   |-----'

                        .---------------------------.
                        V                           |
>-- --grid--=--grid_name----+-----------------------+-+--------->
                            '- --user--=--user_name-'

   .------------------------------.
   V                              |
>----+--------------------------+-+----------------------------><
     '- --node--=--server_group-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group in the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
user_name	Name of the user.	Must be a user with authorization to run grid routines.	Long Identifiers

The following table describes the **cdr disable grid** options.

Long Form	Short Form	Meaning
--grid=	**-g**	Specifies the grid for which to revoke privileges.
--node=	**-n**	Specifies the servers on which to revoke privileges.
--user=	**-u**	Specifies the users to revoke privileges.

Usage

Use the **cdr disable grid** command to revoke the permission to run routines on the specified grid from the specified user or server that were granted by the **cdr enable grid** command.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility.

Examples

The following example revokes privileges from user **bill** on the **grid1** grid:

```
cdr disable grid --grid=grid1 --user=bill
```

The following example shows how to change the authorized server on the **grid1** grid from **gserv1** to **gserv2**:

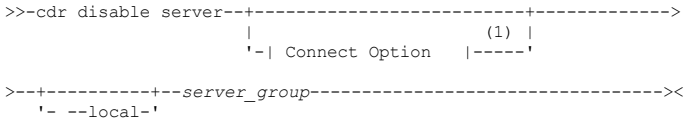
```
cdr disable grid --grid=grid1 --node=gserv1
cdr enable grid --grid=grid1 --node=gserv2
```

Related concepts:
Grid maintenance
Enterprise Replication Server administrator

cdr disable server

The **cdr disable server** command disables replication on a server.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server on which to disable replication.	Must be the name of an existing database server group in sqlhosts.	Long Identifiers

The following table describes the **cdr disable server** option.

Long Form	Short Form	Meaning
--local	**-l**	Disables the specified replication server. Must be run on both the replication server to disable and another replication server in the domain. Use this option if the connection is down between the replication server to disable and other replication servers.

Usage

Use the **cdr disable server** command when you need to temporarily stop replication and your replicates use the time stamp or delete wins conflict resolution rule.

When you run the **cdr disable server** command, the replication server is disabled and the rest of the replication domain is notified that the server is disabled.

If the replication server that you want to disable is not connected to the replication domain, you must run the **cdr disable server** command with the **--local** option on both the replication server to disable and another replication server in the domain. If the server on which you need to disable replication is currently offline, then run the **cdr disable server** command with the **--local** option on it after you restart it.

Disabling replication has the following effects:

- There is no connection between the disabled replication server and active replication servers.
- Transactions on the disabled replication server are not queued for replication
- Transactions on active replication servers are not queued for the disabled replication server.
- Control messages on active replication server are queued for the disabled replication server.
- Information about deleted rows on the disabled replication server is saved in delete tables.
- You can run only the following Enterprise Replication commands on the disabled replication server:
  - **cdr enable server**
  - **cdr stop server**
  - **cdr delete server**
  - **cdr check replicateset** with the **--repair** and the **--enable** options

You must synchronize the server after you enable replication on it. Shutting down and restarting the disabled replication server does not enable replication. You can both enable and synchronize a disabled replication server by running the **cdr check replicateset** command with the **--repair** and the **--enable** options. Alternatively, you can run the **cdr enable server** command and then synchronize the server.

Example 1: Stopping replication on a connected server

The following command disables the server, **g_cdr1**, which is connected to the replication domain:

```
cdr disable server -c g_cdr1 g_cdr1
```

Example 2: Stopping replication on a disconnected server

The following commands disable the replication server, **g_cdr1**, which is not connected to the replication domain:

```
cdr disable server -c g_cdr1 --local g_cdr1
cdr disable server -c g_cdr2 --local g_cdr1
```

The first command runs on the server **g_cdr1** and disables replication on it. The second command runs on the server **g_cdr2** and stops the other servers in the replication domain from queuing transactions for the server **g_cdr1**.

Related concepts:
Time stamp conflict resolution rule
Delete wins conflict resolution rule
Enterprise Replication Server administrator
Related tasks:
Temporarily stopping replication on a server
Related reference:
cdr enable server
cdr check replicateset

cdr disconnect server

The **cdr disconnect server** command stops a server connection.

Syntax

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server group to disconnect.	The database server group must be currently active in Enterprise Replication.	Long Identifiers

Usage

The **cdr disconnect server** command drops the connection (for example, for a dialup line) between *server_group* and the server specified in the **--connect** option. If the **--connect** option is omitted, the command drops the connection between *server_group* and the default database server (the one specified by the **INFORMIXSERVER** environment variable).

When you run the **cdr disconnect server** command, event alarms with class IDs of 54 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example drops the connection between the default database server (the one specified by the **INFORMIXSERVER** environment variable) and the server group **g_store1**:

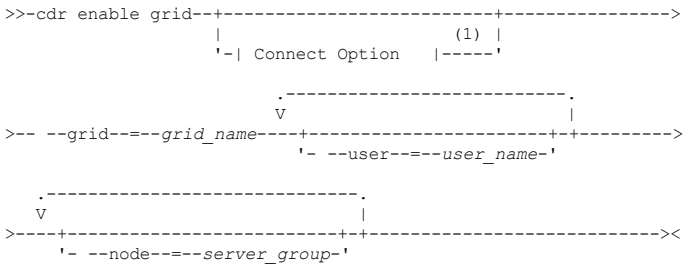
```
cdr disconnect server g_store1
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr connect server
cdr define server
cdr delete server
cdr list server
cdr modify server
cdr resume server
cdr suspend server

cdr enable grid

The **cdr enable grid** command authorizes users to run commands on the grid and designates servers from which grid commands can be run.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers
server_group	Name of a database server group in the grid.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
user_name	Name of the user.	Must have Connect privilege for databases on all the replication servers in the grid.	Long Identifiers

The following table describes the **cdr enable grid** options.

Long Form	Short Form	Meaning
--grid=	**-g**	Specifies the grid for which to provide privileges.
--node=	**-n**	Specifies the servers on which to provide privileges.
--user=	**-u**	Specifies the users to provide privileges.

Usage

Use the **cdr enable grid** command to control who can perform grid operations from which server in the grid. All the authorized users can run grid commands on all the authorized servers. The users must have Connect privilege for all databases on which they run grid routines on all the servers in the grid. You must authorize at least one user and one server to be able to run commands from the grid. User **informix** does not have permission to perform grid operations unless you include it in the user list.

Authorizing more than one server from which to run grid commands can lead to conflicts between grid commands.

After you initially enable a grid, you can add authorized users and servers by running the **cdr enable grid** command with the appropriate options.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility.

Examples

The following example authorizes the users **bill** and **tom** and the server **gserv1** to run grid routines on the grid named **grid1**:

```
cdr enable grid --grid=grid1 --user=bill --user=tom --node=gserv1
```

The following example adds the user **srini** to the list of authorized users for the **grid1** grid:

```
cdr enable grid --grid=grid1 --user=srini
```

The following example adds the server **gserv2** to the list of authorized servers for the **grid1** grid:

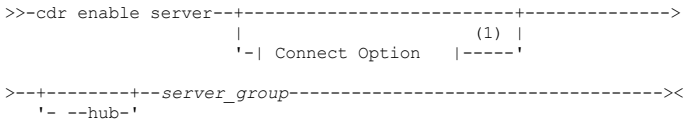
```
cdr enable grid --grid=grid1 --node=gserv2
```

Related concepts:
Grid maintenance
Example of setting up a replication system with a grid
Enterprise Replication Server administrator
Related tasks:
Creating a grid
Related reference:
ifx_grid_connect() procedure

cdr enable server

The **cdr enable server** command enables replication on a replication server that was disabled by the **cdr disable server** command.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server to enable.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers

The following table describes the **cdr enable server** option.

Long Form	Short Form	Meaning
--hub	**-h**	Specifies that when replication on a hub server is enabled, replication on all its child servers is also enabled.

Usage

Use the **cdr enable server** command when you are ready to restart replication on a disabled replication server. After you enable replication, you must synchronize the server with the rest of the replication domain. Before synchronization is complete, the replicates on the newly enabled replication server have the Pending Sync attribute. For replicates with the Pending Sync attribute, ATS and RIS files are not created if transactions are aborted on this server. You can see the Pending Sync attribute of a replicate in the OPTIONS field of the output of the **cdr list replicate** command.

Examples

The following command enables the disabled replication server, **g_cdr1**:

```
cdr enable server -c g_cdr1 g_cdr1
```

The following command enables the disabled replication server, **g_cdr1**, and its child servers:

```
cdr enable server -c g_cdr1 --hub g_cdr1
```

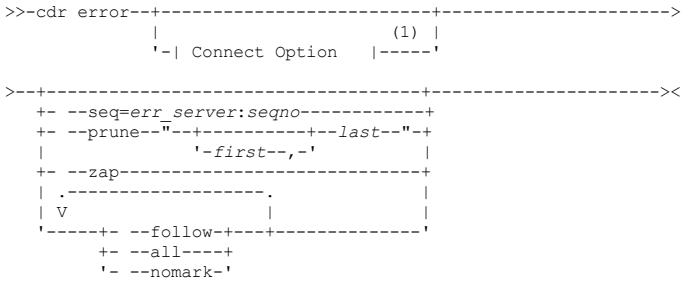
Related concepts:
Enterprise Replication Server administrator
Related tasks:

Restarting Replication on a Server
Related reference:
cdr disable server

cdr error

The **cdr error** command manages the **syscdrerror** table and provides convenient displays of errors.

Syntax



Notes:

1. See Connect Option.

Table 1. Elements for the cdr error command

Element	Purpose	Restrictions	Syntax
err_server	Name of database server group that holds the error table.	The server must be registered for Enterprise Replication.	Long Identifiers
first	Start date for a range.	You must provide a valid date and time.	Frequency Options
last	Ending date for range.	You must provide a later date and time than *first*.	Frequency Options
seqno	Sequence number of a specific error.	You must provide the number of an error in the error table.	Integer

Table 2. Options for the cdr error command

Long Form	Short Form	Meaning
(no options specified)		Print the current list of errors and then mark them as reviewed. Enterprise Replication does not display errors marked as reviewed.
--all	**-a**	Print all errors, including those already reviewed.
--follow	**-f**	Continuously monitor the error table.
--nomark	**-n**	Do not mark errors as reviewed.
--prune	**-p**	Prune the error table to those times in the range from *first* to *last*. If *first* is omitted, then all errors earlier than *last* are removed.
--seq	**-s**	Remove the (single) error specified by *server:seqno* from the error table.
--zap	**-z**	Remove all errors from the error table.

Usage

Run the **cdr error** command to examine replication errors. Sometimes a command succeeds on the server on which it is run but fails on one of the remote servers. For example, if you run the **cdr define replicate** command on **server1**, but the table name is misspelled on **server2**, the command succeeds on **server1** and seems to complete successfully. You can use **cdr error -c server2** to see why replication is failing.

The **cdr error** command also allows you to administer the **syscdrerror** table remotely. The **syscdrerror** table on each replication server contains errors for all replication servers, unless the replication server is a leaf node. The **syscdrerror** tables on leaf nodes do not contain errors for other replication servers. The reviewed flag indicates which errors are new errors while keeping the old errors in the table. For example, you can run **cdr error** periodically and append the output to a file.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following command shows the current list of errors on database server **hill**:

```
cdr error --connect=hill
```

After the errors are shown, Enterprise Replication marks the errors as reviewed.

The following command connects to the database server **lake** and removes from the error table all errors that occurred before the time when the command was issued:

```
cdr error -c lake --zap
```

The following command deletes all errors from the error table that occurred at or before 2:56 in the afternoon on May 1, 2008:

```
cdr error -p "2008-05-01 14:56:00"
```

The following command deletes all errors from the error table that occurred at or after noon on May 1, 2008 and before or at 2:56 in the afternoon on May 1, 2008:

```
cdr error -p "2008-05-01 14:56:00,2008-05-01 12:00:00"
```

Related concepts:
Enterprise Replication Server administrator

cdr finderr

The **cdr finderr** command looks up a specific Enterprise Replication return code and displays the corresponding error text.

Syntax

Element	Purpose	Restrictions
ER_return_code	Enterprise Replication return code to look up.	Must be a positive integer.

You can also view the Enterprise Replication return codes in the file **$INFORMIXDIR/incl/esql/cdrerr.h**.

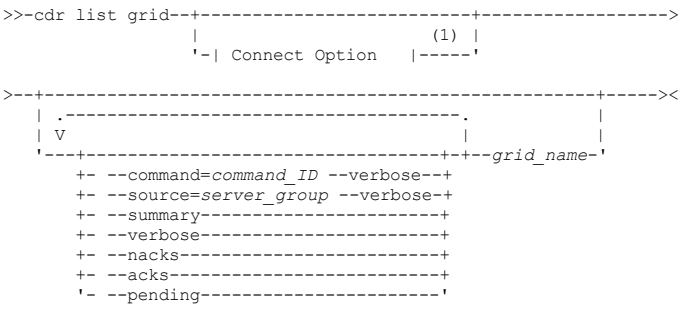
You can run this command from within an SQL statement by using the SQL administration API.

Related concepts:
Return Codes for the cdr Utility
Enterprise Replication Server administrator

cdr list grid

The **cdr list grid** command shows information about a grid.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
command_ID	The ID of a specific command that was run from the grid.		An integer.
grid_name	Name of the grid.	Must be the name of an existing grid.	Long Identifiers
server_group	Name of a database server group from which the command was run.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers

The following table describes the **cdr list grid** options.

Long Form	Short Form	Meaning

Long Form	Short Form	Meaning
--acks	**-a**	Displays the servers in the grid and the commands that succeeded on one or more servers.
--command=	**-C**	Displays the servers in the grid and the specified command.
--nacks	**-n**	Displays the servers in the grid and the commands that failed on one or more servers.
--pending	**-p**	Displays the servers in the grid and the commands that are in progress. A command can be pending because the transaction has not completed processing on the target server, the target server is down, or the target server was added to the grid after the command was run.
--source=	**-S**	Displays the servers in the grid and the commands that were run from the specified server.
--summary	**-s**	Displays the servers in the grid and the commands that were run on the grid.
--verbose	**-v**	Displays the servers in the grid, the commands that were run on the grid, and the results of the commands on each server in the grid.

Usage

Use the **cdr list grid** command to view information about servers in the grid, and about the commands that were run on servers in the grid.

If you run the **cdr list grid** command without any options or without a grid name, the output shows the list of grids.

Servers in the grid on which users are authorized to run grid commands are marked with an asterisk (*).

When you add a server to the grid, any commands that were previously run through the grid have a status of PENDING for that server. If you want to run previous grid commands on a new grid server, use the **ifx_grid_redo()** procedure. If you do not want to run previous grid commands on a new server, you can purge the commands by running the **ifx_grid_purge()** procedure.

When you run an SQL administration API command, the status of the grid command does not necessarily reflect whether the SQL administration API command succeeded. The grid command can have a status of ACK even if the SQL administration API command failed. The **cdr list grid** command shows the return codes of the SQL administration API commands. The **task()** function returns a message indicating whether the command succeeded. The **admin()** function returns an integer which if it is a positive number indicates that the command succeeded.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see Return Codes for the cdr Utility.

Examples

The examples in this section show the output of the **cdr list grid** command on a grid **grid1** that contains three servers: **cdr1**, **cdr2**, and **cdr3**.

Example 1: Display grid members

The following command displays the members of the **grid1** grid:

```
cdr list grid grid1
```

The output of the previous command is:

```
Grid                Node                User
------------------- ------------------- -----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
```

This output shows that the grid contains three member servers and that the authorized user **bill** can run grid routines from the server **cdr1**.

Example 2: Display verbose information about commands

The following command displays verbose information about a series of commands and their results on each server in the grid:

```
cdr list grid --verbose grid1
```

The output of the previous command is:

```
Grid                Node                User
------------------- ------------------- -----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create database tstdb with log
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3
```

```
Node:cdr1 Stmtid:2 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create table tab1 (col1 int, col2 int)
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3

Node:cdr1 Stmtid:3 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create procedure load(maxnum int)
define tnum int;
for tnum = 1 to maxnum
    insert into tab1 values (tnum, 1);
end for;
end procedure;
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3
```

This output shows each command and that all commands succeeded on servers **cdr1** and **cdr2** but are pending on the **cdr3** server because it is offline.

Example 3: Display errors

In this example, the **cdr3** server already has a database with the same name as the database in the CREATE DATABASE statement: therefore, the CREATE DATABASE and CREATE TABLE statements fail. The following command displays information about commands run within the grid that resulted in an error:

```
cdr list grid --nacks grid1
```

The output of the previous command is:

```
Grid                Node                User
------------------- ------------------- -----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create database tstdb with log
NACK cdr3 2010-05-27 15:39:21 SQLERR:-330 ISAMERR:-100

Node:cdr1 Stmtid:2 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create table tab1 (col1 int, col2 int)
NACK cdr3 2010-05-27 15:39:21 SQLERR:-310 ISAMERR:0
     Grid Apply Transaction Failure
```

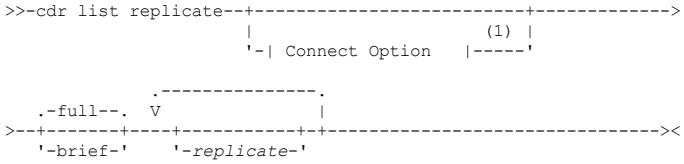
This output shows the SQL and ISAM error codes associated with the failed statements.

Related concepts:
Grid maintenance
Example of setting up a replication system with a grid
Enterprise Replication Server administrator
Related tasks:
Rerunning failed grid routines
Related reference:
cdr define grid
cdr change grid

cdr list replicate

The **cdr list replicate** command displays information about the replicates on the current server.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
replicate	Name or ID of the replicates.	The replicates must exist.	Long Identifiers

Usage

The **cdr list replicate** command displays information about replicates (the **full** option). If no replicates are named, the command lists all replicates on the current server. If one or more replicates are named, the command displays detailed information about those replicates.

To display only replicate names and participant information, use the **brief** option.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers obtain limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list replicate** is run on a leaf server, it displays incomplete information about the other database servers.

The **cdr list replicate** command can be used while the replication server is in DDRBLOCK mode. Before you use the **cdr list replicate** command, you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

Output Description

The `STATE` field can include the following values.

Table 1. Values of the STATE field

Value	Description
Active	Specifies that Enterprise Replication captures data from the logical log and transmits it to participants
Definition Failed	Indicates that the replication definition failed on a peer server
Inactive	Specifies that no database changes are captured, transmitted, or processed
Pending	Indicates that a **cdr delete replicate** command ran and the replicate is waiting for acknowledgment from the participants
Quiescent	Specifies that no database changes are captured for the replicate or participant
Suspended	Specifies that the replicate captures and accumulates database changes but does not transmit any of the captured data

The `CONFLICT` field can include the following values.

Table 2. Values of the CONFLICT field

Value	Description
Deletewins	Specifies that the replicate uses the delete wins conflict-resolution rule
Ignore	Specifies that the replicate uses the ignore conflict-resolution rule
Timestamp	Specifies that the replicate uses the time stamp conflict-resolution rule
Procedure	Specifies that the replicate uses an SPL routine as the conflict-resolution rule

The `FREQUENCY` field can include the following values.

Table 3. Values of the FREQUENCY field

Value	Description
immediate	Specifies that replication occurs immediately
every *hh*:*mm*	Specifies that replications occur at intervals (for example, 13:20 specifies every thirteen hours and 20 minutes)
at *day*.*hh*:*mm*	Specifies that replications occur at a particular time on a particular day (for example, 15.18:30 specifies on the 15th day of the month at 6:30 P.M.)

The `OPTIONS` field can include the following values.

Table 4. Values of the OPTIONS field

Value	Description
ats	Indicates that ATS files are generated if transactions fail to be applied at the target server.
firetrigger	Indicates that the rows that this replicate inserts fire triggers at the destination.
floatcanon	Indicates that floating-point numbers are replicated in machine-independent decimal representation.
floatieee	Indicates that floating-point numbers are replicated in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format.
fullrow	Indicates to replicate only changed columns and disable upserts.
ignoredel	Indicates that rows are retained if they are deleted on other nodes in the domain.
pendingsync	Indicates that the replication server was enabled with the **cdr enable server** command but that the participant is not yet synchronized with the rest of the domain. ATS and RIS files for this participant are not created if transactions are aborted.
ris	Indicates that RIS files are generated if transactions fail to be applied at the target server.
row	Indicates that the replicate uses row scope.
transaction	Indicates that the replicate uses transaction scope.
UTF8	Indicates that code set conversion between replicates is enabled.
TimeSeries	Indicates that the replicate includes a **TimeSeries** column.
alwaysRepLOBs	Indicates that large object columns are always included in replicated rows regardless of whether the large objects changed.

The `REPLTYPE` field can include the following values. If the `REPLTYPE` field does not show, the replicate is a classic replicate.

Table 5. Values of the REPLTYPE field

Value	Description
Master	Indicates that the replicate is defined as a master replicate.
Shadow	Indicates that the replicate is a shadow replicate. A shadow replicate can also be a master replicate.
Grid	Indicates that the replicate belongs to a grid replicate set.
Sendonly	Indicates that the participant only sends data.

The `PARENT REPLICATE` field shows only for shadow replicates. It shows the name of the replicate on which the shadow replicate is based.

Examples

The following example displays a list of the replicates on the current server with full details:

```
cdr list replicate
```

The output from the command shows two replicates:

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------
REPLICATE:         Repl1
STATE:             Inactive
CONFLICT:          Ignore
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       bank:joe.teller
OPTIONS:           row,ris,ats
REPLTYPE:          Master

REPLICATE:         Repl2
STATE:             Inactive
CONFLICT:          Deletewins
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       bank:joe.account
OPTIONS:           row,ris,ats
REPLTYPE:          Master,Shadow
PARENT REPLICATE:  Repl1
```

If the replicate belongs to a grid replicate set, the `REPLTYPE` field includes the value `Grid`.

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------
REPLICATE:         grid_6553604_100_3
STATE:             Active ON:g_delhi
CONFLICT:          Always Apply
FREQUENCY:         immediate
QUEUE SIZE:        0
PARTICIPANT:       tdb:nagaraju.t1
OPTIONS:           row,ris,fullrow
REPLID:            6553605 / 0x640005
REPLMODE:          PRIMARY  ON:g_delhi
APPLY-AS:          INFORMIX ON:g_delhi
REPLTYPE:          Master,Grid
```

The `PARENT REPLICATE` field only shows if the replicate is a shadow replicate.

The following example displays a list of the replicates on the current server with brief details:

```
cdr list replicate brief
```

The output from the command shows the replicates:

```
REPLICATE   TABLE                                SELECT
-----------------------------------------------------------------
Repl1       bank@g_newyork:joe.teller            select * from joe.teller
Repl1       bank@g_sanfrancisco:joe.teller       select * from joe.teller
Repl2       bank@g_portland:joe.teller           select * from joe.teller
Repl2       bank@g_atlanta:joe.teller            select * from joe.teller
```

The following example specifies the names of replicate:

```
cdr list repl brief Repl1
```

The output from the command shows information for the replicate:

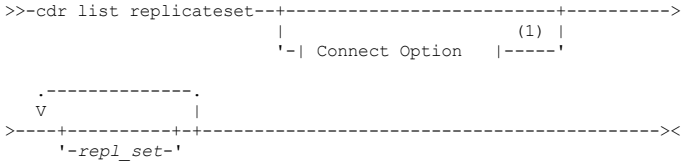
```
REPLICATE   TABLE                                SELECT
-----------------------------------------------------------------
Repl1       bank@g_newyork:joe.teller            select * from joe.teller
Repl1       bank@g_sanfrancisco:joe.teller       select * from joe.teller
```

Related tasks:
Adding an existing replicate to a grid replicate set by using cdr change replicateset
Viewing grid information
Preventing Memory Queues from Overflowing
Related reference:
cdr change replicate
cdr define replicate
cdr delete replicate
cdr modify replicate
cdr resume replicate

cdr list replicateset

The **cdr list replicateset** command displays information about the replication sets defined on the current server.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set.	The replicate set must exist.	Long Identifiers

Usage

The **cdr list replicateset** command displays a list of the replicate sets that are currently defined. To list the information about each of the replicates within the replicate set, use **cdr list replicateset** *repl_set*.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list replicateset** is executed against a leaf server, it displays incomplete information about the other database servers.

If you specify the name of a grid replicate set, the command displays the names of the replicates that were automatically created through the grid and any replicates manually added to the grid replicate set. The name of the grid replicate set is the same as the name of the grid.

The **cdr list replicateset** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list replicateset** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

Examples

The following example displays a list of the replicate sets on the current server:

```
cdr list replicateset
```

The following output might result from the previous command:

```
Ex T REPLSET                  PARTICIPANTS
------------------------------------------------
N  Y  g1                        Repl1, Repl4
N  Y  g2                        Repl2, Repl3, Repl5
```

The `Ex` field shows whether the replicate set is exclusive. The `T` field shows whether the replicate set was created from a template.

This example displays information for all the replicates in the replicate set **g1**:

```
cdr list replset g1
```

The following output might result from the previous command:

```
REPLICATE SET:g1 [Exclusive]
CURRENTLY DEFINED REPLICATES
------------------------------------------------------------------------
REPLICATE:      Repl1
STATE:          Inactive
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    bank:arthur.account
OPTIONS:        row,ris,ats
REPLTYPE:       Master

REPLICATE:      Repl4
STATE:          Inactive
```

```
CONFLICT:       Deletewins
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    bank:arthur.teller
OPTIONS:        row,ris,ats
REPLTYPE:       Master
```

The information supplied for each replicate is the same as the information provided by the **cdr list replicate** command.

Related concepts:
Example of setting up a replication system with a grid

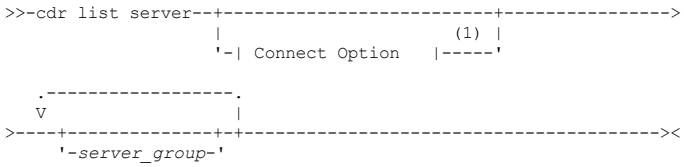
Related tasks:
Viewing grid information

Related reference:
cdr change replicateset
cdr define replicateset
cdr delete replicateset
cdr define replicate
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset
cdr list replicate

cdr list server

The **cdr list server** command displays a list of the Enterprise Replication servers that are visible to the server on which the command is run.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of the server group.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr list server** command displays information about servers. You do not need to be user **informix** to use this command; any user can run it.

The **cdr list server** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list server** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

When no server-group name is given, the **cdr list server** command lists all database server groups that are visible to the current replication server.

In hierarchical topology, leaf servers only have information about their parent database servers in the Enterprise Replication domain. Therefore, when **cdr list server** is executed against a leaf server, it displays incomplete information about the other database servers.

Output Description

The SERVER and ID columns display the name and unique identifier of the Enterprise Replication server group.

The STATE column can have the following values.

Value	Description
Active	The server is active and replicating data.
Deleted	The server has been deleted; it is not capturing or delivering data and the queues are being drained.
Disabled	The server is disabled. It is not capturing or delivering data, but its delete tables are being maintained.
Quiescent	The server is in the process of being defined.
Suspended	Delivery of replication data to the server is suspended.

The STATUS column can have the following values.

Value	Description
Connected	The connection is active.
Connecting	The connection is being established.
Disconnect	The connection was explicitly disconnected.
Disconnected will attempt reconnect	The connection was disconnected but is being reattempted.
Dropped	The connection was disconnected due to a network error because the server is unavailable.
Error	The connection was disconnected due to an error (check the log and contact customer support, if necessary).
Failed	The connection attempt failed.
Local	Identifies that this server is the local server as opposed to a remote server.
Timeout	The connection attempt has timed out, but will be reattempted.

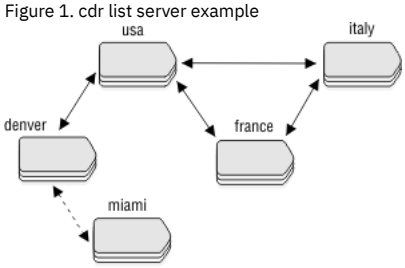
The QUEUE column displays the size of the queue for the server group.

The CONNECTION CHANGED column displays the most recent time that the status of the server connection was changed.

Examples

In the following examples, **usa**, **italy**, and **france** are root servers, **denver** is a nonroot server, and **miami** is a leaf server. The **usa** server is the parent of **denver**, and **denver** is the parent of **miami**.

Figure 1. cdr list server example



When the **cdr list server** command includes the name of a database server group, the output displays the attributes of that database server. The following commands and example output illustrate how the **cdr list server** command displays server information.

In this example, the server **g_usa** generates ATS and RIS files in XML format, has an idle time out of 15 seconds, and is a hub server.

```
cdr list server g_usa

NAME      ID      ATTRIBUTES
--------------------------------------------------
g_usa     1       atsrisformat=xml timeout=15 hub
```

In this example, the **g_denver** server shows the **g_usa** server as its root server.

```
cdr list server -c denver g_denver
NAME      ID      ATTRIBUTES
--------------------------------------------------
g_denver  27      root=g_usa
```

In this example, the attributes of the **g_denver** server are shown from the perspective of the **italy** server. The **g_denver** server has the **g_usa** server as its root server and uses the **g_usa** server to forward replicated transactions between it and the **italy** server.

```
cdr list server -c italy g_denver

NAME      ID      ATTRIBUTES
--------------------------------------------------
g_denver  27      root=g_usa forward=g_usa
```

In this example, the **g_miami** server shows the **g_denver** server as its root server and that it is a leaf server.

```
cdr list server g_miami

NAME      ID      ATTRIBUTES
--------------------------------------------------
g_miami   4       root=g_denver leaf
```

The following example shows possible output for the **cdr list server** command if no server groups are specified:

```
cdr list server
SERVER          ID STATE    STATUS      QUEUE CONNECTION CHANGED
------------------------------------------------------------------
g_denver         1 Active   Local       0
g_miami          2 Active   Connected   0     Mar 19 13:48:44
g_usa            3 Active   Connected   0     Mar 19 13:48:40
g_france         4 Active   Connected   0     Mar 19 13:48:41
g_italy          5 Active   Connected   0     Mar 19 13:48:45
```

Related tasks:
Preventing Memory Queues from Overflowing
Related reference:
cdr connect server

cdr define server
cdr delete server
cdr disconnect server
cdr modify server
cdr resume server
cdr start
cdr suspend server
cdr view
cdr resume replicate
cdr resume replicateset

cdr list shardCollection

The **cdr list shardCollection** command displays the sharding definition for all database servers in a shard cluster.

Syntax

```
>>-cdr list shardCollection--definition_name--+------------------------+-><
                                              |                   (1)  |
                                              '-| Connect Option |-----'
```

Notes:

1. See Connect Option.

Element	Description	Restrictions
definition_name	The name of the sharding definition that is used for distributing data across multiple database servers.	Must be the name of an existing definition.

Usage

The **cdr list shardCollection** command displays the sharding definition for database servers in a shard cluster.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 99, 196, 229.

For information on these error codes, see Return Codes for the cdr Utility.

Example: Output for a sharding definition that uses consistent hash-based sharding

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_1 db_1:john.customers
   --type=delete --key=b --strategy=chash --partitions=3 --versionCol=column_3
   g_shard_server_1 g_shard_server_2 g_shard_server_3
```

The following example shows output when the **cdr list shardCollection** command is run on a database server in the shard cluster. Each shard server has three hashing partitions.

Figure 1. Output when the **cdr list shardCollection** is run on a shard server that uses consistent hash-based sharding.

```
Shard Collection:shrdb Version:0 type:consistent hash key:b
Version Column:column_3
Table:db_1:john.customers
g_shard_server_1     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 4019 and 5469)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5719 and 6123)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2113 and 2652)
g_shard_server_2     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 6124 and 7415)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 5470 and 5718)
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 7416 and 7873)
g_shard_server_3     (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 2653 and 3950)
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) >= 7874
                      or mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) < 2113
                      or (mod(abs(ifx_checksum(b::LVARCHAR, 0)), 10000) between 3951 and 40
```

Example: Output for a sharding definition that uses hash-based sharding

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_1 database_1:john.customers_1
   --type=delete --key=col2 --strategy=hash --versionCol=column_3
   g_shard_server_A
   g_shard_server_B
```

```
   g_shard_server_C
   g_shard_server_D
```

The following example shows output when the **cdr list shardCollection** command is run on a database server in the shard cluster.

Figure 2. Output when the **cdr list shardCollection** is run on a shard server that uses hash-based sharding.

```
Shard Collection:collection_1 Version:0 type:hash key:col2
Version Column:column_3
Table:database_1:john.customers_1
g_shard_server_A      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  = 0
g_shard_server_B      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  in (1, -1)
g_shard_server_C      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  in (2, -2)
g_shard_server_D      mod(ifx_checksum(col2::LVARCHAR, 0), 4)  in (3, -3)
```

Example: Output for a sharding definition that uses an expression

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_2 database_2:joe.customers_2
   -t delete -k state -s expression -v column_3
   g_shard_server_F "IN ('AL','MS','GA')"
   g_shard_server_G "IN ('TX','OK','NM')"
   g_shard_server_H "IN ('NY','NJ')"
   g_shard_server_I  remainder
```

The following example shows output when the **cdr list shardCollection** command is run on a database server in the shard cluster.

Figure 3. Output when the **cdr list shardCollection** is run on a shard server that uses expression-based sharding.

```
Shard Collection:collection_2 Version:0 type:expression key:state
Version Column:column_3
Table:database_2:joe.customers_2
g_shard_server_F      state IN ('AL','MS','GA')
g_shard_server_G      state IN ('TX','OK','NM')
g_shard_server_H      state IN ('NY','NJ')
g_shard_server_I      not ( (state IN ('AL','MS','GA') ) or  (state
IN ('TX','OK','NM') ) or  (state IN ('NY','NJ') ))
```

Example: Output for a sharding definition that was modified

For this example, you have a sharding definition that is created by the following command:

```
cdr define shardCollection collection_3 database_3:tony.customers_3
   -t keep -k bson_value_lvarchar(data,'year') -s expression -v column_3
   g_shard_server_J "BETWEEN 1970 and 1979"
   g_shard_server_K "BETWEEN 1980 and 1989"
   g_shard_server_L "BETWEEN 1990 and 1999"
   g_shard_server_M  remainder
```

The sharding definition is then modified by the following command:

```
cdr change shardCollection collection_3 -a
   g_shard_server_N "BETWEEN 2000 and 2009"
```

The sharding definition is then modified a second time:

```
cdr change shardCollection collection_3 -d g_shard_server_J
```

The following example shows output when the **cdr list shardCollection** command is run on a database server in the shard cluster. The `Version` value increments with each **cdr change shardCollection** command that successfully runs on **collection_3**.

Figure 4. Output when the **cdr list shardCollection** is run on a shard server that has a modified sharding definition.

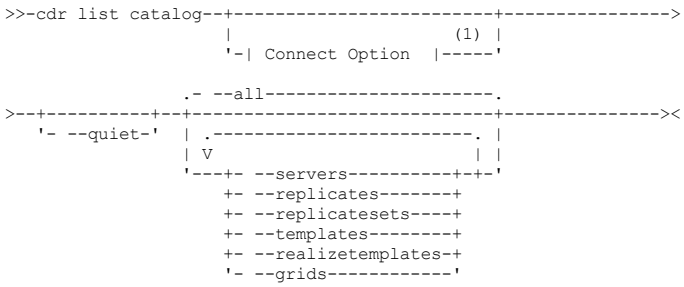
```
Shard Collection:collection_3 Version:2 type:expression
key:bson_value_lvarchar(data,'year') Version Column:column_3
Table:database_3:tony.customers_3
g_shard_server_K      bson_value_lvarchar(data,'year') BETWEEN 1980 and 1989
g_shard_server_L      bson_value_lvarchar(data,'year') BETWEEN 1990 and 1999
g_shard_server_N      bson_value_lvarchar(data,'year') BETWEEN 2000 and 2009
g_shard_server_M      not((bson_value_lvarchar(data,'year') BETWEEN 1980 and 1989)
or (bson_value_lvarchar(data,'year') BETWEEN 1990 and 1999) or (bson_value_lvarchar
(data,'year') BETWEEN 2000 and 2009))
```

Related concepts:
Shard cluster management and monitoring
Related reference:
cdr define shardCollection
cdr change shardCollection
cdr delete shardCollection
Related information:
onstat -g shard command: Print information about the shard cache
Enabling sharding for JSON or relational data
Viewing shard-cluster participants

cdr list catalog

The **cdr list catalog** command lists the commands that created the specified replication objects.

Syntax



Notes:

1. See Connect Option.

The following table describes the options to the **cdr list catalog** command.

Long Form	Short Form	Meaning
--all	**-a**	Lists all definition commands. Default.
--grids	**-g**	Lists **cdr create grid** commands.
--quiet	**-q**	Lists the commands without headings.
--realizetemplates	**-z**	Lists **cdr realize template** commands.
--replicates	**-r**	Lists **cdr define replicate** commands.
--replicatesets	-e	Lists **cdr define replicateset** commands.
--servers	**-s**	Lists **cdr define server** commands.
--templates	**-t**	Lists **cdr define template** commands.

Usage

Run the **cdr list catalog** command to show replication definition commands. You can use the list of commands to easily duplicate a system for troubleshooting or moving a test system into production.

Example: List template commands

The following command lists the **cdr define template** and the **cdr realize template** commands:

```
$ cdr list cat -t

#
# cdr define template commands.
#

cdr define template temp1 --conflict=ignore  --master=g_cat_cdr1
         --database=catdb informix.tab1 informix.tab2 informix.tab4

cdr define template temp2 --conflict=always  --master=g_cat_cdr1
         --database=catdb informix.tab1 informix.tab2 informix.tab4

cdr define template temp3 --conflict=timestamp  --master=g_cat_cdr1
         --database=catdb informix.tab1 informix.tab2 informix.tab4

cdr define template temp4 --conflict=timestamp --floatieee --ats --ris
        --alwaysRepLOBs=y --UTF8=y --master=g_cat_cdr1 --database=catdb
        informix.tab1 informix.tab2 informix.tab4
```

Example: List server commands

The following command lists the **cdr define server** commands:

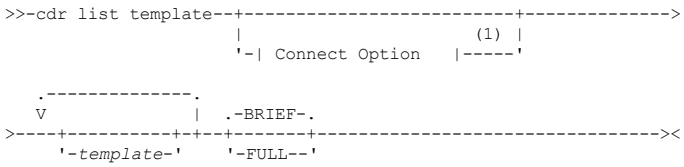
```
$ cdr list cat -s

#
# cdr define server commands.
#

cdr define server --init g_cat_cdr1

cdr define server --connect=g_cat_cdr2 --sync=g_cat_cdr1 --init g_cat_cdr2

cdr define server --connect=g_cat_cdr3 --ats=/usr/informix/ats
       --ris=/usr/informix/ris --sync=g_cat_cdr1 --init g_cat_cdr3
```

```
cdr define server --connect=g_cat_cdr4 --sync=g_cat_cdr1
       --init g_cat_cdr4
```


cdr list template

The **cdr list template** command displays information about the templates on the server on which the command is run.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
template	Name of the template.	The template must exist.	Long Identifiers

Usage

The **cdr list template** command displays information about templates. If no templates are named, the command lists all templates in the Enterprise Replication domain. If one or more templates are named, the command displays the names, database names, and table names for those templates.

To display detailed information for your templates, use the **FULL** option.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list template** is executed against a leaf server, it displays incomplete information about the other database servers.

The **cdr list template** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list template** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the onspaces utility.

Examples

The following example displays detailed information about the templates on the current server:

```
cdr list template
```

The output from the previous command might be the following:

```
TEMPLATE            DATABASE            TABLES
============================================
tem1                newcars             table1
                    newcars             table2
                    newcars             table3
tem2                carparts            table1
                    carparts            table3
```

The following example displays detailed information about the template **tem1**:

```
cdr list template tem1
```

The output from the previous command might be the following:

```
CURRENTLY DEFINED TEMPLATES
===========================
TEMPLATE:     tem1
TEMPLATE ID: 6553605
SERVER:       utah
DATABASE:     newcars
REPLICATE:    tem1_utah_2_1_table1
OWNER:        pravin
TABLE:        table1

TEMPLATE:     tem1
TEMPLATE ID: 6553605
SERVER:       utah
DATABASE:     newcars
REPLICATE:    tem1_utah_2_2_table2
OWNER:        pravin
TABLE:        table2
```

```
TEMPLATE:     tem1
TEMPLATE ID: 6553605
SERVER:       utah
DATABASE:     newcars
REPLICATE:    tem1_utah_2_3_table3
OWNER:        pravin
TABLE:        table3
```

Related reference:
cdr define template
cdr realize template

cdr migrate server

The **cdr migrate server** command automates data migration task between two or more servers. This command also automates setting up of Enterprise Replication between two Informix server instances.

Data migration options

- Static mode-offline data migration
  - Create storage spaces using storage pool
  - Migrate schema (except referential integrity) and data in parallel
  - Build referential integrity
- Dynamic mode – online data migration
  - Create storage spaces using storage pool
  - Migrate schema (except referential integrity) and data in parallel
  - Resynchronize data using Enterprise Replication
  - Build referential integrity

The **cdr migrate server** command automates tasks such as:

- Define Enterprise Replication domain between the two servers.
- Add ERKEY columns for the tables that does not have primary key or unique index columns.
- Create required storage spaces for the databases
- Apply database schema from source server to target server
- Create replicate definitions
- Synchronize data between source and target server instances
- Build referential integrity

Supported phases

1. define_er: Define ER
2.
   a. add_erkey: Add ERKEY to source schema
   b. add_replcheck : Add REPLCHECK column and index to source schema
3. create_spaces: Create storage spaces using storagepool
4.
   a. create_schema_loaddata: Multi phase replay of schema and data
      - Phased migration of schema and data:
        - Phase 1:
          - Migrate database schema except indexes, primary key, unique and referential constraints
          - Create all tables as “RAW” tables
        - Phase 2:
          - Start multiple parallel jobs for data load and index builds
          - Each job includes: make sure table is a raw table, Load data using “insert into ... select * from ...” ISTAR query, Build indexes, primary key and unique key constraints
        - Phase 3:
          - Build referential constraints

   b. create_schema_nodata : Create database schema
   c. create_schema_loaddata_nori : Create database schema without referential integrity and load data
5. create_replicates: Create replicate, replicate set and grid definitions
6. sync_data: Synchronize data
7. add_ri: Add referential integrity
8. all: Execute all phases (with phase create_schema_loaddata_nori)
9. static: Execute create_spaces and create_schema_loaddata phases
10. dynamic: Execute all phases similar to ‘all’

If you decide to execute the above phases individually, then phase order need to be maintained.

Prerequisites to run cdr migrate server command

1. To auto create required storage spaces, storage pool is a requirement at source and target servers.
2. SQLHOSTS files at both source and target server must be already configured with ER group information.
3. Trusted host configuration must be already established between source and target servers.
4. Source server must be 11.70xC1 or higher version.

Restrictions

1. Cannot mix multiple database code-sets in same data migration command.
2. Parallel data load using ISTAR query do not support tables with smart large objects(BLOB and CLOB), user created UDTs and collection datatypes.
   - Requires customization to unload/load data using external tables or using any other supported unload/load commands.
3. Tenant databases and database sharding are not supported.
4. Cannot use this tool for database code-set migration

Command syntax

```
cdr migrate server -s source -t target -p phase [-d database] [--exec]
```

-s --source Source server name

-t --target Target server name

-p --phase migration phase

-e --exec Execute commands. Default: print only.

-d --database Database to replicate

-T --receiveonly Setup oneway replication from source to target

('create_replicates' phase)

-r --checkrepair Synchronize data using 'cdr check'

instead of 'cdr sync'('sync_data' phase)

-g --grid Y/N Enable/Disable grid. Default:Y('create_replicates' phase)

-A --atsdir ATS files directory path('define_er' phase)

-R --risdir ATS files directory path('define_er' phase)

-P --parallelsync Number of table sync jobs to run

in parallel('sync_data' phase)

-f --outfile <file path> Output file for commands

-E --exclude db:owner.tab specify table to exclude

Examples

To print commands to stdout:

```
cdr migrate server   -s cdr_utm_nag_1 -t cdr_utm_nag_2   --phase all
```

To execute commands:

```
cdr migrate server   -s cdr_utm_nag_1 -t cdr_utm_nag_2   --phase all  --exec
```

To run 'define_er' phase:

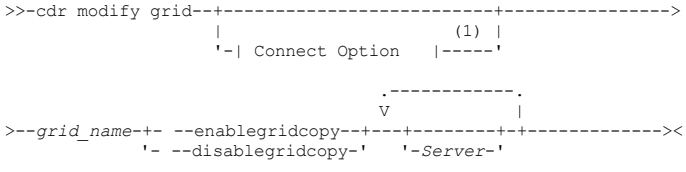
```
cdr migrate server  -s cdr_utm_nag_1 -t cdr_utm_nag_2   --phase define_er --atsdir=/work --risdir=/work
```

---

cdr modify grid

The **cdr modify grid** command modifies grid attributes.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax

Element	Purpose	Restrictions	Syntax
grid_name	Name of a grid to modify		
Server	Name of a server to modify		

The following table describes the options to **cdr modify grid**.

Long Form	Short Form	Meaning
--enablegridcopy	**-E**	Enables the specified server to perform grid copy functions.
--disablegridcopy	**-D**	Disables the specified server from performing grid copy functions.

Usage

The **cdr modify grid** command modifies the attributes of one or more servers in a grid. If the command does not specify a server, the changes apply to all servers in the grid.

The **--enablegridcopy** option is used only if a grid was created using Informix® version 11.70 and then upgraded to Informix version 12.10 or later.

Grids created using Informix version 11.70 and earlier cannot copy external files to a grid. If you upgrade servers in a grid from 11.70 to 12.10, and you want to copy external files to servers in the grid, you must enable the ability to copy external files by running the **cdr modify grid** command with the **--enablegridcopy** option. Similarly, before reverting from Informix version 12.10 to an earlier version of Informix, you must disable the ability to copy external files by running the **cdr modify grid** command with the **--disablegridcopy** option.

It is not necessary to run the **cdr modify grid** command if your grid was created using Informix version 12.10 or later.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example enables copying external files on all servers in the grid named **grid1**:

```
cdr modify grid  grid1 --enablegridcopy
```

The following example enables copying external files on the server **g_serv1** in the grid named **grid1**:

```
cdr modify grid  grid1 --enablegridcopy g_serv1
```

The following example disables copying external files on all servers in the grad named **grid1**:

```
cdr modify grid  grid1  --disablegridcopy
```

The following example disables copying external files for the server **g_serv1** in the grid named **grid1**:

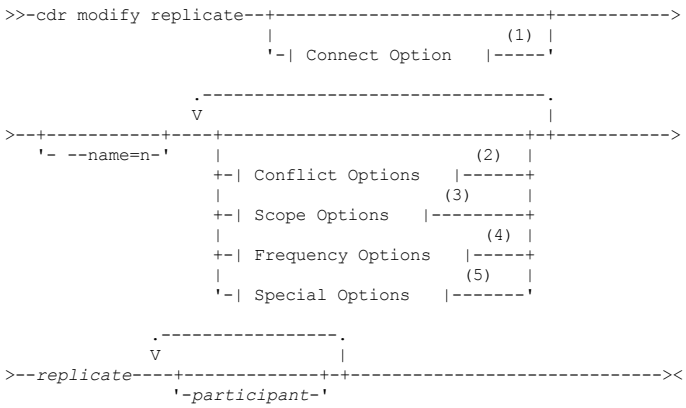
```
cdr modify grid  grid1  --disablegridcopy g_serv1
```

Related concepts:
Enterprise Replication Server administrator

cdr modify replicate

The **cdr modify replicate** command modifies replicate attributes.

Syntax



Notes:

1. See Connect Option.
2. See Conflict Options.
3. See Scope Options.
4. See Frequency Options.
5. See Special Options.

Element	Purpose	Restrictions	Syntax
participant	Name of a participant in the replication.	The participant must be a member of the replicate.	Participant and participant modifier
replicate	Name of the replicate to modify.	The replicate name must exist.	Long Identifiers

The following table describes the option to **cdr modify replicate**.

Long Form	Short Form	Meaning
--name=n	**-n n**	Removes the name verification attribute from a master replicate.

Special Options

```
Special Options

   .-----------------------------.
   V                             |
|----+-------------------------+-+-----------------------------|
     +- --ats--+-y-+-----------+
     |         '-n-'           |
     +- --ris--+-y-+-----------+
     |         '-n-'           |
     +- --firetrigger--+-y-+---+
     |                 '-n-'   |
     +- --fullrow--+-y-+-------+
     |             '-n-'       |
     |               .-n-.     |
     +- --ignoredel=-+-y-+-----+
     +- --serial---------------+
     |          .-n-.          |
     +- --UTF8=-+-y-+----------+
     |                   .-n-. |
     '- --alwaysRepLOBs=-+-y-+-'
```

Table 1. Special options for **cdr modify replicate**.

Long Form	Short Form	Meaning
--alwaysRepLOBS=		Specifies whether columns that contain unchanged large objects are included in replicated rows: • **--alwaysRepLOBS=n**: Default. Columns that contain unchanged large objects are not replicated. • **--alwaysRepLOBS=y**: Columns that contain large objects are always included in replicated rows.
--ats y or **--ats n**	**-A y** or **-A n**	Activates (**y**) or deactivates (**n**) aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
--firetrigger y or **--firetrigger n**	**-T y** or **-T n**	Causes the rows that are inserted by the replicate to fire (**y**) or not fire (**n**) triggers at the destination.
--fullrow y or **--fullrow n**	**-f y** or **-f n**	Specifies to (**y**) replicate the full row and enable upserts or (**n**) replicate only changed columns and disable upserts.
--ignoredel=	**-D**	Specifies whether to retain deleted rows on other nodes: • **--ignoredel=y** = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify **deletewins** as the conflict resolution rule. • **--ignoredel=n** = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--ris y or **--ris n**	**-R y** or **-R n**	Activates (**y**) or deactivates (**n**) row-information spooling for replicate row data that fails conflict resolution or encounters replication-order problems.
--serial	**-s**	Specifies that replicated transactions for the replicate are applied serially instead of in parallel.
--UTF8=		Specifies whether to enable conversion to and from UTF-8 (Unicode) when you replicate data between servers that use different code sets. • **--UTF8=y** Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data that is contained within opaque data types. You cannot use **--UTF8=y** for replicates that contain **TimeSeries** data types, user-defined data types, or DataBlade module data types. • **–UTF8=n** Indicates that code set conversion is ignored.

Usage

The **cdr modify replicate** command modifies the attributes of a replicate or of one or more participants in the replicate. You can also change the mode of a participant. If the command does not specify participants, the changes apply to all participants in the replicate.

To add or delete a participant, use the **cdr change replicate** command.

If you change the conflict resolution rule with **cdr modify replicate**, you must also specify the scope with the **--scope** option, even if you are not changing the scope.

The attributes for **cdr modify replicate** are the same as the attributes for **cdr define replicate**, with the following exceptions:

- You cannot change the machine-independent decimal representation (**--floatcanon**) or IEEE floating point (**--floatieee**) formats.
- You cannot change the conflict resolution from ignore to a non-ignore option (time stamp, SPL routine, or time stamp and SPL routine). You cannot change a non-ignore conflict resolution option to ignore.
  However, you can change from time stamp resolution to SPL routine resolution or from SPL routine resolution to time stamp.
- The **--ats**, **--ris**, **--firetrigger**, and **--fullrow** options require a yes (**y**) or no (**n**) argument.

When you run the **cdr modify replicate** command, an event alarm with a class ID of 63 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example modifies the frequency attributes of replicate **smile** to replicate every five hours:

```
cdr modify repl --every=300 smile
```

The following example modifies the frequency attributes of replicate **smile** to replicate daily at 1:00 A.M.:

```
cdr modify repl -a 01:00 smile
```

The following example modifies the frequency attributes of replicate **smile** to replicate on the last day of every month at 5:00 A.M., to generate ATS files, and not to fire triggers:

```
cdr modify repl -a L.5:00 -A y -T n smile
```

The following example changes the mode of the first participant that is listed to receive-only and the mode of the second to primary:

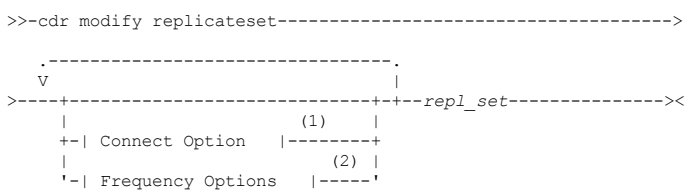
```
cdr mod repl smile "R db1@server1:antonio.table1" \
                   "P db2@server2:carlo.table2"
```

Related concepts:
Frequency Options
Enterprise Replication Server administrator
Change replicate attributes
Related tasks:
Enabling ATS and RIS File Generation
Creating Strict Master Replicates
Related reference:
cdr change replicate
cdr define replicate
cdr delete replicate
cdr list replicate
cdr resume replicate
cdr start replicate
cdr stop replicate
cdr suspend replicate
Enterprise Replication Event Alarms
Participant and participant modifier

cdr modify replicateset

The **cdr modify replicateset** command modifies all the replicates in a replicate set.

Syntax



Notes:

1. See Connect Option.
2. See Frequency Options.

Element	Purpose	Restrictions	Syntax

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to modify.	The replicate set must exist.	Long Identifiers

Usage

The **cdr modify replicateset** command modifies the attributes of all the replicates in the replicate set *repl_set*. To add or delete replicates from a replicate set, use the **cdr change replicateset** command.

You cannot change whether a replicate set is exclusive or not.

When you run the **cdr modify replicateset** command, an event alarm with a class ID of 64 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server (the server specified by the **INFORMIXSERVER** environment variable) and modifies the replicate set **sales_set** to process replication data every hour:

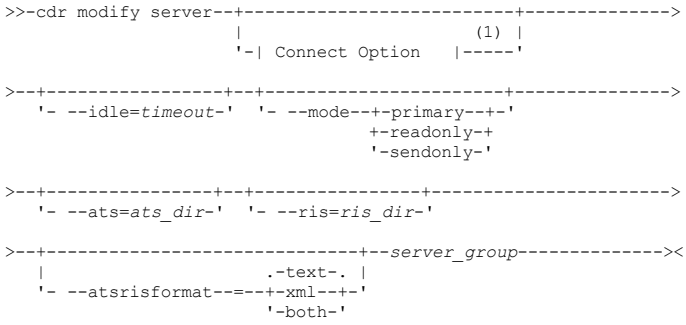
```
cdr mod replset --every=60 sales_set
```

Related concepts:
Frequency Options
Enterprise Replication Server administrator
Related reference:
cdr change replicateset
cdr define replicateset
cdr delete replicateset
cdr list replicateset
cdr define replicate
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset
Enterprise Replication Event Alarms

cdr modify server

The **cdr modify server** command modifies the Enterprise Replication attributes of a database server.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to modify.	The database server group must be defined in Enterprise Replication.	
timeout	Idle timeout for this server.	Must be an integer number of minutes. 0 indicates no timeout. The maximum value is 32,767.	Integer.
ats_dir	Name of Aborted Transaction Spooling directory.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of /dev/null (UNIX) or NUL (Windows) prevents ATS file generation.	Follows naming conventions on your operating system.

Element	Purpose	Restrictions	Syntax
ris_dir	Name of the Row Information Spooling directory.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of /dev/null (UNIX) or NUL (Windows) prevents RIS file generation.	Follows naming conventions on your operating system.

The following table describes the options to **cdr modify server**.

Long Form	Short Form	Meaning
--ats=	**-A**	Activates aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
–atsrisformat=	**-X**	Specifies the format of ATS and RIS files: • **text**: ATS and RIS files are generated in standard text format. • **xml**: ATS and RIS files are generated in XML format. • **both**: ATS and RIS files are generated in both standard text format and XML format.
--idle=	**-i**	Causes an inactive connection to be terminated after *timeout* minutes. If time-out is 0, the connection does not time out. The default value is 0.
--mode	**-m**	Changes the mode of all replicates using this server: • **primary**: The participant both receives and sends replicated data. • **readonly**: The participant only receives replicated data and does not send replicated data. • **sendonly**: The participant only sends replicated data and does not receive replicated data. Note: The **-m** option only affects replicates whose conflict resolution is ignore.
--ris=	**-R**	Activates row-information spooling for replicate-row data that fails conflict resolution or encounters replication-order problems.

Usage

The **cdr modify server** command modifies the replication server *server_group*.

When you run the **cdr modify server** command, an event alarm with a class ID of 70 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the database server **paris** and modifies the idle time-out of server group **g_rome** to 10 minutes. ATS files are generated into the directory /cdr/atsdir in both text and XML format.

```
cdr modify server -c paris -i 10 -A /cdr/atsdir \
-X both g_rome
```

The following example connects to the default database server and sets the modes of all participants on **g_geometrix** to primary:

```
cdr mod ser -m p g_geometrix
```

Related concepts:
Monitor and troubleshooting Enterprise Replication
Enterprise Replication Server administrator
Data consolidation
Primary-Target Data Dissemination
Modify server attributes
Related tasks:
Enabling ATS and RIS File Generation
Disabling ATS and RIS File Generation
Customizing the Replication Server Definition
Related reference:
cdr connect server
cdr define server
cdr delete server
cdr disconnect server
cdr list server
cdr resume server
cdr suspend server
Enterprise Replication Event Alarms

cdr realize template

The **cdr realize template** command creates the replicates, replicate set, and participant tables as specified in a template, and then synchronizes data on all or a subset of the database servers within the replication domain.

Syntax

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
database	Name of the database that includes the table to be replicated.	The database server must be in an Enterprise Replication domain.	Long Identifiers
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be in an Enterprise Replication domain.	
dbspace	The name of the dbspace for tables.	The dbspace must exist on all the database servers listed. If you do not specify a dbspace name and new tables are created, they are created in the default dbspace.	
server_group	Name of the database server group that includes the server to connect to.	The database server group name must be the name of an existing Enterprise Replication server group in sqlhosts.	Long Identifiers
*size***K** or *size* **M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	
template	The name of the template.	The template must exist. Use the **cdr define template** command to create the template.	Long Identifiers

The following table describes the special options to **cdr realize template**.

Long Form	Short Form	Meaning
--applyasowner	**-o**	Specifies that any tables created when you realize the template are owned by the owner of the source tables. By default, the tables are owned by the user **informix**.
--autocreate	**-u**	Specifies that if the tables in the template definition do not exist in the databases on the target servers, they are created automatically. However, the tables cannot contain columns with user-defined data types. Note: Tables that are created with autocreate do not automatically include non-replicate key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. You must manually create these objects.
--dbspace=	**-D**	Specifies the dbspace in which the automatically created objects are placed. If not specified, then the default dbspace is used.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows that are found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--mode=	**-m**	Specifies whether the participant either only sends or only receives replicated data: • **send_only** (**S**): The participant only sends replicated data and does not receive replicated data. • **receive_only** (**R**): The participant only receives replicated data and does not send replicated data.
--syncdatasource=	**-S**	Specifies which server is the source of the data that is used to synchronize all the other servers that are listed in the **cdr realize template** command. The server that is listed with this option must either be listed as one of the servers on which to realize the template, or it must already have the template.
--target	**-t**	Specifies that all of the servers that are listed in the command become receive-only servers, including the source server, unless the template is already realized on the source server. If you use this option, you must run the **cdr realize template** command twice: once to realize the template on the source server and other primary servers, and again to realize the template on receive-only servers.
--verify	**-v**	Specifies that the **cdr realize template** command verifies that the database, tables, column data types are correct on all listed servers, but does not realize the template.

Usage

Before you can use the **cdr realize template** command, you must define Enterprise Replication servers by running the **cdr define server** command and define the template by running the **cdr define template** command. Create the database to be replicated on all database servers in the replication domain. However, only the database on the synchronization data source server must be populated with data.

All specified servers must be online and the **cdr** utility must be able to connect to each server.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The **cdr realize template** command performs the following tasks:

- If you specify the **--autocreate** option, creates database tables on the target servers.
  Recommendation: If you use **--autocreate**, specify a dbspace name. If you do not specify a dbspace name, tables are created in the root dbspace, which is not recommended.
- If you specify the **--verify** option, verifies the database, tables, column data types, and replication keys on all participating servers; however, the template is not realized.
- If you specify the **--syncdatasource** option, synchronizes the data from the source database with the databases specified by this command. If you specify the **--foreground** option, runs synchronization as a foreground process. If you specify the **--memadjust** option, increases the size of the send queue from the value of the CDR_QUEUEMEM configuration parameter.
  If you are running this command with the **--syncdatasource** option as a DBSA, you must have certain permissions granted to you on several system tables in the **syscdr** database. For more information, see Preparing for Role Separation (UNIX).
- Verifies the database and table attributes to ensure that replication can be performed on each database.
- Creates replicates as master replicates on all servers.
- Creates a replicate set for the new replicates.
- Starts the replicates on all servers.

The replicates and replicate set created from a template have generated names. Use the **cdr list template** command to see the names of the replicates and replicate set associated with a particular template.

You can run this command from within an SQL statement by using the SQL administration API.

You can run the **cdr check queue --qname=cntrlq** command to wait for the **cdr realize template** command to be applied at all Enterprise Replication servers before you run the data synchronization task.

Examples

The following example illustrates the **cdr realize template** command:

```
cdr realize template tem1 -c detroit\
new_cars@detroit new_cars0@chicago new_cars1@newark\
new_cars2@columbus
```

Line 1 specifies that the template name is **tem1** and the server to which to connect is the **detroit** server. Lines 2 and 3 list the names of the databases and database servers on which to realize the template.
The following example illustrates realizing the template on the source server, and then, creating the databases and tables, and loading data on the target database servers:

```
cdr realize template tem1 -c detroit\
--syncdatasource=detroit --extratargetrows=keep\
--foreground --memadjust=50M\
--mode=receive_only chicago newark columbus
```

Line 1 realizes the template on the **detroit** server, as a primary server by default.
Line 2 specifies to use the **detroit** server as the source of the data to replicate to all other participating servers. If Enterprise Replication encounters any rows on the **chicago**, **newark**, or **columbus** servers that do not exist on the **detroit** server, those rows are kept.

Line 3 specifies that the synchronization operation is done in the foreground, and the size of the send queue is set to 50 MB.

Line 4 specifies the participant type for each server. The **--mode=receive_only** option makes each server a receive-only participant.

The following example verifies the database and table attributes on the **chicago**, **newark**, and **columbus** servers; the template is not realized on these servers:

```
cdr realize template tem1 -c detroit\
--verify chicago newark columbus
```

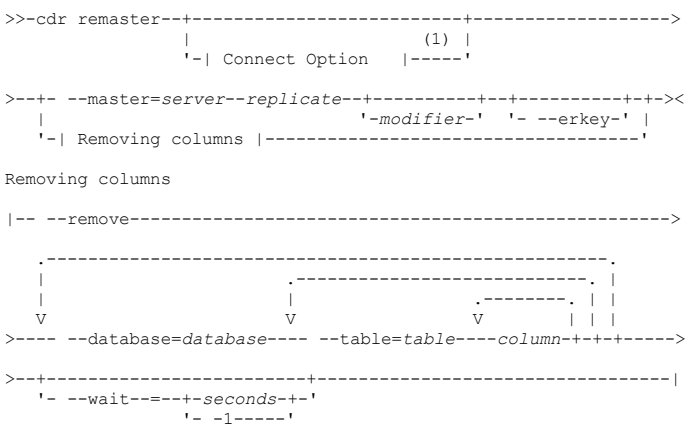
Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related reference:
cdr define template
cdr delete template
cdr list template
cdr define server

---

cdr remaster

The **cdr remaster** command changes the SELECT clause or the server from which to base the master replicate definition of an existing master replicate. This command can also convert a classic (non-master) replicate to a master replicate.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
column	Name of the column to remove from replication.	The column must exist and be a replicated column.	
database	Name of the database from which to remove one or more columns from replication.	The database must exist and contain replicated tables.	
modifier	Specifies the rows and columns to replicate.		Participant and participant modifier
replicate	Name of the replicate to be mastered.	The replicate must exist.	Long Identifiers
server	Name of the database server group from which to base the master replicate definition.	The name must be the database server group name.	Long Identifiers
table	Name of the table from which to remove one or more columns from replication.	The table must exist and belong to a replicate.	

The following table describes the options to the **cdr remaster** command.

Long Form	Short Form	Meaning
--database=	**-d**	Specifies the database name from which to delete replicated columns.
--erkey	**-K**	Includes the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, in the participant definition, if the table that is being replicated has the ERKEY shadow columns. The ERKEY shadow columns are used as the replication key.
--master=	**-M**	Specifies that the replicate being created is a master replicate.
--remove	**-r**	Removes the specified columns from replicate definitions.
--table=	**-t**	Specifies the table name from which to remove one or more replicated columns.
--wait=	**-w**	Specifies how long to wait for remastering to complete. Default is **-1**: wait indefinitely until all replicates are finished being remastered. If the remaster operation is not complete at the end of the waiting time, the operation is rolled back and the columns are not removed.

Usage

Remastering updates the replicate definition in the global catalogs of the replication servers. Use the **cdr remaster** command to perform one of the following tasks:

- Convert a classic replicate to a master replicate. Master replicates ensure schema consistency among the participants in the replicates.
- Update the definition of a master replicate whose participant was changed in an alter operation. You can change the SELECT clause or the server from which to base the master replicate definition.
- Remove one or more replicated columns from one or more replicates. The columns can belong to different replicates. You do not need to know the names of the replicates.

To use the **cdr remaster** command, the master replicate definition must be created with name verification turned on, by using the **cdr define replicate** command with the **--name=y** option.

Use the **--erkey** option if you are adding ERKEY columns to the participant definition, or if you are changing a participant definition that contains the ERKEY shadow columns.

You can run this command from within an SQL statement by using the SQL administration API.

The remastering operation creates temporary shadow replicates that are deleted when the remastering operation is complete. If shadow replicates exist, the remastering operation is in progress. You can run the **cdr list replicate** command to determine if the shadow replicate exists. An example of a shadow replicate name is:

```
Shadow_4_Repl1_GMT1090373046_GID10_PID28836
```

Shadow replicate names have the following format:

```
Shadow_4_basereplicatename_GMTtime_GIDgroupID_PIDpid
```

basereplicatename
: The name of the replicate that is being remastered. If the replicate name is longer than 64 characters, only the first 64 characters are included.

time
: The time stamp of when the shadow replicate was created, in GMT.

groupID
: The group ID of the server. The group ID is the number that is specified by the **-i** option in the group definition in the sqlhosts file.

pid
: The process ID of the client computer.

Example: Add columns to a replicate definition

The following command shows the original definition of the master replicate before the alter operation:

```
cdr define repl --master=delhi -C timestamp\
newrepl "test@delhi.tab" "select col1, col2 from tab"\
```

The following command shows the **cdr remaster** command adding a column, **col3**, in the **newrepl** participant:

```
cdr remaster --master=delhi newrepl\
"select col1, col2, col3 from tab"
```

The following command shows adding the ERKEY shadow columns after the table was altered to include them:

```
cdr remaster --master=delhi newrepl --erkey\
"select col1, col2, col3 from tab"
```

The following command shows changing the participant in the previous example to add another column and to continue to include the ERKEY shadow columns:

```
cdr remaster --master=delhi newrepl --erkey\
"select col1, col2, col3, col4 from tab"
```

Example: Remove columns from replicate definitions

The following command removes three columns from the database **mydb**: the column **prefix** from the table **customer** and the columns **discount** and **season** from the table **sales**:

```
cdr remaster --remove --database=mydb --table=customer prefix \
--table=sales discount season
```

The following command removes one column each from the databases **mydb1**, **mydb2**, and **mydb3**:

```
cdr remaster --remove --database=mydb1 --table=customer prefix \
--database=mydb2 --table=cars brand \
--database=mydb3 --table=regions northwest
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Remastering a Replicate
Removing replicated columns
Preparing tables without primary keys
Related reference:
cdr alter

---

cdr remaster gridtable

The **cdr remaster gridtable** command validates tables in a grid after an alter operation.

Syntax

Usage

You can run the **cdr remaster gridtable** command to check whether tables in a grid have consistent metadata. The **cdr remaster gridtable** command checks every table in a grid on every grid server. The **cdr remaster gridtable** command is run automatically after a grid table is altered.

Return codes

A return code of 0 indicates that the command was successful.

Examples

The following command checks all grid tables for consistency:

```
cdr remaster gridtable
```

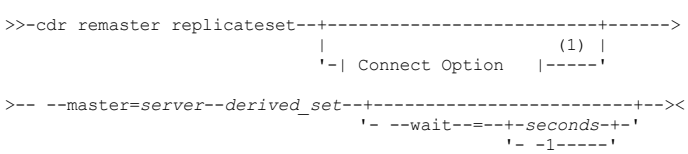
Related concepts:
Grid queries
Related tasks:
Defining tables for grid queries
Related reference:
cdr change gridtable
Related information:
GRID clause

---

cdr remaster replicateset

The **cdr remaster replicateset** command updates the definitions of the set of replicates whose participants were changed by ALTER operations.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
derived_set	Name of the derived replicate set to be mastered.	The derived replicate set must exist.	Long Identifiers

Element	Purpose	Restrictions	Syntax
seconds	Number of seconds to wait for remastering to complete.	The number must be -1 or a positive integer.	
server	Name of the database server group from which to base the replicate definitions.	The name must be the database server group name.	Long Identifiers

The following table describes the options to the **cdr remaster replicateset** command.

Long Form	Short Form	Meaning
--master=	**-M**	Specifies the server from which to base the definitions of the replicates.
--wait=	**-w**	Specifies how long to wait for remastering to complete. Default is **-1**: wait indefinitely until all replicates are finished being remastered. If you specify a waiting time, but the remaster operation is not complete at the end of the waiting time, the operation is rolled back and the replicate definitions are not updated.

Usage

All participant servers in the derived replicate set must be online and the **cdr** utility must be able to connect to each participant.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

When you change replicate participants by running ALTER operations, you must remaster the replicate. Remastering updates the replicate definition in the global catalogs of the replication servers. Before you can run the **cdr remaster replicateset** command, you must run the **cdr define replicateset** command with the **--needRemaster** option to create a derived replicate set.

You can run this command from within an SQL statement by using the SQL administration API.

The remastering operation creates temporary shadow replicates that are deleted when the remastering operation is complete. If shadow replicates exist, the remastering operation is in progress. You can run the **cdr list replicate** command to determine if the shadow replicate exists. An example of a shadow replicate name is:

```
Shadow_4_Repl1_GMT1090373046_GID10_PID28836
```

Shadow replicate names have the following format:

```
Shadow_4_basereplicatename_GMTtime_GIDgroupID_PIDpid
```

basereplicatename
The name of the replicate that is being remastered. If the replicate name is longer than 64 characters, only the first 64 characters are included.

time
The time stamp of when the shadow replicate was created, in GMT.

groupID
The group ID of the server. The group ID is the number that is specified by the **-i** option in the group definition in the sqlhosts file.

pid
The process ID of the client computer.

Example

The following command remasters a derived replicate set named **derived_accounts** and sets the replication server named **server1** as the master server:

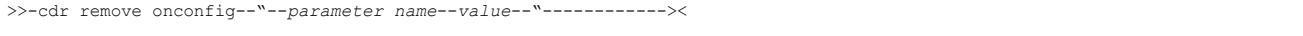
```
cdr remaster replicateset --master=server1 derived_accounts
```

Related tasks:
Altering multiple tables in a replicate set

cdr remove onconfig

The **cdr remove onconfig** command removes the specified value from a configuration parameter in the ONCONFIG file.

Syntax



Element	Purpose	Restrictions	Syntax

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter from which to remove the value.	Not all configuration parameters can be changed with this command. Only the following parameters can be changed: • CDR_LOG_LAG_ACTION • CDR_LOG_STAGING_MAXSIZE • CDR_QDATA_SBSPACE • CDR_SUPRESS_ATSRISWARN • ENCRYPT_CIPHERS • ENCRYPT_MAC • ENCRYPT_MACFILE • CDR_ENV: ◦ CDRSITES_731 ◦ CDRSITES_92X ◦ CDRSITES_10X	
value	The value of the configuration parameter to remove.	Must be an existing value of the configuration parameter.	Follows the syntax rules for the specific configuration parameter.

Usage

Use the **cdr remove onconfig** command to replace the existing value of an Enterprise Replication configuration parameter with a new value in the ONCONFIG file. You can set environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

Suppose the ENCRYPT_MAC configuration parameter is set to allow medium and high encryption levels, so that it appears in the ONCONFIG file as: `ENCRYPT_MAC medium,high`. The following command removes the medium encryption level and retains only the high encryption level:

```
cdr remove onconfig "ENCRYPT_MAC medium"
```

Suppose the CDR_SITES_92X environment variable specifies the cdrIDs of 3, 4, and 5, so that it appears in the ONCONFIG file as: CDR_ENV CDR_SITES_92X=3,4,5. The following command removes the cdrID of 3 from the list of supported version 9.2x servers:

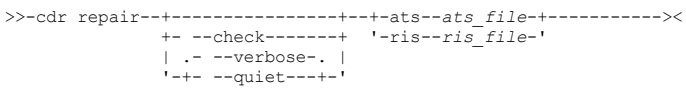
```
cdr remove onconfig "CDR_ENV CDR_SITES_92X=3"
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Dynamically Modifying Configuration Parameters for a Replication Server
Related reference:
cdr add onconfig
cdr change onconfig

cdr repair

The **cdr repair** command synchronizes data based on ATS or RIS files.

Syntax



Element	Purpose	Restrictions	Syntax
ats_file	Name of the file for Aborted Transaction Spooling.	Must be a full path name and file name. The path for the directory can be no longer than 256 bytes. The file must be in text format; it cannot be in XML format.	Follows naming conventions on your operating system.
ris_file	Name of the file for Row Information Spooling.	Must be a full path name and file name. The path for the directory can be no longer than 256 characters. The file must be in text format; it cannot be in XML format.	Follows naming conventions on your operating system.

The following table describes the option to **cdr repair**.

Long Form	Short Form	Meaning

Long Form	Short Form	Meaning
--check	**-C**	Check the consistency between the database server and the ATS or RIS file. Display repair operations to stderr, but do not perform the repair operations. In an active system, operations displayed with this option will not necessarily match those performed later during an actual repair.
--quiet	**-q**	Quiet mode. Repair operations are not displayed to stderr.
--verbose	**-v**	Verbose mode (default). All repair operations are displayed to stderr.

Usage

The **cdr repair** command reconciles rows that failed to be applied based on the information in the specified ATS or RIS file. If a row exists on the source database server, it is replicated again. If a row does not exist on the source database server, but does exist on the target server, then it is deleted from the target database server. By default, each of the repair operations is displayed to stderr.

If you are running this command as a DBSA, you must have read permission on the ATS and RIS files. Permissions on ATS and RIS files can be set with the **chown** operating system command.

The ATS or RIS file you specify in the **cdr repair** command must be in text format, which is the default format. You cannot specify the XML format of an ATS or RIS in the **cdr repair** command.

Before you run a repair, preview the repair to make sure the operations that would be performed are correct. To preview the repair operations, use the **–check** option. All repair operations are displayed to stderr, but not performed. In an active system, however, the operations displayed by the **–check** option might not be the same as the operations performed when you later run the repair.

The server on which you run the **cdr repair** command must have a copy of the ATS or RIS file and be able to connect to the source and target database servers involved in the failed transaction. In a hierarchical routing environment where the source and target database servers are not directly connected you might need to run the **cdr repair** command from an intermediate server. If necessary, copy the ATS or RIS file to the intermediate server.

ATS and RIS files do not include code set information, therefore, the code sets associated with the locales specified by the DB_LOCALE and CLIENT_LOCALE environment variables must be the same.

You can run this command from within an SQL statement by using the SQL administration API.

Note: The **cdr repair** command is not supported for replicates that are defined with the `--UTF8=y` option. For replicates that are defined with the `--UTF8=y` option, use the **cdr check replicate --repair** or **cdr check replicateset --repair** command to repair data.

Examples

The following example repairs inconsistencies between the **g_beijing** and **g_amsterdam** servers resulting from an aborted transaction:

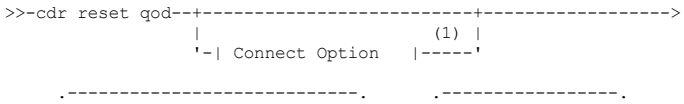
```
% cdr repair ats ats.g_beijing.g_amsterdam.D_2.070827_12:58:55.1
Attempting connection to syscdr@amsterdam...
Using syscdr@amsterdam.
Source ID:10 / Name:g_amsterdam
Target ID:20 / Name:g_beijing
(1) [s = "1"]: Row will be updated on the source for replicate <655361>
(2) [s = "2"]: Row not found on the source for replicate <655361>
(2) [s = "2"]: Row not found on the target for replicate <655361>
(2) [s = "2"]: No operation will be performed.
(3) [s = "3"]: Row will be updated on the source for replicate <655361>
(4) [s = "4"]: Row will be updated on the source for replicate <655361>
(5) [s = "5"]: Row will be updated on the source for replicate <655361>
(6) [s8 = "1911"]: Row will be updated on the source for replicate <655362>
(7) [s8 = "1912"]: Row will be updated on the source for replicate <655362>
(8) [s8 = "1913"]: Row will be updated on the source for replicate <655362>
(9) [s8 = "1914"]: Row will be updated on the source for replicate <655362>
(10) [s8 = "1915"]: Row will be updated on the source for replicate <655362>
```

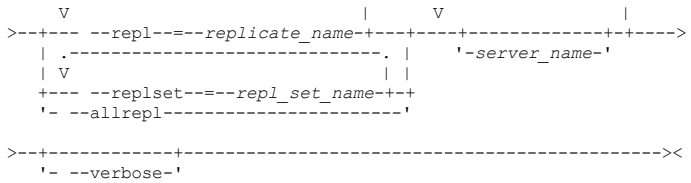
Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related reference:
cdr view

cdr reset qod

The **cdr reset qod** command resets failed-transaction counts for replicates on replicate servers. Connection Manager service-level agreements (SLA) that contains a FAILURE or LATENCY redirection policy use failed-transaction counts to determine where to route client requests.

Syntax





Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
replicate_name	The name of the replicate.	The replicate must exist.	Long Identifiers
repl_set_name	The name of the replicate set.	The replicate set must exist.	Long Identifiers
server_name	The name of the server.	Must be the name of an existing database server group in SQLHOSTS. Cannot be a leaf server.	Long Identifiers

The following table describes the options to the **cdr reset qod** command.

Long Form	Short Form	Meaning
--allrepl	**-A**	Resets the failed-transaction count on all replicates.
--repl=	**-r**	Specifies the replicate for which to reset the failed-transaction count.
--replset=	**-s**	Specifies the replicate set for which to reset the failed-transaction count.
--verbose	**-v**	Displays details of the operations the command is performing

Usage

Use the **cdr reset qod** command to reset the failed-transaction count to zero for replicates or replicate sets on specified replication servers. Run the **cdr reset qod** command before you repair inconsistent data, so that you can count failures that occur after the repair.

You must run the **cdr reset qod** command from a non-leaf server. If you do not specify any servers to reset, the current server to which you are connected is reset. If you specify one or more servers to reset, you must explicitly include the server to which you are connected if you want to reset it.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 44, 217.

For information on error codes, see Return Codes for the cdr Utility.

Example 1: Resetting failed-transaction counts for a specific replicate on a specific replication server

The following example resets the failed-transaction count for **replicate_1** on **server_1**:

```
cdr reset qod --repl=replicate_1 server_1
```

Example 2: Resetting failed-transaction counts for all replicates on specific replication servers

The following example resets the failed-transaction count for all replicates on **server_2** and **server_3**:

```
cdr reset qod --allrepl server_2 server_3
```

Example 3: Resetting failed-transaction counts for all replicates in a specific replicate set on a specific replication server

The following example resets the failed-transaction count for all replicates in **replicate_set_1** on **server_4**.

```
cdr reset qod --replset=replicate_set_1 server_4
```

Example 4: Resetting failed-transaction counts for all replicates in specific replicate sets on a specific replication server

The following example resets the failed-transaction count for all replicates in **replicate_set_2** and **replicate_set_4** on **server_5**.

```
cdr reset qod -s replicate_set_2 -s replicate_set_4 server_5
```

Example 5: Resetting failed-transaction counts for all replicates on a specific replication server, and displaying operation details

The following example connects to **server_6**, and then resets the failed-transaction count for all of replicates on **server_6**. The command displays details of the operations that are performed:

```
cdr reset qod -c -A server_6  -v
```

Figure 1. Output of **cdr reset qod** with verbose details.

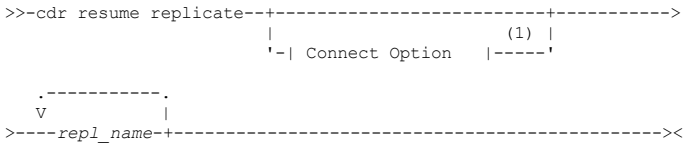
```
Resetting Quality of Data on server_6
  Resetting replicate replicate_1
  Resetting replicate replicate_2
  Resetting replicate replicate_3
  Resetting replicate replicate_4
  Resetting replicate replicate_5
  Resetting replicate replicate_6
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr define qod
cdr start qod
cdr stop qod
Related information:
SLA Connection Manager configuration parameter

cdr resume replicate

The **cdr resume replicate** command resumes delivery of replication data.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate to change to active state.	The replicate must be suspended.	Long Identifiers

Usage

The **cdr resume replicate** command causes all participants in the replicate *repl_name* to enter the active state.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr resume replicate** to resume that individual replicate. You must use **cdr resume replicateset** to resume all replicates in the exclusive replicate set. If a replicate belongs to a non-exclusive replicate set, you can resume the individual replicates in the set.

When you run the **cdr resume replicate** command, an event alarm with a class ID of 57 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server (the one specified by the **INFORMIXSERVER** environment variable) and resumes the replicate **smile**:

```
cdr res repl smile
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Exclusive Replicate Sets
Related reference:
cdr change replicate
cdr define replicate
cdr delete replicate
cdr list replicate
cdr modify replicate

cdr start replicate
cdr stop replicate
cdr suspend replicate
Enterprise Replication Event Alarms
cdr list server
cdr resume replicateset

cdr resume replicateset

The **cdr resume replicateset** command resumes delivery of replication data for all the replicates in a replicate set.

Syntax

```
>>-cdr resume replicateset--+-------------------------+-------->
                            |                   (1)   |
                            '-| Connect Option    |-----'

   .----------.
   V          |
>----repl_set-+-------------------------------------------------><
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to resume.	None.	Long Identifiers

Usage

The **cdr resume replicateset** command causes all replicates contained in the replicate set *repl_set* to enter the active state for all participants.

If not all the replicates in a non-exclusive replicate set are suspended, the **cdr resume replicateset** command displays a warning and only resumes the replicates that are currently suspended.

When you run the **cdr resume replicateset** command, an event alarm with a class ID of 58 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server (the one specified by the **INFORMIXSERVER** environment variable) and resumes the replicate set **accounts_set**:

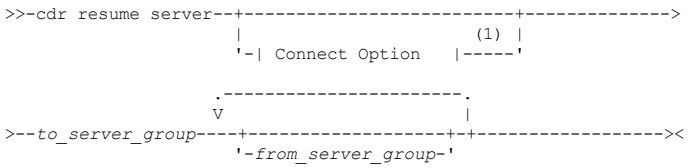
```
cdr res replset accounts_set
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr change replicateset
cdr define replicateset
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr define replicate
cdr start replicateset
cdr stop replicateset
cdr suspend replicateset
Enterprise Replication Event Alarms
cdr resume replicate
cdr list server

cdr resume server

The **cdr resume server** command resumes delivery of replication data to a suspended database server.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions
to_server_group	Name of the database server group to which to resume delivery of replication data.	The database server group must be currently active in Enterprise Replication.
from_server_group	Name of the database server group from which to resume sending data to *to_server_group*.	The database server group must be currently active in Enterprise Replication.

Usage

The **cdr resume server** command resumes delivery of replication data to the *to_server_group* database server from the database servers included in the *from_server_group* list. If the *from_server_group* list is omitted, the command resumes replication of data from all database servers participating in the Enterprise Replication system to the *to_server_group*. Replication data must have previously been suspended to the server with the **cdr suspend server** command.

When you run the **cdr resume server** command, an event alarm with a class ID of 52 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server (the one specified by the **INFORMIXSERVER** environment variable) and resumes replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

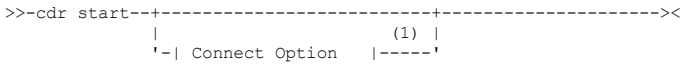
```
cdr res serv g_iowa g_ohio g_utah
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr connect server
cdr define server
cdr delete server
cdr disconnect server
cdr list server
cdr modify server
cdr suspend server
Enterprise Replication Event Alarms

cdr start

The **cdr start** command starts Enterprise Replication processing.

Syntax



Notes:

1. See Connect Option.

Usage

Use **cdr start** to restart Enterprise Replication after you stop it with the **cdr stop** command or replication stops for another reason, such as memory allocation problems. When you issue **cdr start**, Enterprise Replication activates all connections to other connected replication servers. Replication servers, replicates, and replicate sets that were suspended before the **cdr stop** command was issued remain suspended; no data is sent for the suspended servers, replicates, or sets.

Enterprise Replication resumes evaluation of the logical log (if required for the instance of Enterprise Replication) at the *replay* position. The replay position is the position where Enterprise Replication stops evaluating the logical log when replication is stopped. When replication resumes, all appropriate database transactions that occurred while replication was stopped are replicated. If replication is stopped for a prolonged period of time, the replay position in the logical log might be overwritten. If the

replay position is not available, the **cdr start** command fails with return code 214 and event alarm 75 is raised. In this situation, you must empty the send queues and reset the replay position by running the **cdr cleanstart** command, and then synchronize the data.

When you run the **cdr start** command, event alarm 49 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Important: Whenever replication is stopped, data can become inconsistent. Therefore, issue **cdr start** and **cdr stop** with extreme caution.

Examples

The following example restarts Enterprise Replication processing on database server **utah**:

```
cdr start -c utah
```

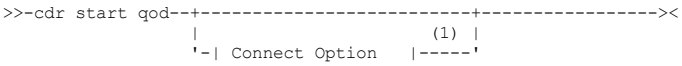
Related concepts:
Enterprise Replication Server administrator
Related tasks:
Restarting Replication on a Server
Related reference:
cdr cleanstart
cdr list server
cdr stop
Enterprise Replication Event Alarms

---

cdr start qod

The **cdr start qod** command starts quality of data (QOD) monitoring for replication servers.

Syntax



Notes:

1. See Connect Option.

Usage

Use the **cdr start qod** command to start monitoring the quality of data for replications servers. If Connection Manager service-level agreements (SLAs) use a apply-failure or transaction-latency redirection policy, the Connection Manager uses quality of data information to decide where to route client connections.

Quality of data information is used for the following SLA redirection policies:

- FAILURE: Connection requests are directed to the replication server that has the fewest apply failures.
- LATENCY: Connection requests are directed to the replication server that has the lowest transaction latency.

You must run the **cdr define qod** command to define a master server before you can run the **cdr start qod** command. The **cdr start qod** command must be run on the master server.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 217.

For information on error codes, see Return Codes for the cdr Utility.

Example

The following command starts quality of data monitoring when it is run on the master server:

```
cdr start qod
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr define qod
cdr stop qod
cdr reset qod
Related information:

cdr start replicate

The **cdr start replicate** command starts the capture and transmittal of replication transactions.

Syntax

```
>>-cdr start replicate--+-------------------------+------------>
                        |                    (1)  |
                        '-| Connect Option   |-----'

              .------------------.
              V                  |
>--repl_name----+--------------+-+------------------------------>
                '-server_group-'

>--+-------------------------------------------------------------+-><
   '- --syncdatasource=data_server--+---------------------------+-'
                                    '-| Synchronization Options |-'

Synchronization Options

|--+--------------------------------+--------------------------->
   '- --extratargetrows=-+-delete-+-'
                         +-keep---+
                         '-merge--'

>--+-----------------------------------------------+---------------|
   '- --foreground--+-------------------------+-'
                    '- --memadjust=size-+-K-+-'
                                        '-M-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be in an Enterprise Replication domain.	
repl_name	Name of the replicate to start.	The replicate must exist.	Long Identifiers
server_group	Name of database server groups on which to start the replicate.	The database server must be in an Enterprise Replication domain.	
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	

The following table describes the **cdr start replicate** options.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--syncdatasource=	**-S**	Specifies the name of the database server to use as the reference copy of the data. This server is started even if it is not listed as one of the servers to start.

Usage

The **cdr start replicate** command causes the replicate to enter the active state (capture-send) on the specified database servers and the source database server specified by the **--syncdatasource** option.

If you are running this command with the **--syncdatasource** option as a DBSA, you must have certain permissions granted to you on several system tables in the **syscdr** database. For more information, see Preparing for Role Separation (UNIX).

If you would like the synchronization operation to be run in the foreground, use the **--foreground** option.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

If no server is specified, the *repl_name* starts on all servers that are included in the replicate. A replicate can have both active and inactive participants. When at least one participant is active, the replicate is active, however, replication does not start until at least two participants are active. You cannot start replicates that have no participants.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr start replicate** to start that individual replicate. You must use **cdr start replicateset** to start all replicates in the exclusive replicate set.

Because Enterprise Replication does not process log records that were produced before the **cdr start replicate** command was run, transactions that occur during this period might be partially replicated. To avoid problems, either issue the **cdr start replicate** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.

You can run the **cdr check queue --qname=cntrlq** command to wait for the **cdr start replicate** command to be applied at all Enterprise Replication servers before running the data synchronization task.

When you run the **cdr start replicate** command, an event alarm with a class ID of 59 is generated, if that event alarm is enabled.

Examples

The following command starts the replicate **accounts** on the server groups **g_svr1** and **g_svr2**:

```
cdr sta rep accounts g_svr1 g_svr2
```

The following example starts the replicate named **accounts** on the server **g_svr1** with **g_svr2** as the source server:

```
cdr start replicate accounts g_svr1 --syncdatasource=g_svr2\
--foreground --memadjust=50M
```

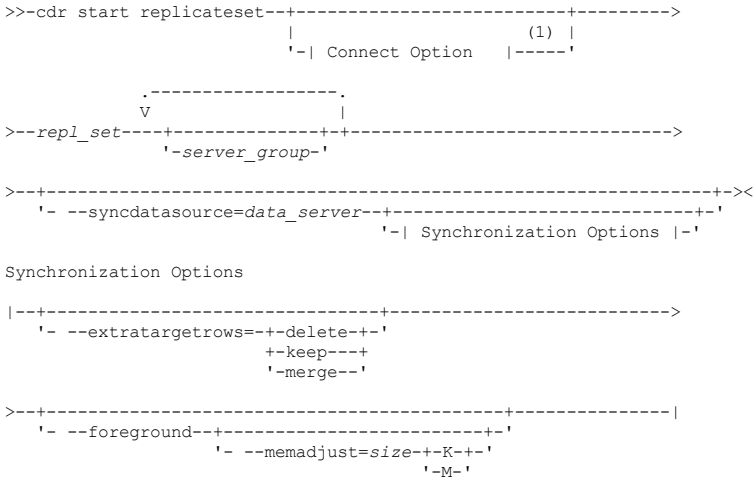
The second line indicates that the synchronization happens in the foreground and the size of the send queue is 50 MB.

Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related reference:
cdr change replicate
cdr define replicate
cdr delete replicate
cdr list replicate
cdr modify replicate
cdr resume replicate
cdr stop replicate
cdr suspend replicate
Enterprise Replication Event Alarms

cdr start replicateset

The **cdr start replicateset** command starts the capture and transmittal of replication transactions for all the replicates in a replicate set.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be defined in Enterprise Replication.	
repl_set	Name of replicate set to start.	The replicate set must exist.	Long Identifiers
server_group	Names of database server groups on which to start the replicate set.	The database server groups must be defined for Enterprise Replication.	
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	

The following table describes the **cdr start replicateset** options.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element
--syncdatasource=	**-S**	Specifies the name of the database server to use as the reference copy of the data. This server is started even if it is not listed as one of the servers to start.

Usage

The **cdr start replicateset** command causes the replicates defined in the specified replicate set to enter the active state (capture-send) on the specified database servers and the source database server specified by the **--syncdatasource** option.

If you are running this command with the **--syncdatasource** option as a DBSA, you must have certain permissions granted to you on several system tables in the **syscdr** database. For more information, see Preparing for Role Separation (UNIX).

If you would like the synchronization operation to be run as in the foreground, use the **--foreground** option.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

If the *server_group* list is omitted, the replicate set *repl_set* enters the active state for all database servers participating in the replicate set.

Because Enterprise Replication does not process log records that were produced before the **cdr start replicateset** command took place, transactions that occur during this period might be partially replicated. To avoid problems, either issue the **cdr start replicateset** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicates in the replicate set.

If not all the replicates in a non-exclusive replicate set are inactive, the **cdr start replicateset** command displays a warning and only starts the replicates that are currently inactive.

When you run the **cdr start replicateset** command, an event alarm with a class ID of 60 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server specified by the **INFORMIXSERVER** environment variable and starts the replicate set **accounts_set** on the server groups **g_hill** and **g_lake**:

```
cdr sta replset accounts_set g_hill g_lake
```

The following example starts the replicate set **accounts_set** on the server **g_hill** with **g_lake** as the source server:

```
cdr start replicateset accounts_set g_hill --syncdatasource=g_lake\
--foreground --memadjust=50M
```

The second line indicates that the synchronization happens in the foreground and the size of the send queue is 50 MB.

Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related reference:
cdr change replicateset
cdr define replicateset
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr define replicate

cdr stop replicateset
cdr suspend replicateset
Enterprise Replication Event Alarms

cdr start sec2er

The **cdr start sec2er** command converts a high availability cluster to replication servers.

Syntax

```
>>-cdr start sec2er--+-------------------------+--secondary---><
                     |                    (1)  |
                     '-| Connect Option   |-----'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
secondary	Name of the secondary server in the cluster.		Long Identifiers

Usage

You must run the **cdr start sec2er** command from a primary server in a cluster with a high-availability data replication secondary or a remote stand-alone secondary server. The **cdr start sec2er** command converts the two cluster servers into replication servers.

The following conditions must be met on both the primary and secondary cluster servers before running the **cdr start sec2er** command:

- The sqlhosts files must be configured for Enterprise Replication:
  - Each server must belong to a different group.
  - The group identifier for each server must be different.
  - The sqlhosts files on each server must contain a server and a group entry for the other server.
- All databases and tables must be logged.
- No tables can be defined with label-based access control.
- Typed tables must have primary keys.
- User-defined types must support Enterprise Replication.
- The CDR_QDATA_SBSPACE configuration parameter must be set.
- Both server must be running version 11.10 or later.
- If the servers are running database software prior to 11.70, Enterprise Replication cannot be defined.
- Enterprise Replication must be active if it is already defined on either of the servers.

The **cdr start sec2er** command performs the following tasks:

- The servers are defined as replication servers.
- Any tables on the primary server that do not have a primary key are altered to add ERKEY shadow columns.
- A replicate is created and started for each user table on the primary server.

If the **cdr start sec2er** command fails or is interrupted, you might see the following error message:

```
ERROR:  Command cannot be run on pre-11.70 instance if ER is already running.
```

If you receive this error, remove replication by running the **cdr delete server** command for both servers and then run the **cdr start sec2er** command again.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error codes is returned: 225.

For information on these error codes, see Return Codes for the cdr Utility.

Example

The following example converts a cluster consisting of a primary server named **priserv** and a secondary server named **secserv** into replication servers. The output of the **cdr start sec2er** command shows the commands that are run.

```
$cdr start sec2er secserv
--
--  Define ER for the first time
--
cdr define serv -c priserv -I priserv

--
--  Creating Replication Key
```

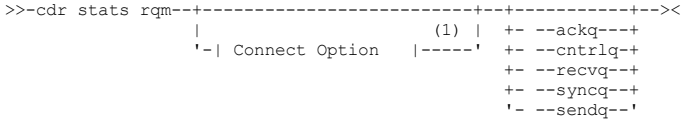
```
--
dbaccess - - <<EOF
database stores_demo;
alter table 'bill'.classes add ERKEY;
EOF

--
--  Define the replicates
--
--
--  Defining Replicates for Database stores_demo
--
cdr define repl --connect=priserv sec2er_1_1282611664_call_type --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.call_type" \
         "select * from 'bill'.call_type"
cdr start repl --connect=priserv sec2er_1_1282611664_call_type

cdr define repl --connect=priserv sec2er_2_1282611664_cust_calls --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.cust_calls" \
         "select * from 'bill'.cust_calls"
cdr start repl --connect=priserv sec2er_2_1282611664_cust_calls

. . .

cdr define repl --connect=priserv sec2er_5_1282611664_customer --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.customer" \
         "select * from 'bill'.customer"
cdr start repl --connect=priserv sec2er_5_1282611664_customer

cdr define repl --connect=priserv sec2er_6_1282611664_classes --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.classes" \
         "select * from 'bill'.classes"
cdr start repl --connect=priserv sec2er_6_1282611664_classes
--
--  Starting RSS to ER conversion
--
--
--  WARNING:
--
--  DDL statements will not be automatically propagated to the ER server
--  after converting the secondary server into an ER server.  If you
--  create or alter any objects, such as databases, tables, indexes, and
--  so on, you must manually propagate those changes to the ER node and
--  change any replication rules affecting those objects.
--
```

Related concepts:
Preparing Logging Databases
Enterprise Replication Server administrator
Related tasks:
Configuring ports and service names for replication servers
Related reference:
CDR_QDATA_SBSPACE Configuration Parameter
cdr check sec2er
Example of creating a new replication domain by cloning

cdr stats rqm

The **cdr stats rqm** command displays information about the reliable queue manager (RQM) queues used for Enterprise Replication.

Syntax



Notes:

1. See Connect Option.

The following table describes the **cdr stats rqm** options.

Long Form	Short Form	Meaning
--ackq	**-A**	Prints the statistics for the acknowledgment send queue.
--cntrlq	**-C**	Prints the statistics for the control send queue.

Long Form	Short Form	Meaning
--recvq	**-R**	Prints the statistics for the receive queue.
--syncq	**-S**	Prints the statistics for the sync send queue.
--sendq	**-T**	Prints the statistics for the send queue.

Usage

The **cdr stats rqm** command displays the RQM (reliable queue manager) statistics for the queues used by Enterprise Replication. These queues are the ack send, control send, send, sync send, and the receive queue. If no queue is specified, the **cdr stats rqm** command displays statistics for all Enterprise Replication queues.

The **cdr stats rqm** command shows, among other things, how many transactions are currently queued in memory and spooled, the size of the data in the queue, how much real memory is being used, pending transaction buffers and data, the maximum memory used for data and headers (overhead), and totals for the number of transactions queued, the number of transactions, the number of deleted transactions, and the number of transaction lookups that have occurred.

If the Connect option is specified, Enterprise Replication connects to the specified remote server and retrieves the statistics for its Enterprise Replication queues.

Examples

The following example shows the output for **cdr stats rqm --ackq**:

```
RQM Statistics for Queue number: 1 name:  ack_send
 Flags:                       ACKSEND_Q, SENDQ_MASK
 Txns in queue:               0
 Txns in memory:              0
 Txns in spool only:          0
 Txns spooled:                0
 Unspooled bytes:             0
 Size of Data in queue:       0 Bytes
 Real memory in use:          0 Bytes
 Pending Txn Buffers:         0
 Pending Txn Data:            0 Bytes
 Max Real memory data used:  44 Bytes
 Max Real memory hdrs used:  320 Bytes
 Total data queued:          120 Bytes
 Total Txns queued:          0
 Total Txns                  3
 Total Txns spooled:         0
 Total Txns restored:        0
 Total Txns recovered:       0
 Spool Rows read:            0
 Total Txns deleted:         3
 Total Txns duplicated:      0
 Total Txn Lookups:          8
```

The following example shows the output for **cdr stats rqm --cntrlq**:

```
RQM Statistics for Queue number: 2 name:  control_send
 Transaction Spool Name:     control_send_stxn
 Flags:                      CTRL_SEND_Q, STABLE, USERTXN, PROGRESS_TABLE,
                             NEED_ACK, SENDQ_MASK
 Txns in queue:              0
 Txns in memory:             0
 Txns in spool only:         0
 Txns spooled:               0
 Unspooled bytes:            0
 Size of Data in queue:      0 Bytes
 Real memory in use:         0 Bytes
 Pending Txn Buffers:        0
 Pending Txn Data:           0 Bytes
 Max Real memory data used:  185 Bytes
 Max Real memory hdrs used:  320 Bytes
 Total data queued:          185 Bytes
 Total Txns queued:          0
 Total Txns                  1
 Total Txns spooled:         1
 Total Txns restored:        0
 Total Txns recovered:       0
 Spool Rows read:            0
 Total Txns deleted:         1
 Total Txns duplicated:      0
 Total Txn Lookups:          4
```

The following example shows the output for **cdr stats rqm --recvq**:

```
RQM Statistics for Queue number: 4 name:  trg_receive
 Transaction Spool Name:     trg_receive_stxn
 Flags:                      RECV_Q, SPOOLED, PROGRESS_TABLE
 Txns in queue:              0
 Txns in memory:             0
 Txns in spool only:         0
 Txns spooled:               0
 Unspooled bytes:            0
 Size of Data in queue:      0 Bytes
 Real memory in use:         0 Bytes
 Pending Txn Buffers:        0
 Pending Txn Data:           0 Bytes
 Max Real memory data used:  0 Bytes
 Max Real memory hdrs used:  0 Bytes
 Total data queued:          0 Bytes
```

```
 Total Txns queued:          0
 Total Txns                  0
 Total Txns spooled:         0
 Total Txns restored:        0
 Total Txns recovered:       0
 Spool Rows read:            0
 Total Txns deleted:         0
 Total Txns duplicated:      0
 Total Txn Lookups:          0
```

The following example shows the output for **cdr stats rqm --syncq**:

```
RQM Statistics for Queue number: 3 name:  sync_send
 Flags:                      SYNC_Q, NEED_ACK, SENDQ_MASK
 Txns in queue:              0
 Txns in memory:             0
 Txns in spool only:         0
 Txns spooled:               0
 Unspooled bytes:            0
 Size of Data in queue:      0 Bytes
 Real memory in use:         0 Bytes
 Pending Txn Buffers:        0
 Pending Txn Data:           0 Bytes
 Max Real memory data used:  0 Bytes
 Max Real memory hdrs used:  0 Bytes
 Total data queued:          0 Bytes
 Total Txns queued:          0
 Total Txns                  0
 Total Txns spooled:         0
 Total Txns restored:        0
 Total Txns recovered:       0
 Spool Rows read:            0
 Total Txns deleted:         0
 Total Txns duplicated:      0
 Total Txn Lookups:          1131
```

The following example shows the output for **cdr stats rqm --sendq**:

```
RQM Statistics for Queue number: 0 name:  trg_send
 Transaction Spool Name:     trg_send_stxn
 Flags:                      SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK,
                             SENDQ_MASK, SREP_TABLE
 Txns in queue:              12
 Txns in memory:             12
 Txns in spool only:         0
 Txns spooled:               0
 Unspooled bytes:            24960
 Size of Data in queue:      24960 Bytes
 Real memory in use:         24960 Bytes
 Pending Txn Buffers:        0
 Pending Txn Data:           0 Bytes
 Max Real memory data used:  24960 Bytes
 Max Real memory hdrs used:  22080 Bytes
 Total data queued:          27560 Bytes
 Total Txns queued:          0
 Total Txns                  14
 Total Txns spooled:         0
 Total Txns restored:        0
 Total Txns recovered:       0
 Spool Rows read:            0
 Total Txns deleted:         2
 Total Txns duplicated:      0
 Total Txn Lookups:          28
```

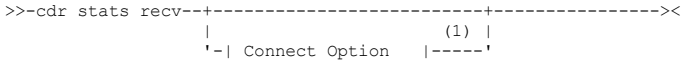
Related concepts:
Enterprise Replication Server administrator

---

cdr stats recv

The **cdr stats recv** command displays receiver parallelism statistics and latency statistics by source node.

Syntax



Notes:

1. See Connect Option.

Usage

The **cdr stats recv** command displays the parallelism statistics for the receiver, including transaction count, number of pending and active transactions, the maximum that have been pending and active, the average number of pending and active transactions, and the commit rate. Totals and averages are calculated for pending and active transactions for the servers listed.

The Statistics by Source report shows the breakdown of transactions (number of inserts, updates, and deletes) and the latest source commit time and target apply time per server. The replication latency is the difference between the time when the transaction was committed on the source server and the time when the same transaction is applied on the target.

If the Connect option is specified, Enterprise Replication connects to the specified remote server and retrieved the statistics from it.

Examples

The following output is an example of the **cdr stats recv** command:

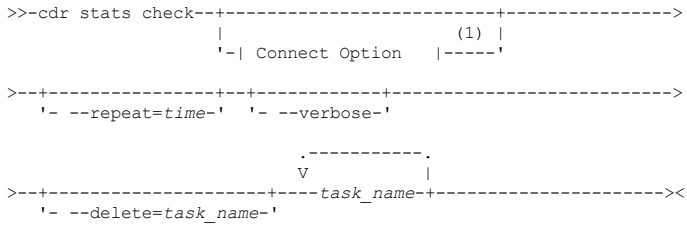
```
cdr stats recv

Receive Parallelism Statistics
Server Tot.Txn. Pending Active MaxPnd MaxAct  AvgPnd  AvgAct CommitRt
   144     11          0      0      3      2    1.27    1.36    0.01

Tot Pending:0       Tot Active:0    Avg Pending:0.00    Avg Active:0.00

Avg Commit Rate:0.01


Statistics by Source

Server  Repl     Txn   Ins   Del   Upd   Last Target Apply    Last Source Commit
144     9371650   11     0     0   220   2005/03/30 09:36:25  2005/03/30 09:36:25
```

Related concepts:
Enterprise Replication Server administrator

---

cdr stats check

The **cdr stats check** command displays the progress of a consistency check that specified a progress report task name.



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task to display.	Must be an existing named task.	Long Identifiers
time	The number of seconds between progress reports.	Must be a positive integer.	

The following table describes the options to the **cdr stats check** command.

Long Form	Short Form	Meaning
--delete=	**-d**	Specifies to delete the specified named task information from the **replcheck_stat** and **replcheck_stat_node** tables.
--repeat=	**-r**	Specifies to repeat the progress report every specified interval of seconds.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr stats check** command to display the progress of a consistency check operation while the **cdr check replicate** or **cdr check replicateset** command is running. You must have specified a task name in the **cdr check replicate** or **cdr check replicateset** command. You must be connected to the same server on which the **cdr check replicate** or **cdr check replicateset** command was run when you run the **cdr stats check** command.

The **cdr stats check** command displays a snapshot of the consistency report and an estimate of the time remaining to complete the consistency check. If you use the **--repeat** option, the consistency report is displayed every specified time interval.

You can view the progress of previously run consistency checks that have named tasks, if those progress report tasks have not been overwritten or deleted.

If you want to see the detailed progress report, include the **--verbose** option. The format of the verbose progress report is the same as the verbose consistency report generated by the **cdr check replicate** and **cdr check replicateset** commands.

If you want to delete a named task, use the **--delete** option.

Examples

The following example checks a replicate named **repl1**, creates a task named **tst**, and then displays a progress report every two seconds.

```
cdr check repl -r repl1 -m cdr1 -a --name=tst
cdr stats check --repeat=2 tst
```

The progress report from the previous command might look like this:

```
        Job tst
        repl1                   Started Jan 17 16:10:59
        *********+----+----+----+----+----+----+----+----+ Remaining 0:00:08

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:10:59
        ********************--+----+----+----+----+----+ Remaining 0:00:04

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:10:59
        ***********************************----+----+----+ Remaining 0:00:02

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:10:59
        ************************************************+ Remaining 0:00:01

        ------------------------------------------------
        Job tst
        repl1                   Completed
                                Started Jan 17 16:10:59, Elapsed Time 0:00:07
```

The following example checks and repairs the replicate, creates a task named **tst**, and displays a verbose progress report every four seconds.

```
cdr check repl -r repl1 -m cdr1 -a --name=tst --repair
cdr stats check --repeat=4 --verbose tst
```

The progress report from the previous command might look like this:

```
        Job tst
        repl1                   Started Jan 17 16:34:42
        *******--+----+----+----+----+----+----+----+----+ Remaining 0:00:12

        Node                Total     Extra   Missing  Mismatch     Child Processed
        ---------------- ---------- ---------- ---------- ---------- ---------- ----------
        cdr1                 9000          0          0          0          0          0
        cdr2                 9000          0          0         99          0         99
        cdr3                 9000          0          0          0          0          0

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:34:42
        *********************************-+----+----+----+ Remaining 0:00:02

        Node                Total     Extra   Missing  Mismatch     Child Processed
        ---------------- ---------- ---------- ---------- ---------- ---------- ----------
        cdr1                43000          0          0          0          0          0
        cdr2                43000          0          0         99          0         99
        cdr3                43000          0          0          0          0          0

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:34:42
        ************************************************+ Remaining 0:00:01

        Node                Total     Extra   Missing  Mismatch     Child Processed
        ---------------- ---------- ---------- ---------- ---------- ---------- ----------
        cdr1                39000          0          0          0          0         99
        cdr2                38901          0         99         99          0         99
        cdr3                39000          0          0          0          0          0

        ------------------------------------------------
        Job tst
        repl1                   Completed
                                Started Jan 17 16:34:42, Elapsed Time 0:00:11

        Node                Total     Extra   Missing  Mismatch     Child Processed
        ---------------- ---------- ---------- ---------- ---------- ---------- ----------
        cdr1                64099          0          0          0          0         99
        cdr2                64000          0         99         99          0         99
        cdr3                64099          0          0          0          0          0
```

The following example checks a replicate set named **set**, creates a task named **tst**, and displays a progress report every five seconds:

```
cdr check replset -s set -m cdr1 -a -n tst
cdr stats check -r 5 tst
```

The progress report from the previous command might look like this:

```
        Job tst
        repl3                     Started Jan 17 16:41:19
        *****----+----+----+----+----+----+----+----+----+ Remaining 0:00:16
        repl2                     Pending
        repl1                     Pending
        Estimated time remaining for job tst  0:00:52

        ------------------------------------------------
        Job tst
        repl3                     Started Jan 17 16:41:19
        ****************************************+----+----+ Remaining 0:00:01
        repl2                     Pending
        repl1                     Pending
        Estimated time remaining for job tst  0:00:19

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Started Jan 17 16:41:27
        ******************+----+----+----+----+----+----+ Remaining 0:00:06
        repl1                     Pending
        Estimated time remaining for job tst  0:00:13

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Completed
                                  Started Jan 17 16:41:27, Elapsed Time 0:00:08
        repl1                     Started Jan 17 16:41:35
        ----+----+----+----+----+----+----+----+----+----+ Remaining 0:01:08
        Estimated time remaining for job tst  0:01:08

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Completed
                                  Started Jan 17 16:41:27, Elapsed Time 0:00:08
        repl1                     Started Jan 17 16:41:35
        **********************************-+----+----+----+ Remaining 0:00:02
        Estimated time remaining for job tst  0:00:02

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Completed
                                  Started Jan 17 16:41:27, Elapsed Time 0:00:08
        repl1                     Completed
                                  Started Jan 17 16:41:35, Elapsed Time 0:00:11
        Run time for job tst  0:00:27
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Checking Consistency and Repairing Inconsistent Rows
Related reference:
The replcheck_stat Table
The replcheck_stat_node Table
cdr check replicate
cdr check replicateset

cdr stats sync

The **cdr stats sync** command displays the progress of a synchronization operation that specified a progress report task name.

```
>>-cdr stats sync--+-------------------------+----------------->
                  |                    (1)  |
                  '-| Connect Option   |-----'

>--+----------------+--+------------+-------------------------->
   '- --repeat=time-'  '- --verbose-'

                           .-----------.
                           V           |
>--+---------------------+----task_name-+---------------------><
   '- --delete=task_name-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task to display.	Must be an existing named task.	Long Identifiers
time	The number of seconds between progress reports.	Must be a positive integer.	

The following table describes the options to the **cdr stats sync** command.

Long Form	Short Form	Meaning
--delete=	**-d**	Specifies to delete the specified named task information from the **replcheck_stat** and **replcheck_stat_node** tables.
--repeat=	**-r**	Specifies to repeat the progress report every specified interval of seconds.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr stats sync** command to display the progress of a synchronization operation (**cdr sync replicate** or **cdr sync replicateset**). You must be connected to the same server on which the **cdr sync replicate** or **cdr sync replicateset** command was run when you run the **cdr stats sync** command. The **cdr stats sync** command displays a snapshot of the progress report and an estimate of the time remaining to complete the synchronization operation. If you use the **--repeat** option, the progress report is displayed every specified time interval.

You can view the progress of previously run synchronization operations that have named tasks, if those progress report tasks have not been overwritten or deleted.

If you want to see the detailed progress report, include the **--verbose** option. The format of the verbose progress report is the same as the verbose consistency report generated by the **cdr check replicate** and **cdr check replicateset** commands.

If you want to delete a named task, use the **--delete** option.

Examples

The following example synchronizes a replicate named **repl1**, creates a task named **tst**, and then displays a progress report every two seconds.

```
cdr sync repl -r repl1 -m cdr1 -a --name=tst
cdr stats sync --repeat=2 tst
```

The progress report from the previous command might look like this:

```
        Job tst
        repl1                   Started Jan 17 16:10:59
        *********+----+----+----+----+----+----+----+----+ Remaining 0:00:08

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:10:59
        **********************--+----+----+----+----+----+ Remaining 0:00:04

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:10:59
        ************************************----+----+----+ Remaining 0:00:02

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:10:59
        ************************************************+ Remaining 0:00:01

        ------------------------------------------------
        Job tst
        repl1                   Completed
                                Started Jan 17 16:10:59, Elapsed Time 0:00:07
```

The following example synchronizes the replicate, creates a task named **tst**, and displays a verbose progress report every four seconds.

```
cdr sync repl -r repl1 -m cdr1 -a --name=tst
cdr stats sync --repeat=4 --verbose tst
```

The progress report from the previous command might look like this:

```
        Job tst
        repl1                   Started Jan 17 16:34:42
        *******--+----+----+----+----+----+----+----+----+ Remaining 0:00:12

        Node                  Total     Extra   Missing  Mismatch     Child Processed
        ----------------- --------- --------- --------- --------- --------- ---------
        cdr1                   9000         0         0         0         0         0
        cdr2                   9000         0         0        99         0        99
        cdr3                   9000         0         0         0         0         0

        ------------------------------------------------
        Job tst
        repl1                   Started Jan 17 16:34:42
        *********************************-+----+----+----+ Remaining 0:00:02

        Node                  Total     Extra   Missing  Mismatch     Child Processed
        ----------------- --------- --------- --------- --------- --------- ---------
        cdr1                  43000         0         0         0         0         0
```

```
        cdr2                  43000         0         0        99         0        99
        cdr3                  43000         0         0         0         0         0

        ------------------------------------------------
        Job tst
        repl1                     Started Jan 17 16:34:42
        ***********************************************+ Remaining 0:00:01

        Node                  Total     Extra   Missing  Mismatch     Child Processed
        ----------------- ---------- --------- --------- --------- --------- ---------
        cdr1                  39000         0         0         0         0        99
        cdr2                  38901         0        99        99         0        99
        cdr3                  39000         0         0         0         0         0

        ------------------------------------------------
        Job tst
        repl1                     Completed
                                  Started Jan 17 16:34:42, Elapsed Time 0:00:11

        Node                  Total     Extra   Missing  Mismatch     Child Processed
        ----------------- ---------- --------- --------- --------- --------- ---------
        cdr1                  64099         0         0         0         0        99
        cdr2                  64000         0        99        99         0        99
        cdr3                  64099         0         0         0         0         0
```

The following example synchronizes a replicate set named **set**, creates a task named **tst**, and displays a progress report every five seconds:

```
cdr sync replset -s set -m cdr1 -a -n tst
cdr stats sync -r 5 tst
```

The progress report from the previous command might look like this:

```
        Job tst
        repl3                     Started Jan 17 16:41:19
        *****----+----+----+----+----+----+----+----+----+ Remaining 0:00:16
        repl2                     Pending
        repl1                     Pending
        Estimated time remaining for job tst  0:00:52

        ------------------------------------------------
        Job tst
        repl3                     Started Jan 17 16:41:19
        ****************************************+----+----+ Remaining 0:00:01
        repl2                     Pending
        repl1                     Pending
        Estimated time remaining for job tst  0:00:19

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Started Jan 17 16:41:27
        ******************+----+----+----+----+----+----+ Remaining 0:00:06
        repl1                     Pending
        Estimated time remaining for job tst  0:00:13

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Completed
                                  Started Jan 17 16:41:27, Elapsed Time 0:00:08
        repl1                     Started Jan 17 16:41:35
        ----+----+----+----+----+----+----+----+----+----+ Remaining 0:01:08
        Estimated time remaining for job tst  0:01:08

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Completed
                                  Started Jan 17 16:41:27, Elapsed Time 0:00:08
        repl1                     Started Jan 17 16:41:35
        *********************************-+----+----+----+ Remaining 0:00:02
        Estimated time remaining for job tst  0:00:02

        ------------------------------------------------
        Job tst
        repl3                     Completed
                                  Started Jan 17 16:41:19, Elapsed Time 0:00:08
        repl2                     Completed
                                  Started Jan 17 16:41:27, Elapsed Time 0:00:08
        repl1                     Completed
                                  Started Jan 17 16:41:35, Elapsed Time 0:00:11
        Run time for job tst  0:00:27
```

Related concepts:
Enterprise Replication Server administrator
Related tasks:
Performing Direct Synchronization
Related reference:
The replcheck_stat Table
The replcheck_stat_node Table

cdr sync replicate
cdr sync replicateset

cdr stop

The **cdr stop** command stops replication on the server to which you are connected without shutting down the database server.

Syntax

```
                                (1)
>>-cdr stop--| Connect Option |--------------------------------><
```

Notes:

1. See Connect Option.

Usage

Generally, to stop replication on a server, you should shut down the database server. Under rare conditions, you might want to temporarily stop the Enterprise Replication processing without shutting down the database server.

The **cdr stop** command shuts down replication in an orderly manner; however no data to be replicated is captured. When the shutdown of Enterprise Replication is complete, the message `CDR shutdown complete` appears in the database server log file.

Stopping replication has the following effects:

- There is no connection between the stopped server and active replication servers.
- Transactions on the stopped server are not captured for replication. However, after restarting replication, transaction capture restarts at the replay position. If replication is stopped for long enough that the replay position is overwritten, you must restart replication with the **cdr cleanstart** command. If the CDR_LOG_LAG_ACTION configuration parameter is set to **logstage**, logs are staged to protect the replay position.
- Transactions on active replication servers are queued for the stopped server, but there is the possibility of filling up the send queues.
- Control messages on active replication servers are queued for the stopped server.
- The only Enterprise Replication commands you can run on the stopped server are **cdr start**, **cdr cleanstart**, and **cdr delete server** with the **--force** option.

To ensure consistency, prevent database update activity while Enterprise Replication is stopped. Replication threads remain stopped until you issue a **cdr start** command. Shutting down and restarting the stopped database server does not restart replication.

If you plan to stop replication for a long period of time and your replicates use time stamp or delete wins conflict resolution rules, consider using the **cdr disable server** command instead of the **cdr stop** command.

When you run the **cdr stop** command, event alarms with class IDs of 50 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Return Codes

0
: The command was successful.

5
: Enterprise Replication cannot connect to the specified server.

48
: There is not enough memory to perform the operation.

62
: Enterprise Replication is not active.

93
: Enterprise Replication is in the process of starting.

94
: Enterprise Replication is already in the process of stopping.

Examples

The following example stops Enterprise Replication processing on database server **paris**. Processing does not resume until a **cdr start** command restarts it:

```
cdr stop -c paris
```

Related concepts:
Resynchronizing Data among Replication Servers
Enterprise Replication Server administrator
Related tasks:
Temporarily stopping replication on a server
Related reference:
cdr start
Enterprise Replication Event Alarms

cdr stop qod

The **cdr stop qod** command stops quality of data (QOD) monitoring for replication servers.

Syntax

```
>>-cdr stop qod--+-------------------------+------------------><
                |                     (1) |
                '-| Connect Option   |-----'
```

Notes:

1. See Connect Option.

Usage

Use the **cdr stop qod** command to stop monitoring quality of data for the replication servers.

The **cdr stop qod** command must be run on the master server defined by the **cdr define qod** command.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error code is returned: 217.

For information on this error code, see Return Codes for the cdr Utility.

Example 1: Stopping quality of data monitoring from a master server

The following example stops quality of data monitoring:

```
cdr stop qod
```

This command must be run on the master server that was defined by the **cdr define qod** command.

Example 2: Connecting to a master server, and then stopping quality of data monitoring

For the following example, **server_1** was defined as the master server by the **cdr define qod** command. The following example connects to **server_1**, and then stops quality of data monitoring:

```
cdr stop qod -c server_1
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr define qod
cdr start qod
cdr reset qod
Related information:
SLA Connection Manager configuration parameter

cdr stop replicate

The **cdr stop replicate** command stops the capture, transmittal, and reception of transactions for replication.

Syntax

```
>>-cdr stop replicate--+-------------------------+------------->
                      |                     (1) |
                      '-| Connect Option   |-----'

              .---------------------.
              V                     |
>--repl_name----+-----------------+-+--------------------------><
                '-at_server_group-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the new replicate.	The replicate must be active and not in an exclusive replicate set.	Long Identifiers
at_server_group	List of database server groups on which to stop the replicate.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr stop replicate** command changes the state of the replicate *repl_name* to inactive (no replicated data is captured, sent or received) on the replication servers in the specified *at_server_group* list. In addition, this command deletes any data in the send queue for the stopped replicate. You cannot stop replicates that have no participants.

If you omit the *at_server_group* list, the replicate enters the inactive state on all database servers participating in the replicate and all send queues for the replicate are deleted.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr stop replicate** to stop that individual replicate. You must use **cdr stop replicateset** to stop all replicates in the exclusive replicate set.

If you run this command while direct synchronization or consistency checking with repair is in progress, that repair process will stop. (Consistency checking continues; only the repair stops.) Direct synchronization and consistency checking repair cannot be resumed; you must rerun **cdr sync replicate** or **cdr check replicate** command with the **--repair** option.

When you run the **cdr stop replicate** command, an event alarm with a class ID of 61 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following command connects to the database server **lake** and stops the replicate **aRepl** on server groups **g_server1** and **g_server2**:

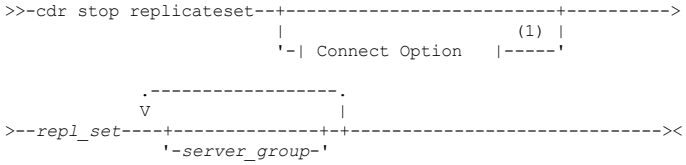
```
cdr sto rep -c lake aRepl g_server1 g_server2
```

Related concepts:
Resynchronizing Data among Replication Servers
Enterprise Replication Server administrator
Related reference:
cdr change replicate
cdr define replicate
cdr delete replicate
cdr list replicate
cdr modify replicate
cdr resume replicate
cdr start replicate
cdr suspend replicate
Enterprise Replication Event Alarms

cdr stop replicateset

The **cdr stop replicateset** command stops capture and transmittal transactions for all the replicates in a replicate set.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to stop.	The replicate set must exist	Long Identifiers
server_group	Name of database server group on which to stop the replicate group.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr stop replicateset** command causes all replicates in the replicate set *repl_set* to enter the inactive state (no capture, no send) on the database servers in the *server_group* list.

If the *server_group* list is omitted, the replicate set *repl_set* enters the inactive state for all database servers participating in the replicate set.

If not all the replicates in the non-exclusive replicate set are active, the **cdr stop replicateset** command displays a warning and only stops the replicates that are currently active.

If you run this command while direct synchronization or consistency checking with repair is in progress, that repair process will stop. (Consistency checking continues; only the repair stops.) Direct synchronization and consistency checking repair cannot be resumed; you must rerun **cdr sync replicate** or **cdr check replicate** command.

When you run the **cdr stop replicateset** command, an event alarm with a class ID of 62 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the database server **paris** and stops the replicate set **accounts_set** on server groups **g_utah** and **g_iowa**:

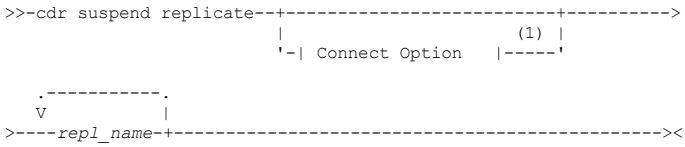
```
cdr sto replset --connect=paris accounts_set g_utah g_iowa
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr change replicateset
cdr define replicateset
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr define replicate
cdr suspend replicateset
Enterprise Replication Event Alarms

cdr suspend replicate

The **cdr suspend replicate** command suspends delivery of replication data.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate.	The replicate must be active.	Long Identifiers

Usage

The **cdr suspend replicate** command causes the replicate *repl_name* to enter the suspend state (capture, no send) for all participants.

Attention: When a replicate is suspended, Enterprise Replication holds the replication data in the send queue until the replicate is resumed. If a large amount of data is generated for the replicate while it is suspended, the send queue space can fill, causing data to be lost. Enterprise Replication does not synchronize transactions if a replicate is suspended. For example, a transaction that updates tables X and Y will be split if replication for table X is suspended.
If a replicate belongs to an exclusive replicate set, you cannot run **cdr suspend replicate** to suspend that individual replicate. You must use **cdr suspend replicateset** to suspend all replicates in the exclusive replicate set.

When you run the **cdr suspend replicate** command, an event alarm with a class ID of 55 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the database server **stan** and suspends the replicate **house**:

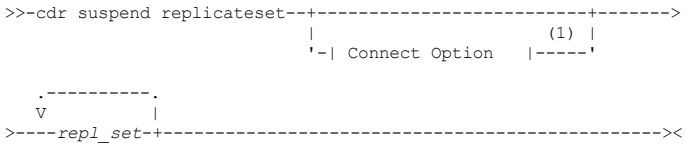
```
cdr sus repl --connect=stan house
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr change replicate
cdr define replicate
cdr delete replicate
cdr list replicate
cdr modify replicate
cdr resume replicate
cdr start replicate
cdr stop replicate
Enterprise Replication Event Alarms
cdr suspend replicateset

cdr suspend replicateset

The **cdr suspend replicateset** command suspends delivery of replication data for all the replicates in a replicate set.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to suspend.	The replicate set must exist.	Long Identifiers

Usage

The **cdr suspend replicateset** command causes all the replicates in the replicate set *repl_set* to enter the suspend state. Information is captured, but no data is sent for any replicate in the set. The data is queued to be sent when the set is resumed.

Attention: When a replicate set is suspended, Enterprise Replication holds the replication data in the send queue until the set is resumed. If a large amount of data is generated for the replicates in the set while it is suspended, the send queue space can fill, causing data to be lost. Enterprise Replication does not synchronize transactions if a replicate in a replicate set is suspended. For example, a transaction that updates tables X and Y will be split if replication for table X is suspended.
If not all the replicates in the non-exclusive replicate set are active, the **cdr suspend replicateset** command displays a warning and only suspends the replicates that are currently active.

When you run the **cdr suspend replicateset** command, an event alarm with a class ID of 56 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server specified by **$INFORMIXSERVER** and suspends the replicate set **accounts_set**:

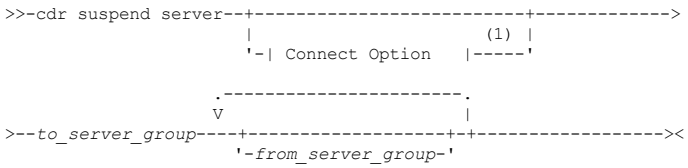
```
cdr sus replset account_set
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr change replicateset
cdr define replicateset
cdr delete replicateset
cdr list replicateset
cdr modify replicateset
cdr resume replicateset
cdr start replicateset
cdr stop replicateset
cdr define replicate
Enterprise Replication Event Alarms
cdr suspend replicate

cdr suspend server

The **cdr suspend server** command suspends the delivery of replication data to a database server from either a specified list of database servers or from all database servers in the domain.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions
to_server_group	Name of database server group to which to suspend delivery of replication data.	The database server group must be currently active in Enterprise Replication.
from_server_group	Name of the database server group from which to stop sending data to *to_server_group*.	The database server group must be currently active in Enterprise Replication.

Usage

The **cdr suspend server** command suspends delivery of replication data to the *to_server_group* database server from the database servers included in the *from_server_group* list. If the *from_server_group* list is omitted, the command suspends replication of data from all database servers participating in the replication domain to the *to_server_group*.

Suspending replication has the following effects:

- The connections between the suspended server and active replication servers remain active.
- Transactions on the suspended replication server are sent to the active replication servers.
- Transactions on active replication servers are queued for the suspended replication server.
- Control messages on active replication servers are sent to the suspended replication server.
- Control messages on the suspended replication server are sent to the active replication servers.

To restart replication on a suspended replication server, run the **cdr resume server** command. Shutting down and restarting the suspended database server does not resume replication.

When you run the **cdr suspend server** command, an event alarm with a class ID of 51 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server (the one specified by the **INFORMIXSERVER** environment variable) and suspends replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

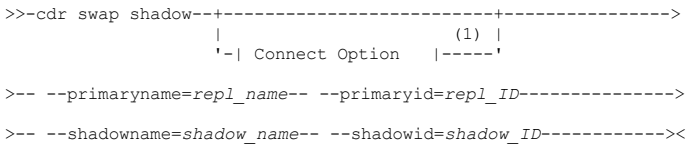
```
cdr sus serv g_iowa g_ohio g_utah
```

Related concepts:
Enterprise Replication Server administrator
Related reference:
cdr connect server
cdr define server
cdr delete server
cdr disconnect server
cdr list server
cdr modify server
cdr resume server
Enterprise Replication Event Alarms

cdr swap shadow

The **cdr swap shadow** command switches a replicate with its shadow replicate during manual remastering.

Syntax



Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the primary replicate.	The primary replicate participant attributes state, type (**P** or **R**), and table owner (**O** or **I**) must match the shadow replicate participant attributes.	Long Identifiers
repl_ID	Internal Enterprise Replication identification code for the primary replicate.		
shadow_name	Name of the shadow replicate.	The shadow replicate state must match the primary replicate state. Shadow replicate participants must match the primary replicate participants.	Long Identifiers
shadow_ID	Internal Enterprise Replication identification code for the shadow replicate.		

The following table describes the **cdr swap shadow** options.

Long Form	Short Form	Meaning
--primaryname=	**-p**	Specifies the name of the primary replicate.
--primaryid=	**-P**	Specifies the ID of the primary replicate.
--shadowname=	**-s**	Specifies the name of the shadow replicate.
--shadowid=	**-S**	Specifies the ID of the shadow replicate.

Usage

Use the **cdr swap shadow** command to switch a replicate with its shadow replicate as the last step in manually remastering a replicate that was created with the **--name=n** option. You create a shadow replicate using the **cdr define replicate** command with the **--mirrors** option.

Use the **onstat -g cat repls** command to obtain the *repl_ID* and *shadow_ID*. Alternatively, you can query the **syscdrrepl** view in the **sysmaster** database.

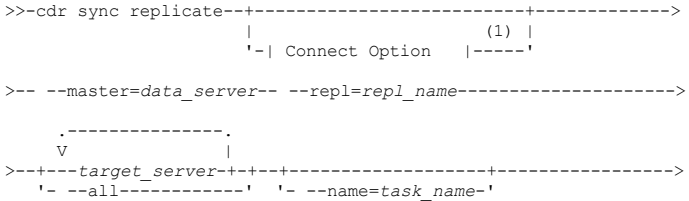
You can run this command from within an SQL statement by using the SQL administration API.

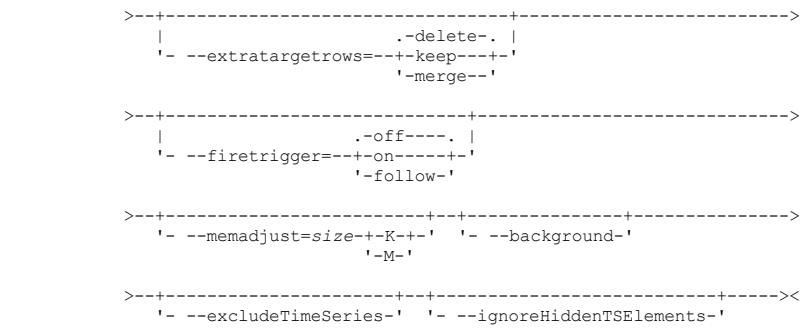
Related concepts:
Enterprise Replication Server administrator
Related tasks:
Remastering replicates without name verification
Enabling code set conversion between replicates
Related reference:
cdr alter
cdr define replicate
cdr list replicate
Participant and participant modifier
onstat -g rep: Prints the schedule manager queue

cdr sync replicate

The **cdr sync replicate** command synchronizes data among replication servers to repair inconsistent data within a replicate.

Syntax





Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
repl_name	Name of the replicate to synchronize.		Long Identifiers
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	
target_server	Name of a database server group on which to perform synchronization.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers

The following table describes the **cdr sync replicate** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are checked.
--background	**-B**	Specifies that the operation is run in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name that is specified by the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--excludeTimeSeries		Specifies to prevent the checking of time series data.
--extratargetrows=	**-e**	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns. Note: When **cdr sync replicate** is used with **-extratargetrows** (or -e) option for SEND-ONLY replicate, server displays the following warning and continue the operation: `WARNING: Extra row option isn't applicable to Send-Only participant.` `'--extratargetrows' option is ignored` Note: When **cdr syn replicate** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted: `Error: Receive only participant '%s' can not be a master node in data synchronization task`
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while data is synchronizing: • **off**: (default) do not fire triggers at target servers during synchronization • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--master=	**-m**	Specifies the database server to use as the reference copy of the data.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.

Long Form	Short Form	Meaning
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--repl=	**-r**	Specifies the name of the replicate to synchronize.

Usage

Use the **cdr sync replicate** command to synchronize data between multiple database servers for a specific replicate. This command performs direct synchronization as a foreground process.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

If you want to monitor the progress of the synchronization operation, include the **--name** option and specify a name for the progress report task. Then run the **cdr stats sync** command and specify the progress report task name.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the **--name** option so that you can monitor the operation with the **cdr stats sync** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

The **cdr sync replicate** command performs the following tasks:

1. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
2. Performs a sequential scan of the replicated table on the source server.
3. Replicates the all rows in the table from the source server to the target server by copying the data directly into the send queue, bypassing the logical logs. Rows are not replicated to participants that include the **S** option in the participant definition because those participants only send data.
4. Deletes the shadow replicate.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 17, 18, 31, 37, 48, 53, 61, 75, 99, 101, 121, 172, 174, 178, 193, 194, 195, 200, 203, 204.

For information on these error codes, see Return Codes for the cdr Utility

Example 1: Synchronize all servers

The following example illustrates synchronizing all replication servers for the replicate named **repl_1**:

```
cdr sync replicate --master=g_serv1 --repl=repl_1\
--all --extratargetrows=keep\
--firetrigger=on
```

The data on the server group **g_serv1** is used as the reference for correcting the data on the other servers. Line 2 indicates that all servers associated with the replicate are synchronized and that if the synchronization operation detects rows on the target servers that do not exist on the reference server (**g_serv1**), that those rows should remain on the other servers. Line 3 indicates that triggers should be fired on the target servers even if the replicate definition does not include the **--firetrigger** option.

Example 2: Synchronize three servers

The following example illustrates synchronizing three servers for the replicate named **repl_2**:

```
cdr sync replicate -m g_serv1 -r repl_2\
g_serv2 g_serv3
```

The reference server is **g_serv1** and the target servers are **g_serv2** and **g_serv3**. Because the **--extratargetrows** option is not specified, the default behavior occurs: rows, and any dependent rows that are based on referential integrity constraints, that are on the target servers but not on the reference server, are deleted.

Example 3: Synchronize in the background and set the send queue size

The following example illustrates synchronizing in the background and setting the size of the send queue to 50 MB:

```
cdr sync replicate --master=g_serv1 --repl=repl_1\
--memadjust=50M --background
```

Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related tasks:
Performing Direct Synchronization
Related reference:
cdr check replicate

cdr sync replicateset

ASDF The **cdr sync replicateset** command synchronizes data among replication servers to repair inconsistent data within a replicate set.

Syntax

```
>>-cdr sync replicateset--+-------------------------+---------->
                          |                    (1) |
                          '-| Connect Option   |-----'

>-- --master=data_server--+- --replset=repl_set-+--------------->
                          '- --allrepl----------'

     .---------------.
     V               |
>--+---target_server-+-+--+------------------+----------------->
   '- --all-----------'  '- --name=task_name-'

>--+-----------------------------------+-------------------------->
   |                       .-delete-. |
   '- --extratargetrows= -+-keep---+-'
                         '-merge--'

>--+-------------------------------+------------------------------>
   |                   .-off----. |
   '- --firetrigger= -+-on-----+-'
                     '-follow-'

>--+-------------------------+--+---------------+--------------->
   '- --memadjust=size-+-K-+-'  '- --background-'
                       '-M-'

>--+-----------------------------+--+----------------------+---->
   '- --process=number_processes-'  '- --excludeTimeSeries-'

>--+---------------------------+-------------------------------><
   '- --ignoreHiddenTSElements-'
```

Notes:

1. See Connect Option.

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
number_processes	The number of parallel processes to use for the command.	The maximum number of processes Enterprise Replication can use is equal to the number of replicates in the replicate set.	
repl_set	Name of the replicate set. Can be the name of a derived replicate set.		Long Identifiers
seconds	The number of seconds to wait for a disabled replication server to be recognized as active by other replication servers in the domain and how long to wait for control messages queued at peer servers to be applied at newly-enabled server.	Must be an integer value from 0 to 60.	
target_server	Name of a database server group to check.	Must be the name of an existing database server group in SQLHOSTS.	Long Identifiers
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	Long Identifiers

The following table describes the **cdr sync replicateset** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are checked.
--allrepl	**-A**	Specifies that all replicates are synchronized.
--excludeTimeSeries		Specifies to prevent the checking of time series data.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows that are found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server. You cannot use this option for replicates that include **TimeSeries** columns. Note: When **cdr sync replicateset** is used with **-extratargetrows** (or -e) option for SEND-ONLY replicate, server displays the following warning and continue the operation: `WARNING: Extra row option isn't applicable to Send-Only participant.` `'--extratargetrows' option is ignored` Note: When **cdr syn replicateset** is used with **-extratargetrows** (or -e) option for RECV-ONLY participant as a MASTER node, server displays the following error and operation is aborted: `Error: Receive only participant '%s' can not be a master node in data synchronization task`
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while data is synchronizing: • **off**: (default) do not fire triggers at target servers during synchronization • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option
--ignoreHiddenTSElements		Specifies to avoid checking time series elements that are marked as hidden.
--master=	**-m**	Specifies the database server to use as the reference copy of the data.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--process=	**-p**	Specifies to run the command in parallel, using the specified number of processes. At most, Enterprise Replication can use one process for each replicate in the replicate set. If you specify more processes than replicates, the extra processes are not used. Not all replicates can be processed in parallel. For example, if replicates have referential integrity rules, the replicates with the parent tables must be processed before the replicates with the child tables.
--replset	**-s**	Specifies the name of the replicate set to synchronize.

Usage

Use the **cdr sync replicateset** command to synchronize data between multiple database servers for a replicate set. This command performs direct synchronization as a foreground process.

If you run this command as a DBSA instead of as user **informix**, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

You can significantly improve the performance of synchronizing a replicate set by synchronizing the member replicates in parallel. You specify the number of parallel processes with the **--process** option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

If you want to monitor the progress of the synchronization operation, include the **--name** option and specify a name for the progress report task. Then run the **cdr stats sync** command and specify the progress report task name.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the **--name** option so that you can monitor the operation with the **cdr stats sync** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

To synchronize all replicates at once, use the **--allrepl** option.

The **cdr sync replicateset** command performs the following tasks:

1. Determines the order in which to repair tables if they have referential relationships.
2. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.

3. Performs a sequential scan of the replicated table on the source server.
4. Replicates the all rows in the table from the source server to the target server by copying the data directly into the send queue, bypassing the logical logs. Rows are not replicated to participants that include the **S** option in the participant definition because those participants only send data.
5. Deletes the shadow replicate.
6. Repeats steps 2 through 5 for each replicate in the replicate set.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 11, 17, 18, 31, 37, 48, 53, 61, 75, 99, 101, 121, 166, 172, 174, 193, 194, 195, 200, 203, 204, 213.

For information on these error codes, see Return Codes for the cdr Utility

Example 1: Synchronize all servers

The following example illustrates synchronizing all replication servers for the replicate set **replset_1** using **g_serv1** as the reference server:

```
cdr sync replicateset --master=g_serv1 --replset=replset_1\
--all --extratargetrows=keep
```

Line 2 indicates that all servers associated with the replicate set are synchronized and that if the synchronization process detects rows on the target servers that do not exist on the reference server (**g_serv1**), that those rows should remain on the other servers.

Example 2: Synchronize three servers in parallel

The following example illustrates synchronizing three servers for the replicate set named **replset_2** and using two processes to synchronize each of the two replicates in the set in parallel:

```
cdr sync replicateset -m g_serv1 -s replset_2\
g_serv2 g_serv3 --process=2
```

The reference server is **g_serv1** and the target servers are **g_serv2** and **g_serv3**. Because the **--extratargetrows** option is not specified, the default behavior occurs: rows, and any dependent rows that are based on referential integrity constraints, that are on the target servers but not on the reference server, are deleted.

Example 3: Synchronize in the background and set the send queue size

The following example illustrates synchronizing in the background and setting the size of the send queue to 50 MB:

```
cdr sync replicateset --master=g_serv1 --replset=replset_1\
--memadjust=50M --background
```

Example 4: Synchronize all replicate sets on a replication server

The following command synchronizes all replicate sets on a replication server named **g_serv2**:

```
cdr sync replicateset --allrepl g_serv2
```

The replicate set name is not specified because the **--allrepl** option is used.

Related concepts:
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related tasks:
Altering multiple tables in a replicate set
Performing Direct Synchronization
Related reference:
cdr check replicateset
cdr stats sync

cdr -V

The **cdr -V** command displays the version of Informix® that is currently running.

Syntax

```
>>-cdr -V-------------------------------------------------------><
```

Usage

Use the **cdr -V** command if you need to obtain the version of the database server, usually at the request of Software Support.

Examples

The following example shows an example output of the **cdr -V** command:

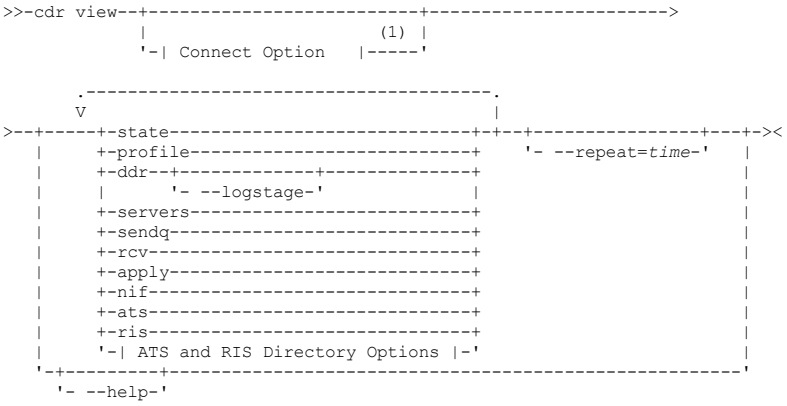
```
IBM Informix Version 11.70.UC1    Software Serial Number RDS#N000000
```

Related concepts:
Enterprise Replication Server administrator

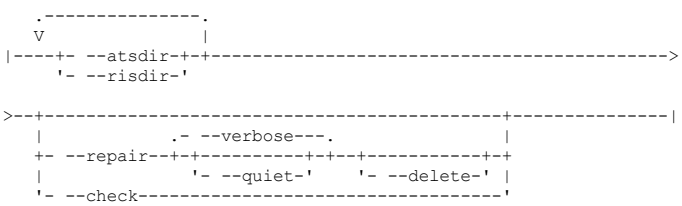
cdr view

The **cdr view** command shows information about every Enterprise Replication server in the domain.

Syntax



ATS and RIS Directory Options



Notes:

1. See Connect Option.

Element	Purpose	Restrictions
time	The number of seconds before the **cdr view** command is repeated.	Must be a positive integer.

The following table describes the **cdr view** subcommands.

Long Form	Meaning
apply	Show a summary of how data is being applied on each of the target servers, including the latency of each target server.
ats	Show a portion of each ATS file that is in text format.
atsdir	Show the names of the files in the ATS directory that are in text format and optionally run repair operations that are based on those files. If you are running this command as a DBSA, you must have read permission on the ATS files. Permissions on ATS files can be set with the **chown** operating system command.
ddr	Show the state, key log positions, and the proximity to transaction blocking for each server in the replication domain.
nif	Show information about the network connections between Enterprise Replication servers, including the number of transactions that are waiting to be transmitted to target servers.
profile	Show a summary of the state, data capture, data apply, errors, connectivity, queues, and the size of spooling files for every Enterprise Replication server.
rcv	Show information about the receive statistics for each target server, including the number of transaction failures and the rate at which transactions are applied.

Long Form	Meaning
ris	Show a portion of each RIS file that is in text format.
risdir	Show the names of the files in the RIS directory that are in text format and optionally run repair operations that are based on those files. If you are running this command as a DBSA, you must have read permission on the RIS files. Permissions on RIS files can be set with the **chown** operating system command.
sendq	Show information about the send queues for each Enterprise Replication server.
servers	Show information about the state, connection status to each peer server, and queue size for each Enterprise Replication server.
state	Show the Enterprise Replication state and the state of data capture, network connections, and data apply for each Enterprise Replication server.

The following table describes the **cdr view** options.

Long Form	Short Form	Meaning
--check	**-C**	Check the consistency between the database server and the ATS or RIS file. Send repair operations to stderr, but do not perform the repair operations.
--delete	**-d**	Delete ATS or RIS files after processing them with the repair operation.
--help	**-h**	Show the **cdr view** command usage.
–logstage	**-l**	Show log staging statistics.
--quiet	**-q**	Quiet mode. Repair operations are not sent to stderr.
--repair	**-R**	Synchronize data based on ATS or RIS files in text format.
--repeat=	**-r**	Repeat the **cdr view** command after the number of seconds specified by the *time* element.
--verbose	**-v**	Verbose mode (default). All repair operations are sent to stderr.

Usage

Use the **cdr view** command to monitor the Enterprise Replication domain. Each subcommand results in different output information.

You can choose to show the output of multiple subcommands sequentially by including them in the same **cdr view** command. You can choose to automatically repeat the command by using the **--repeat** option to specify the seconds in between commands.

You can repair inconsistencies that are listed in ATS or RIS files on every server by using the **--repair** option. Use the **--delete** option to delete the ATS or RIS files after the repair is complete.

Tip: Using the **--repair** option is equivalent to running the **cdr repair** command. The **--check** option is equivalent to the **cdr repair --check** command.

The cdr view state Command Output

The following example of the output of the **cdr view state** command shows the state of Enterprise Replication and each of its main components for every server in the Enterprise Replication domain.

```
STATE
Source      ER              Capture         Network         Apply
            State           State           State           State
--------------------------------------------------------------------------------
cdr1        Active          Running         Running         Running
cdr2        Active          Running         Running         Running
cdr3        Active          Running         Running         Running
cdr4        Active          Running         Running         Running
```

In this example, Enterprise Replication is active and running normally on all servers.

Possible values in the ER State column include:

Abort
: Enterprise Replication is aborting on this server.

Active
: Enterprise Replication is running normally.

Down
: Enterprise Replication is stopped.

Dropped
: The attempt to drop the **syscdr** database failed.

Init Failed
: The initial start of Enterprise Replication on this server failed, most likely because of a problem on the specified global catalog synchronization server.

Initializing
: Enterprise Replication is being defined.

Initial Startup
: Enterprise Replication is starting for the first time on this server.

Shutting Down
: Enterprise Replication is stopping on this server.

Startup Blocked
: Enterprise Replication cannot start because the server was started with the **oninit -D** command.

Synchronizing Catalogs
: The server is receiving a copy of the **syscdr** database.

Uninitialized
: The server does not have Enterprise Replication defined on it.

Possible values in the Capture State, Network State, and Apply State columns include:

Running
: The Enterprise Replication component is running normally.

Down
: The Enterprise Replication component is not running.

Uninitialized
: The server is not a source server for replication.

The cdr view profile Command Output

The following example of the output of the **cdr view profile** command shows a summary of the other **cdr view** commands and information about the sbspaces that are designated for spooled transaction data.

```
ER PROFILE for Node cdr2                ER State Active

DDR - Running                           SPOOL DISK USAGE
  Current        4:16879616               Total                100000
  Snoopy         4:16877344               Metadata Free          5025
  Replay         4:24                     Userdata Free         93193
  Pages from Log Lag State   43879
                                        RECVQ
SENDQ                                     Txn In Queue               0
  Txn In Queue               0            Txn In Pending List        0
  Txn Spooled                0
  Acks Pending               0          APPLY - Running
                                          Txn Processed           1838
NETWORK - Running                         Commit Rate            76.58
  Currently connected to 3 out of 3       Avg. Active Apply       1.16
  Msg Sent                1841            Fail Rate               0.00
  Msg Received            5710            Total Failures             0
  Throughput           1436.94            Avg Latency             0.00
  Pending Messages           0            Max Latency                0
                                          ATS File Count             0
                                          RIS File Count             0
```

In this example, only the output for a single server, **cdr2**, is shown. The actual output of the **cdr view profile** command includes a similar profile for every server.

The DDR section is a summary of the **cdr view ddr** command.

The SPOOL DISK USAGE section shows the total amount of memory, in bytes, in the sbspaces that Enterprise Replication uses to store spooled transaction row data, and the amount of available metadata and user data space.

The SENDQ section is a summary of the **cdr view sendq** command.

The RECVQ section is a summary of the **cdr view rcv** command.

The NETWORK section is a summary of the **cdr view nif** command.

The APPLY section is a summary of the **cdr view apply** command.

The cdr view ddr Command Output

The following example of the output of the **cdr view ddr** command shows the status of log capture.

```
Server   Snoopy       Replay       Current      total        log pages to   LogLag   Cur LogLag
         log page     log page     log page     log pages    LogLag State   State    Action
-------------------------------------------------------------------------------------------
g_bombay 16:133       16:0         16:134       30000        17866          Off       dlog
g_delhi  30:490       30:0         30:491       5000         3508           Off       logstage
```

The following example of the output of the **cdr view ddr -l** command shows the status of log capture.

```
Server   Disk Space   Max allowed  Max disk     Cur Staged
         Usage(%)     Space(KB)    ever used(KB) log file cnt
-------------------------------------------------------------------------------------------
g_bombay 0.00         0            0.00         0
g_delhi  0.00         1048576      0.00         0
```

The columns in the output of the **cdr view ddr** command provide the following information:

Server
: The name of the Enterprise Replication server.

Snoopy log page
: The current log ID and position at which transactions are being captured for replication.

Replay log page
: The current log ID and position at which transactions have been applied. This is the position from which the log would must be replayed to recover Enterprise Replication if Enterprise Replication or the database server shut down.

Current® log page
: The log page on which replicated transactions are being captured.

total log pages
: The total number of log pages on the server.

log pages to LogLag State
: The number of log pages that must be used before transaction blocking occurs.

LogLag State
: The state of DDR log lag: on or off.

Cur LogLag Action
: The action being taken to catch up logs.

For more information on interpreting this output, see onstat -g ddr: Print status of ER log reader.

The cdr view servers Command Output

The following example of the output of the **cdr view servers** command shows the state of the Enterprise Replication servers and their connections to each other.

```
SERVERS
Server Peer   ID   State    Status      Queue  Connection Changed
-------------------------------------------------------------------------
cdr1   cdr1   1    Active   Local       0
       cdr2   2    Active   Connected   0      Apr 15 10:46:16
       cdr3   3    Active   Connected   0      Apr 15 10:46:16
       cdr4   4    Active   Connected   0      Apr 15 10:46:15
cdr2   cdr1   1    Active   Connected   0      Apr 15 10:46:16
       cdr2   2    Active   Local       0
       cdr3   3    Active   Connected   0      Apr 15 10:46:16
       cdr4   4    Active   Connected   0      Apr 15 10:46:16
cdr3   cdr1   1    Active   Connected   0      Apr 15 10:46:16
       cdr2   2    Active   Connected   0      Apr 15 10:46:16
       cdr3   3    Active   Local       0
       cdr4   4    Active   Connected   0      Apr 15 10:46:16
cdr4   cdr1   1    Active   Connected   0      Apr 15 10:46:16
       cdr2   2    Active   Connected   0      Apr 15 10:46:16
       cdr3   3    Active   Connected   0      Apr 15 10:46:16
       cdr4   4    Active   Local       0
```

In this example, each of the four servers is connected to each other.

The output of this command is similar to the output of the **cdr list server** command, except that the **cdr view server** command shows all servers in the Enterprise Replication domain, not just the servers connected to the one from which the command is run. For information about the columns in this output, see cdr list server.

The cdr view sendq Command Output

The following example of the output of the **cdr view sendq** command shows information about the send queue for each server.

```
RQM SENDQ
Server     Trans.     Trans.     Trans.       Data     Memory       ACKS
           in que     in mem    spooled   in queue     in use    pending
-------------------------------------------------------------------------------
cdr1          594        594          0      49896      49896          0
cdr2            0          0          0          0          0          0
cdr3            0          0          0          0          0          0
cdr4            0          0          0          0          0          0
```

In this example, only the server **cdr1** has transactions in the send queue, all of which are in memory.

The columns of the **cdr view sendq** command provide the following information in addition to the server name:

Trans. in que
: The number of transactions in the send queue.

Trans. in mem
: The number of transactions in the send queue that are currently in memory.

Trans. spooled
: The number of transactions in the send queue that have been spooled to disk.

Data in queue
: The number of bytes of data in the send queue, including both in-memory and spooled transactions.

Memory in use
: The number of bytes of data in the send queue that resides in memory.

ACKS pending
: The number of acknowledgments that have been received but have not yet been processed.

The cdr view rcv Command Output

The following example of the output of the **cdr view rcv** command shows information about the receive queue for each server.

```
RCV
Server Received Spooled    Memory Pending Waiting
           Txn.    Txn.    In Use    Txn.    Txn.
-------------------------------------------------------------------------
cdr1          0       0         0       0       0
cdr2        372       0    871164     372       0
cdr3        220       0     18480     220       0
cdr4          0       0         0       0       0
```

In this example, the servers **cdr2** and **cdr3** have transactions in the receive queue, all of which have been preprocessed and are in the pending state waiting to be applied.

The columns of the **cdr view rcv** command provide the following information in addition to the server name:

Received Txn.
: The number of transactions in the receive queue.

Spooled Txn.
: The number of transactions in the receive queue that have been spooled to disk.

Memory In Use
: The size, in bytes, of the receive queue.

Pending Txn.
: The number of transactions that have been preprocessed but not yet applied.

Waiting Txn.
: The number of acknowledgments waiting to be sent back to the source server.

The cdr view apply Command Output

The following example of the output of the **cdr view apply** command shows how replicated data is being applied.

```
APPLY
Server   Pl  Failure       Num     Num   Apply  --Latency--  ATS  RIS
       Rate    Ratio       Run  Failed    Rate  Max    Avg.    #    #
------------------------------------------------------------------------
cdr1      0    0.000         0       0   0.000    0   0.000    0    0
cdr2      0    0.000     10001       0   0.112    0   0.000    0    0
cdr3      0    0.000     10001       0   0.112    0   0.000    0    0
cdr4      0    0.000     10001       0   0.112    0   0.000    0    0
```

In this example, the servers **cdr2**, **cdr3**, and **cdr4** each applied 10 001 transactions.

The columns of the **cdr view apply** command provide the following information in addition to the server name:

Pl Rate
: Indicates the degree of parallelism used when data is being applied. Zero indicates the highest possible rate of parallelism.

Failure Ratio
: The ratio of the number of times data could not be applied in parallel because of deadlocks or lock time outs.

Num Run
: The number of transactions processed.

Num Failed
: The number of failed transactions because of deadlocks or lock time outs.

Apply Rate
: The number of transactions that have been applied divided by the amount of time that replication has been active. The Apply Rate is equal to the Commit Rate in the **cdr view profile** command.

Max. Latency
: The maximum number of seconds for processing any transaction.

Avg. Latency
: The average number of seconds of the lifecycle of a replicated transaction.

ATS #
: The number of ATS files.

RIS #
: The number of RIS files.

The cdr view nif Command Output

The following example of the output of the **cdr view nif** command shows the status and statistics of connections between servers.

```
NIF
Source Peer   State          Messages  Messages Messages  Transmit
                                Sent  Received  Pending      Rate
------------------------------------------------------------------------
cdr1   cdr2   Connected         24014       372        6 21371.648
       cdr3   Connected         24020        17        0 20527.105
       cdr4   Connected         24014        23        6 21925.727
cdr2   cdr1   Connected           392     24015        0 21380.879
       cdr3   Connected            14        14        0    10.857
       cdr4   Connected            14        14        0    11.227
cdr3   cdr1   Connected            17     24021        0 20310.611
       cdr2   Connected            14        14        0    10.739
       cdr4   Connected            14        14        0    11.227
cdr4   cdr1   Connected           236     24015        0 21784.225
       cdr2   Connected            14        14        0    11.101
       cdr3   Connected            14        14        0    11.101
```

In this example, all servers are connected to each other. The server **cdr1** has six messages that have not yet been sent to server **cdr2** and server **cdr4**.

The columns of the **cdr view nif** command provide the following information in addition to the source server name:

Peer
: The name of the server to which the source server is connected.

State
: The connection state. Values include:

  Connected
  : The connection is active.

  Disconnected
  : The connection was explicitly disconnected.

  Timeout
  : The connection attempt has timed out, but will be reattempted.

  Logic error
  : The connection disconnected due to an error during message transmission.

  Start error
  : The connection disconnected due to an error while starting a thread to receive remote messages.

  Admin close

Enterprise Replication was stopped by a user issuing the **cdr stop** command.

Connecting
: The connection is being established.

Never Connected
: The servers have never had an active connection.

Messages Sent
: The number of messages sent from the source server to the target server.

Messages Received
: The number of messages received by the source server from the target server.

Messages Pending
: The number of messages that the source server must send to the target server.

Transmit Rate
: The total bytes of messages sent and received by the server divided by the amount of time that Enterprise Replication has been running. Same as the Throughput field in the **cdr view profile** command.

The cdr view ats and cdr view ris Command Output

The following example of the output of the **cdr view ats** command shows that there are no ATS files in text format.

```
ATS for cdr1 - no files
--------------------------------------------------------------------------------

ATS for cdr2 - no files
--------------------------------------------------------------------------------

ATS for cdr3 - no files
--------------------------------------------------------------------------------

ATS for cdr4 - no files
```

The following example of the **cdr view ris** command shows two RIS files in text format.

```
RIS for cdr1 - no files
--------------------------------------------------------------------------------

RIS for cdr2 - 1 files
Source Txn. Commit         Receive
       Time                Time
--------------------------------------------------------------------------------
cdr1   08-04-15 11:56:13 | 08-04-15 11:56:14
File:ris.cdr2.cdr1.D_4.080415_11:56:14.1

Row:2 / Replicate Id: 262146 / Table: stores_demo@user.customer / DbOp:Update
CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
--------------------------------------------------------------------------------

RIS for cdr3 - no files
--------------------------------------------------------------------------------

RIS for cdr4 - 1 files
Source Txn. Commit         Receive
       Time                Time
--------------------------------------------------------------------------------
cdr1   08-04-15 11:56:13 | 08-04-15 11:56:14
File:ris.cdr4.cdr1.D_1.080415_11:56:14.1

Row:3 / Replicate Id: 262146 / Table: stores_demo@user.customer / DbOp:Update
CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
```

In this example, the servers **cdr2** and **cdr4** each have one RIS file.

The cdr view atsdir and cdr view risdir Command Output

The **cdr view atsdir** command and **cdr view risdir** command outputs have the same format. The following example of the output of the **cdr view risdir** command shows the names of two RIS files.

```
RISDIR
Server File                                    Size              Create
       Name                                                        Time
--------------------------------------------------------------------------------
cdr2   ris.cdr2.cdr1.D_4.080415_11:56:14.1    465  2008-04-15 11:56:15
cdr4   ris.cdr4.cdr1.D_1.080415_11:56:14.1    475  2008-04-15 11:56:15
```

In this example, both server **cdr2** and server **cdr4** have a single RIS file. The Size column shows the size of the file, in bytes.

Examples

The following command would show information about the send queue and the network every 10 seconds:

```
cdr view sendq nif --repeat=10
```

The following command can be used in a daemon or script that runs every five minutes to check all servers for ATS and RIS files, repair inconsistencies, and delete the processed ATS and RIS files:

```
cdr view atsdir risdir --repair --delete --repeat=300
```

Related concepts:

Failed Transaction (ATS and RIS) Files
Monitor and troubleshooting Enterprise Replication
Preparing for Role Separation (UNIX)
Enterprise Replication Server administrator
Related reference:
cdr list server
cdr repair
onstat -g ddr: Print status of ER log reader

Enterprise Replication configuration parameter and environment variable reference

You can use configuration parameters and environment variables to configure the behavior of Enterprise Replication.

The database server **onconfig** configuration file includes the configuration parameters that affect the behavior of Enterprise Replication. If you use both the DBSERVERNAME and DBSERVERALIASES configuration parameters, the DBSERVERNAME configuration parameter must specify the network connection and not to a shared-memory connection. For information about database server aliases, see the *IBM® Informix® Administrator's Guide*.

Use the CDR_ENV configuration parameter to set the environment variables that affect the behavior of Enterprise Replication.

You can view the setting of Enterprise Replication configuration parameters and environment variables with the **onstat -g cdr config** command. See onstat -g cdr config: Print ER settings.

- CDR_APPLY Configuration Parameter
  Specifies the minimum and maximum number of data sync threads. The value is updated dynamically as needed.
- CDR_AUTO_DISCOVER configuration parameter
  Use the CDR_AUTO_DISCOVER configuration parameter to enable connectivity autoconfiguration for a high-availability cluster or Enterprise Replication domain, or to autoconfigure replication.
- CDR_DBSPACE Configuration Parameter
  Specifies the dbspace where the **syscdr** database is created.
- CDR_DELAY_PURGE_DTC configuration parameter
  Specifies how long to retain rows in delete tables to support the delete wins conflict resolution rule.
- CDR_DSLOCKWAIT Configuration Parameter
  Specifies the number of seconds the data sync component waits for the database locks to be released.
- CDR_ENV Configuration Parameter
  Sets the Enterprise Replication environment variables CDR_ALARMS, CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, or CDR_RMSCALEFACT.
- CDR_EVALTHREADS Configuration Parameter
  Specifies the number of group evaluator threads to create when Enterprise Replication starts, and enables parallelism.
- CDR_LOG_LAG_ACTION configuration parameter
  Specifies how Enterprise Replication responds to a potential log wrap situation.
- CDR_LOG_STAGING_MAXSIZE Configuration Parameter
  Specifies the maximum amount of space that Enterprise Replication uses to stage compressed log files in the directory specified by the LOG_STAGING_DIR configuration parameter.
- CDR_MAX_DYNAMIC_LOGS Configuration Parameter
  Specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.
- CDR_MAX_FLUSH_SIZE configuration parameter
  Specifies the maximum number of replicated transactions that are applied before the logs are flushed.
- CDR_MEM configuration parameter
  The CDR_MEM configuration parameter is used to specify Enterprise Replication's method for memory-pool allocation.
- CDR_NIFCOMPRESS Configuration Parameter
  Specifies the level of compression the database server uses before sending data from the source database server to the target database server.
- CDR_QDATA_SBSPACE Configuration Parameter
  Specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data.
- CDR_QUEUEMEM Configuration Parameter
  Specifies the maximum amount of memory that is used for the send and receive queues.
- CDR_SERIAL Configuration Parameter
  Enables control over generating values for serial columns in tables that are defined for replication.
- CDR_SUPPRESS_ATSRISWARN Configuration Parameter
  Specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files.
- CDR_TSINSTANCEID configuration parameter
  Specifies how to generate unique identifiers for time series instances across replication servers. If a replicate includes a column with a **TimeSeries** column, the CDR_TSINSTANCEID configuration parameter must be set to a different value on every participating replication server before you create any time series instances.
- ENCRYPT_CDR Configuration Parameter
  Use the ENCRYPT_CDR configuration parameter to set the level of encryption for Enterprise Replication.
- ENCRYPT_SMX Configuration Parameter
  Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high-availability configurations on secondary servers and between Enterprise Replication servers.
- GRIDCOPY_DIR Configuration Parameter
  Specifies the default directory used by the **ifx_grid_copy** procedure.
- SHARD_EDGE_NODE configuration parameter
  Specifies how to allocate shared memory for sharded queries on a shard server.
- SHARD_ID configuration parameter
  Sets the unique ID for a shard server in a shard cluster.
- SMX_COMPRESS Configuration Parameter
  Use the SMX_COMPRESS configuration parameter to specify the level of compression that the database server uses before sending data from the source database server to the target database server.

- SMX_NUMPIPES Configuration Parameter
  The SMX_NUMPIPES configuration parameter sets the number of pipes for server multiplexer group (SMX) connections.
- CDR_ALARMS Environment Variable
  Enables Enterprise Replication event alarms.
- CDR_ATSRISNAME_DELIM Environment Variable
  Specifies the delimiter to use to separate the parts of the time portion of ATS and RIS file names that are in text format.
- CDR_DISABLE_SPOOL Environment Variable
  Controls the generation of ATS and RIS files.
- CDR_LOGDELTA Environment Variable
  Determines when the send and receive queues are spooled to disk as a percentage of the logical log size.
- CDR_PERFLOG Environment Variable
  Enables queue tracing.
- CDR_RMSCALEFACT Environment Variable
  Sets the number of data sync threads started for each CPU VP.
- CDR_ROUTER Environment Variable
  Disables intermediate acknowledgments of transactions in the hierarchical topologies.
- CDRSITES_10X Environment Variable
  Works around a malfunction in version reporting for fix pack versions of 10.00 servers.
- CDRSITES_731 Environment Variable
  Works around a malfunction in version reporting for post-7.3x, 7.20x, or 7.24x version servers.
- CDRSITES_92X Environment Variable
  Works around a malfunction in version reporting for 9.21 or 9.20 servers.

Related concepts:
Configuring network encryption for replication servers
Related tasks:
Dynamically Modifying Configuration Parameters for a Replication Server
Related reference:
Set configuration parameters for replication

CDR_APPLY Configuration Parameter

Specifies the minimum and maximum number of data sync threads. The value is updated dynamically as needed.

onconfig.std value
: Not in the onconfig.std file.

range of values
: Two positive integers that are separated by a comma and that represent the minimum and maximum number of threads per CPU virtual processor. If only one value is specified, that value is the maximum and the minimum is set to 1.

If you do not set the CDR_APPLY configuration parameter explicitly, Enterprise Replication automatically allocates data sync threads for each CPU VP based on need:

- The minimum number of data sync threads is the number of available CPU VPs.
- The maximum number of data sync threads is four times the number of available CPU VPs.

You can set the CDR_APPLY configuration parameter to control the minimum and maximum configurable data sync apply threads irrespective of the CPU VP configuration.

CDR_AUTO_DISCOVER configuration parameter

Use the CDR_AUTO_DISCOVER configuration parameter to enable connectivity autoconfiguration for a high-availability cluster or Enterprise Replication domain, or to autoconfigure replication.

onconfig.std value
: CDR_AUTO_DISCOVER 0

default value if you created a server during installation
: CDR_AUTO_DISCOVER 1

values
: 0 = Disable connectivity and Enterprise Replication autoconfiguration.

  1 = Enable connectivity and Enterprise Replication autoconfiguration.

takes effect
: After you edit your onconfig file and restart the database server.
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
  When you reset the value in memory by running the **onmode -wm** command.
  After you run the SQL administration API **task()** or **admin()** function with the **-wf CDR_AUTO_DISCOVER=***value* or **-wm CDR_AUTO_DISCOVER=***value* arguments.

Usage

When the CDR_AUTO_DISCOVER configuration parameter is set to 1, the following commands are enabled:

- **cdr autoconfig serv**, which autoconfigures connectivity for servers in a high-availability cluster or Enterprise Replication domain, and can autoconfigure replication.

- **ifxclone** with the **--autoconf** option, which autoconfigures connectivity information between a newly added server and the other servers of a high-availability cluster or Enterprise Replication domain. If you use the **ifxclone** utility to create an Enterprise Replication server, the **--autoconf** option can autoconfigure replication.

Related tasks:
Preparing the Network Environment
Related reference:
cdr autoconfig serv
Related information:
cdr autoconfig serv argument: Autoconfigure connectivity and replication (SQL administration API)
The ifxclone utility

CDR_DBSPACE Configuration Parameter

Specifies the dbspace where the **syscdr** database is created.

onconfig.std value
: none
units
: any valid dbspace
takes effect
: When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_DBSPACE configuration parameter specifies the dbspace where the **syscdr** database is created.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the CDR_DBSPACE configuration parameter when defining a replication server. If the CDR_DBSPACE configuration parameter is not set and the database server has a storage pool with sufficient space, the **cdr define** command performs the following tasks:

- Creates a new dbspace using one or more new chunks from the storage pool
- Sets the CDR_DBSPACE configuration parameter both in memory and in the onconfig file to the newly defined dbspace.

For clusters, the **cdr define** command creates new dbspaces and sets the CDR_DBSPACE configuration parameters in all secondary database servers, as well.
Note: A database server's storage pool must have 200 MB of free space for the dbspace, and chunk sizes of 100 MB or greater for the database server to use automatic storage provisioning.
From 14.10xC6 onwards, row data for smaller transactions of size less than 26KB are stored in-line along with transaction header table trg_send_stxn. Row data for transaction size above 26KB is still stored in ER queue smart blob space configured using CDR_QDATA_SBSPACE configuration parameter. Make sure to have enough space allocated for storage space used to store trg_send_stxn table. trg_send_stxn table is created in space configured using CDR_DBSPACE configuration parameter. If CDR_DBSPACE is not configured, then rootdbs will be used for syscdr database and transaction header table.

CDR_DELAY_PURGE_DTC configuration parameter

Specifies how long to retain rows in delete tables to support the delete wins conflict resolution rule.

onconfig.std value
: 0
default value if not present in the onconfig
: 0
syntax
: CDR_DELAY_PURGE_DTC *timeunit*
range of values
: The range of values for *time* are 0 and positive integers.
The range of values for *unit* are:

  - S = seconds (Default)
  - M = minutes
  - H = hours
  - D = days

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

By default, rows in delete tables are deleted when those rows are no longer required by the timestamp conflict resolution rule. If you want to perform time stamp repair and your replicates use the delete wins conflict resolution rule, set the CDR_DELAY_PURGE_DTC configuration parameter to the maximum age of modifications to rows that are being actively updated. The longer you retain rows in delete tables, the more accurate time stamp repairs are, but the more disk space the delete tables consume.

Tip: Right before you enable a disabled server, dynamically update the CDR_DELAY_PURGE_DTC configuration parameter to set it to a value slightly greater than the time that the server was disabled plus the amount of time a repair takes.
Related concepts:
Repair inconsistencies by time stamp
Delete wins conflict resolution rule

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

CDR_DSLOCKWAIT Configuration Parameter

Specifies the number of seconds the data sync component waits for the database locks to be released.

onconfig.std value
: 5

units
: seconds

takes effect
: When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_DSLOCKWAIT configuration parameter specifies the number of seconds the data sync component waits for database locks to be released. The CDR_DSLOCKWAIT parameter behaves similarly to the SET LOCK MODE statement. Although the SET LOCK MODE is set by the end user application, CDR_DSLOCKWAIT is used by Enterprise Replication while applying data at the target database. This parameter is useful in conditions where different sources require locks on the replicated table. These sources could be a replicated transaction from another server or a local application operating on that table.

Transactions that receive updates and deletes from another server in the replicate can abort because of locking problems. If you experience transaction aborts in the data sync due to lock timeouts like this, you might want to increase the value of this parameter.

CDR_ENV Configuration Parameter

Sets the Enterprise Replication environment variables CDR_ALARMS, CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, or CDR_RMSCALEFACT.

Important: Use the CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, and CDR_RMSCALEFACT environment variables only if instructed to do so by Support.

units
: Enterprise Replication environment variable name and value, separated by an equal sign

takes effect
: When the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command

The onconfig file can contain multiple entries for the CDR_ENV environment variable. You can specify only one environment variable per CDR_ENV entry.

The following line in the onconfig file sets the CDR_ALARMS environment variable to add event alarm 51 to the event alarms that are enabled by default:

```
CDR_ENV CDR_ALARMS=30-39,47,48,50,51,71,73-75
```

When you update the CDR_ALARMS environment variable in the onconfig file, you must list all the Enterprise Replication event alarms that you want to be enabled.

The following lines in the onconfig file set the CDR_LOGDELTA environment variable to `30` and the CDR_ROUTER environment variable to `1`:

```
CDR_ENV CDR_LOGDELTA=30
CDR_ENV CDR_ROUTER=1
```

Related tasks:
Enabling or Disabling Enterprise Replication Event Alarms

CDR_EVALTHREADS Configuration Parameter

Specifies the number of group evaluator threads to create when Enterprise Replication starts, and enables parallelism.

onconfig.std value
: 1,2

units
: evaluator thread instances

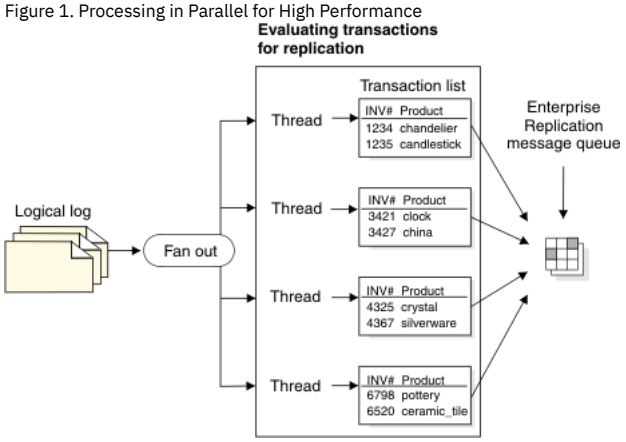
range of values
: first value: `0` or a positive integer representing the number of evaluator threads to create per CPU VP. Although evaluator threads are not assigned to specific CPU VPs, you can create evaluator threads that are proportional in number to the number of CPU VPs.
second value: `0` or a positive integer representing the additional number of evaluator threads to create irrespective of the number of CPU VPs.

Do not set both CDR_EVALTHREADS values to 0.

takes effect
: When the database server is shut down and restarted, or immediately after the **cdr change onconfig** command is used

Enterprise Replication evaluates the images of a row in parallel to assure high performance. Figure 1 illustrates how Enterprise Replication uses parallel processing to evaluate transactions for replication.

Figure 1. Processing in Parallel for High Performance



The CDR_EVALTHREADS configuration parameter specifies the number of grouper evaluator threads to create when Enterprise Replication starts and enables parallelism. The format is:

```
(per-cpu-vp,additional)
```

The following table provides four examples of CDR_EVALTHREADS.

Number of Threads	Explanation	Example
1,2	1 evaluator thread per CPU VP, plus 2	For a 3 CPU VP server: (3 * 1) + 2 = 5
2	2 evaluator threads per CPU VP	For a 3 CPU VP server: (3 * 2) = 6
2,0	2 evaluator threads per CPU VP	For a 3 CPU VP server: (3* 2) +0 = 6
0,4	4 evaluator threads for any database server	For a 3 CPU VP server: (3 * 0) +4 = 4

Attention: Do not configure the total number of evaluator threads to be smaller than the number of CPU VPs in the system. As noted above, do not set both CDR_EVALTHREADS values to 0.

CDR_LOG_LAG_ACTION configuration parameter

Specifies how Enterprise Replication responds to a potential log wrap situation.

onconfig.std value
: CDR_LOG_LAG_ACTION ddrblock

separators
: +

range of values
: See the Usage section.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

Use the CDR_LOG_LAG_ACTION configuration parameter to specify one or more actions, in priority order, that Enterprise Replication takes during a potential log wrap situation.

```
Syntax for the CDR_LOG_LAG_ACTION configuration parameter

>>-CDR_LOG_LAG_ACTION-------------------------------------------->

>--+-+-logstage--+---------+-+--+-------------+-+--------------><
   | |           '-+--dlog-' |  +-+--ignore---+ |
   | '-dlog--+-------------+-'  +-+--ddrblock-+ |
   |         '-+--logstage-'    '-+--shutdown-' |
   '-+-ignore---+--------------------------------'
     +-ddrblock-+
     '-shutdown-'
```

Table 1. Options for the CDR_LOG_LAG_ACTION configuration parameter value

Option	Description

Option	Description
logstage	Enables compressed logical log staging. The following configuration parameters must also be set: • The LOG_STAGING_DIR configuration parameter must be set to a directory. The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user informix, must belong to group informix, and must not have public read, write, or execute permission. • The CDR_LOG_STAGING_MAXSIZE configuration parameter must be set to a positive number. Log files are staged in the directory specified by the LOG_STAGING_DIR configuration parameter, until the maximum size specified by the CDR_LOG_STAGING_MAXSIZE configuration parameter is reached. The staged log files are deleted after advancing the log replay position. If the amount of disk space specified by the CDR_LOG_STAGING_MAXSIZE configuration parameter is exceeded, event alarm 30005 is raised. If log staging is configured, Enterprise Replication monitors the log lag state and stages log files even when Enterprise Replication is inactive.
dlog	Enables the dynamic addition of logical logs. The following configuration parameters must be set: • The CDR_MAX_DYNAMIC_LOGS configuration parameter must be set to -1 or a positive number. • The DYNAMIC_LOGS configuration parameter must be set to 2.
ignore	Ignore the potential for log wrapping. The Enterprise Replication replay position might be overrun. If the replay position is overrun, event alarm 30 is raised. Restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data. The **ignore** option must be the only or the last option. If the snoopy log position overrun is detected, Enterprise Replication shuts down with event alarm 47005.
ddrblock	Default. Block client applications update activity. The **ddrblock** option must be the only or the last option.
shutdown	Shut down Enterprise Replication on the affected server. If replay position overrun is detected, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data. If the replay position was not overrun, restart Enterprise Replication using the **cdr start** command; there is no need to synchronize the data. If replay position overrun is detected and the **cdr start** command fails with error code 214 and raises event alarm class 75, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data. The **shutdown** option must be the only or the last option. If a log lag state is detected, Enterprise Replication is shut down and event alarm ID 47006 is raised.

Staged log file format

Enterprise Replication creates a directory named: ifmxddrlog_*SERVERNUM* in the directory specified by the LOG_STAGING_DIR configuration parameter. Log file names are in the following format:

```
ifmxERDDRBLOCKUniqueLog_lf_used_loguniqueid.dat
```

Enterprise Replication also creates an empty token file for each staged log file. The token file is used to detect log files that are only partially written. If a token file is not found then Enterprise Replication treats the staged log file as partially written log file and deletes it. The token log file format is:

```
ifmxERDDRBLOCKUniqueLog_lf_used_loguniqueid
```

Transferring log files to a high-availability cluster secondary server when using ER

If your configuration consists of an HDR, RSS, or SDS secondary server configured as an Enterprise Replication node, transfer staged log files to the secondary server using the alarm program script. The staged log files are required by Enterprise Replication in case the primary server in a high-availability cluster fails and a secondary server takes over the role of the primary server.

Enterprise Replication raises alarm class ID 30 and unique ID 30006 when a log is staged to the log staging directory. Enterprise Replication raises alarm class ID 30 and unique ID 30007 after deleting a staged log file. Using these alarms, you can automate the transfer of staged log files to the high-availability cluster secondary server using the alarm program script.

Ensure that the directory under the directory specified the LOG_STAGING_DIR configuration parameter exists and is named using the format ifmxddrlog_*SERVERNUM*. The script copies the staged log files to ifmxddrlog_*SERVERNUM* and creates a token log file after copying the staged log file.

Example

Suppose that you want Enterprise Replication to handle potential log wrap situations by first staging compressed logs until they reach 1 MB in size, then dynamically add up to two logical logs, and then block user transactions. Set the following configuration parameters:

```
CDR_LOG_LAG_ACTION      logstage+dlog+ddrblock
LOG_STAGING_DIR         $INFORMIXDIR/tmp
CDR_LOG_STAGING_MAXSIZE 1MB
CDR_MAX_DYNAMIC_LOGS    2
DYNAMIC_LOGS            2
```

Related concepts:
Handle potential log wrapping
Related information:
LOG_STAGING_DIR configuration parameter

CDR_LOG_STAGING_MAXSIZE Configuration Parameter

Specifies the maximum amount of space that Enterprise Replication uses to stage compressed log files in the directory specified by the LOG_STAGING_DIR configuration parameter.

default value
: 0

onconfig.std value
: CDR_LOG_STAGING_MAXSIZE 0

syntax
: CDR_LOG_STAGING_MAXSIZE *sizeunit*

range of values
: The range of values for *size* is:

  - 0 = Default. Log staging is disabled.
  - Positive integers = The maximum size of the stage log files.

  The range of value for *unit* is:

  - KB (default)
  - MB
  - GB
  - TB

takes effect
: After you edit your onconfig file and restart the database server.
  When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
  When you reset the value in memory by running the **onmode -wm** command.

Use the CDR_LOG_STAGING_MAXSIZE configuration parameter to limit the size of the log staging directory. Logs are staged if all of the following conditions are true:

- Enterprise Replication detects a potential for log wrapping.
- The CDR_LOG_LAG_ACTION configuration parameter setting includes the **logstage** option.
- The LOG_STAGING_DIR configuration parameter is set.
  The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user informix, must belong to group informix, and must not have public read, write, or execute permission.

When the contents of the staging directory reaches the maximum allowed size, Enterprise Replication stops staging log files. Enterprise Replication stops staging files only at a log file boundary; that is, a file is not staged in the middle of a log file.

Example

Suppose that you want Enterprise Replication to handle potential log wrap situations by staging compressed logs until the staging directory reached 100 KB, you would set the following configuration parameters:

```
CDR_LOG_STAGING_MAXSIZE 100
CDR_LOG_LAG_ACTION      logstage
LOG_STAGING_DIR         $INFORMIXDIR/tmp
```

Related concepts:
Handle potential log wrapping
Related information:
LOG_STAGING_DIR configuration parameter
onmode -wf, -wm: Dynamically change certain configuration parameters

CDR_MAX_DYNAMIC_LOGS Configuration Parameter

Specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.

onconfig.std value
: 0

range of values

- -1 add dynamic log files indefinitely
- 0 disable dynamic log addition
- >0 number of dynamic logs that can be added

takes effect
: when the database server is shut down and restarted, and the DYNAMIC_LOGS configuration parameter is set to 2 or when the **cdr change onconfig** command is used. For more information on the DYNAMIC_LOGS configuration parameter, see the *IBM® Informix® Administrator's Reference*.

The CDR_MAX_DYNAMIC_LOGS configuration parameter specifies the number of dynamic log file requests that Enterprise Replication can make in one server session. The DYNAMIC_LOGS configuration parameter must be set to 2.

Related concepts:
Handle potential log wrapping
Related information:
DYNAMIC_LOGS configuration parameter

CDR_MAX_FLUSH_SIZE configuration parameter

Specifies the maximum number of replicated transactions that are applied before the logs are flushed.

onconfig.std value
: CDR_MAX_FLUSH_SIZE 50

default value if not present in the onconfig file
: 50

range of values
: A positive integer that represents the maximum number of transactions to apply before the logs are flushed.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.

By default, replication servers flush logs after 50 replicated transactions are applied, or after 5 seconds, whichever happens first.

If a replication server is a primary server for shared-disk secondary servers, you might want to reduce the replication latency. Set the CDR_MAX_FLUSH_SIZE configuration parameter to 1 to flush the logs after each replicated transaction.

Related concepts:
Replication latency for secondary servers
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

CDR_MEM configuration parameter

The CDR_MEM configuration parameter is used to specify Enterprise Replication's method for memory-pool allocation.

onconfig.std value
: CDR_MEM 0

values
: `0`: Memory allocation from the generic pool is taken from the CDR pool. Memory allocation from the RQM pool is taken from the queue's memory pool.
: `1`: Memory allocation pools are associated with specific CPU virtual processors. Enterprise Replication allocates memory to the CPU virtual processors based on which CPU virtual processor the cdr thread is executing on.
: `2`: Memory allocation pools are associated with specific block sizes, so that all allocations from a pool are the same size, and the first free block that is found can be used.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.
: After you run the SQL administration API **task()** or **admin()** function with the `"onmode","-wf CDR_MEM=`*value*`"` or `"onmode","-wm CDR_MEM=`*value*`"` argument.

Usage

`CDR_MEM 0` is the traditional method of memory-allocation. Use this setting when resource allocation is more important than performance.

`CDR_MEM 1` prevents multiple threads from simultaneously accessing a memory pool. The performance of large-scale Enterprise Replication environments can improve, because memory allocation is done by multiple threads that are working in parallel.

`CDR_MEM 2` improves performance at the cost of increased memory usage. Memory allocation requests are increased to the closest fixed-block size, so that free memory blocks can be found faster. Memory pools are not associated with specific CPU virtual processors, so memory can be freed directly to the memory pool.

CDR_NIFCOMPRESS Configuration Parameter

Specifies the level of compression the database server uses before sending data from the source database server to the target database server.

onconfig.std value
: 0

range of values

- `-1` specifies no compression
- `0` specifies to compress only if the target server expects compression
- `1` - `9` specifies increasing levels of compression

takes effect
When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_NIFCOMPRESS (network interface compression) configuration parameter specifies the level of compression that the database server uses before sending data from the source database server to the target database server. Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data.

The values have the following meanings.

Value	Meaning
`-1`	The source database server never compresses the data, regardless of whether or not the target site uses compression.
0	The source database server compresses the data only if the target database server expects compressed data.
1	The database server performs a minimum amount of compression.
9	The database server performs the maximum possible compression.

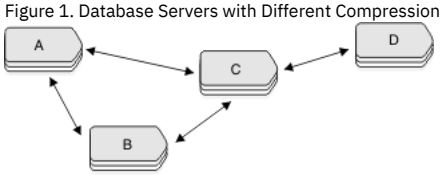
When Enterprise Replication is defined between two database servers, the CDR_NIFCOMPRESS values of the two servers are compared and changed to the higher compression values.

The compression values determine how much memory can be used to store information while compressing, as follows:

```
0 = no additional memory
1 = 128k + 1k    = 129k
2 = 128k + 2k    = 130k
...
6 = 128k + 32k   = 160k
...
8 = 128k + 128k = 256k
9 = 128k + 256k = 384k
```

Higher levels of CDR_NIFCOMPRESS cause greater compression.

Different sites can have different levels. For example, Figure 1 shows a set of three root servers connected with LAN and a nonroot server connected over a modem. The CDR_NIFCOMPRESS configuration parameter is set so that connections between A, B, and C use no compression. The connection from C to D uses level 6.
Figure 1. Database Servers with Different Compression Levels



Important: Do not disable NIF compression if the network link performs compression in hardware.
Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers and SMX_COMPRESS shall be used to enable compression. For communicating with older server versions before 14.10xC6, Enterprise Replication still require configuring CDR_NIFCOMPRESS.

CDR_QDATA_SBSPACE Configuration Parameter

Specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data.

onconfig.std value
none
separators
comma
range of values
up to 128 characters for each sbspace name; up to 32 sbspace names. Use a comma to separate each name in the list. At least one sbspace name must be specified.
takes effect
when the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

The CDR_QDATA_SBSPACE configuration parameter specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data. Enterprise Replication creates one smart large object per transaction. The sbspaces must be used only for Enterprise Replication. If CDR_QDATA_SBSPACE is configured for multiple sbspaces, then Enterprise Replication uses all appropriate sbspaces in round-robin order.

You can have Enterprise Replication automatically configure disk space from the storage pool and set the CDR_QDATA_SBSPACE configuration parameter when defining a replication server. If the CDR_QDATA_SBSPACE configuration parameter is not set and the database server has a storage pool with sufficient space, the **cdr define server** command automatically creates the necessary disk space and sets the configuration parameter to the appropriate value in memory and the onconfig file. For clusters, the **cdr define** command creates new sbspaces and sets the CDR_QDATA_SBSPACE configuration parameters in all secondary database servers, as well.
Note: A database server's storage pool must have 500 MB of free space for the sbspace. The sbspace must be comprised of chunks of size 100 MB or greater for the database server to use automatic storage provisioning.

Warning: Do not change the value of CDR_QDATA_SBSPACE while Enterprise Replication is running.
Related concepts:
Row Data sbspaces
Related tasks:
Monitoring Disk Usage for Send and Receive Queue Spool
Related reference:
cdr start sec2er

CDR_QUEUEMEM Configuration Parameter

Specifies the maximum amount of memory that is used for the send and receive queues.

onconfig.std value
: 131072

units
: kilobytes

range of values
: From 500 through 4194304

takes effect
: When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_QUEUEMEM configuration parameter specifies the maximum amount of memory that the send and receive queues use for transaction headers and for transaction data. The total size of the transaction headers and transaction data in a send or receive queue could be up to twice the size of that value of CDR_QUEUEMEM. If your logical logs are large, the Enterprise Replication reads a large amount of data into queues in memory. You can use CDR_QUEUEMEM to limit the amount of memory devoted to the queues.

When you increase the value of CDR_QUEUEMEM, you reduce the number of elements that must be written to disk, which can eliminate I/O overhead. Therefore, if elements are frequently stored on disk, increase the value of CDR_QUEUEMEM. Conversely, if you set the value of CDR_QUEUEMEM too high, you might adversely impact the performance of your system. High values for CDR_QUEUEMEM also increase the time necessary for recovery. Tune the value of CDR_QUEUEMEM for the amount of memory available on your computer.

CDR_SERIAL Configuration Parameter

Enables control over generating values for serial columns in tables that are defined for replication.

onconfig.std value
: CDR_SERIAL 0

range of values
: 0 = Default. Disable control of serial column value generation.
delta,*offset* = Enable control of serial column value generation:

delta
: A positive integer that sets the incremental size of the serial column values. This value must be the same on all replication servers and must be at least the number of expected servers in the Enterprise Replication domain.

offset
: A positive integer that sets the offset of the serial value to be generated. This value must be different on all replication servers and must be between 0 and one less than the value of *delta*, inclusive.

takes effect
: After you edit your onconfig file and restart the database server.
After you run the **cdr change onconfig** command.

The CDR_SERIAL configuration parameter controls generating values for SERIAL, SERIAL8, and BIGSERIAL columns in replicated tables so that no conflicting values are generated across multiple Enterprise Replication servers. You must set the CDR_SERIAL configuration parameter if the serial column is the replication key column and no other replication key column, such as a server ID, guarantees the uniqueness of the replication key. If the serial column is not the replication key, you can set the CDR_SERIAL configuration parameter to ensure that the serial values are unique across all servers. Only tables that are marked as the source of a replicate are controlled by the CDR_SERIAL configuration parameter settings.

For example, suppose that you have two primary servers, **g_usa** and **g_japan**, and one read-only target server, **g_italy**. You plan to add three more servers in the future. You might set CDR_SERIAL to the values shown in the following table.

Table 1. CDR_SERIAL Example Settings and Results

Server	Example CDR_SERIAL Value	Resulting Values for the Serial Column
g_usa	5,0	5, 10, 15, 20, 25, and so on
g_japan	5,1	1, 6, 11, 16, 21, 26, and so on
g_italy	0	no local inserts into the serial column

The following CDR_SERIAL settings are reserved for future servers:

- 5,2
- 5,3

- 5,4

If you must add more servers than the *delta* value of CDR_SERIAL, you must reset CDR_SERIAL on all servers simultaneously and ensure that the serial values on the new servers are unique.

Related concepts:
Serial data types and replication keys

CDR_SUPPRESS_ATSRISWARN Configuration Parameter

Specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files.

onconfig.std value
: none

units
: numbers or hyphen-separated ranges of numbers

separator
: commas

takes effect
: when the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

The CDR_SUPPRESS_ATSRISWARN configuration parameter specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files. For example, you can set CDR_SUPPRESS_ATSRISWARN to 2-5, 7 to suppress the generation of error and warning messages 2, 3, 4, 5, and 7. For a list of error and message numbers see Data sync warning and error messages.

CDR_TSINSTANCEID configuration parameter

Specifies how to generate unique identifiers for time series instances across replication servers. If a replicate includes a column with a **TimeSeries** column, the CDR_TSINSTANCEID configuration parameter must be set to a different value on every participating replication server before you create any time series instances.

onconfig.std value
: CDR_TSINSTANCEID 0

range of values
: 0 = Default. Disable the replication of **TimeSeries** columns.
: 1 - 32768 = The number that is added to the time series instance identifiers.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

You set the value of CDR_TSINSTANCEID to a different value on each replication server that replicates a **TimeSeries** column. A time series instance identifier is automatically generated when you create a time series instance. The unique values of CDR_TSINSTANCEID configuration parameter on each replication server ensures that no time series instance identifiers overlap in the replication domain.

Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

ENCRYPT_CDR Configuration Parameter

Use the ENCRYPT_CDR configuration parameter to set the level of encryption for Enterprise Replication.

onconfig.std value
: ENCRYPT_CDR 0

values
: 0 = Default. Do not encrypt.
: 1 = Encrypt when possible. Encryption is used for Enterprise Replication transactions only when the database server being connected to also supports encryption.

: 2 = Always encrypt. Only connections to encrypted database servers are allowed.

takes effect
: After you edit your onconfig file and restart the database server.
: After you run the **cdr change onconfig** command.

Usage

If you enable encryption with the ENCRYPT_CDR configuration parameter, you must also set the ENCRYPT_MAC, ENCRYPT_MACFILE, ENCRYPT_SWITCH, and ENCRYPT_CIPHERS configuration parameter to configure encryption.

If you use both encryption and compression (by setting the CDR_NIFCOMPRESS configuration parameter), then compression occurs before encryption.

Note:
From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers and ENCRYPT_SMX or onsocssl shall be used to enable encryption. For communicating with older server versions before 14.10xC6, Enterprise Replication still requires configuring ENCRYPT_CDR.

Related information:
ENCRYPT_CIPHERS configuration parameter
ENCRYPT_MACFILE configuration parameter
ENCRYPT_SWITCH configuration parameter
ENCRYPT_MAC configuration parameter

ENCRYPT_SMX configuration parameter

Use the ENCRYPT_SMX configuration parameter to set the level of encryption for high-availability configurations on secondary servers and between Enterprise Replication Servers.

onconfig.std value
: Not set.

values
: 0 = Off. Do not encrypt.
1 = On. Encrypt where possible. Encrypt SMX transactions when the database server being connected to also supports encryption.
2 = On. Always encrypt. Only connections to encrypted database servers are allowed.

takes effect
: After you edit your onconfig file and restart the database server.

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.
Related information:
Set the wait time for SMX activity between servers
Using High-Availability Clusters with Enterprise Replication

GRIDCOPY_DIR Configuration Parameter

Specifies the default directory used by the **ifx_grid_copy** procedure.

onconfig.std value
: $INFORMIXDIR

default value if not present in the onconfig file
: $INFORMIXDIR

values
: *pathname* = $INFORMIXDIR or a valid file path that is relative to $INFORMIXDIR.

takes effect
: After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

The **ifx_grid_copy()** procedure copies files from a grid database server to the other nodes of the same grid. The GRIDCOPY_DIR value is the default directory of file paths in the **ifx_grid_copy()** command:

- On a database server running the **ifx_grid_copy()** procedure, the GRIDCOPY_DIR value and the **ifx_grid_copy()** command *source_path_and_filename* is the location from which files are copied.
- On a database server sharing the same grid with a node running the **ifx_grid_copy()** procedure, the GRIDCOPY_DIR value and the **ifx_grid_copy()** command's *target_path_and_filename* is the location to which files are copied. If *target_path_and_filename* is not specified as part of the **ifx_grid_copy()** command, the GRIDCOPY_DIR value and the **ifx_grid_copy()** command's *source_path_and_filename* is the location to which files are copied.

If a node directory specified by a GRIDCOPY_DIR value does not exist, the node directory is created by the **ifx_grid_copy()** procedure.

Example

To specify $INFORMIXDIR/usr/informix/copydir as a node's default directory for **ifx_grid_copy()** procedure actions, set the following value in the onconfig file:

```
GRIDCOPY_DIR usr/informix/copydir
```

Related tasks:
Propagating external files through a grid
Related reference:

ifx_grid_copy() procedure

SHARD_EDGE_NODE configuration parameter

Specifies how to allocate shared memory for sharded queries on a shard server.

onconfig.std value
SHARD_EDGE_NODE
range of values
0: Default Value. Disable shard edge node functionality.
1: Enable shard edge node functionality.
takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Usage

SHARD_EDGE_NODE configuration parameter controls the shard edge server behavior.

SHARD_ID configuration parameter

Sets the unique ID for a shard server in a shard cluster.

onconfig.std value
SHARD_ID 0
range of values
0 = Default. The database server cannot run parallel sharded queries.
1 - 65535 = The unique ID of the shard server.
takes effect
After you edit your onconfig file and restart the database server.
If the value is 0 or not set, you can set the value dynamically in your onconfig file by running the **onmode -wf** command.

You set the value of the SHARD_ID configuration parameter to a different number on each shard server in a shard cluster. If the value of the SHARD_ID configuration parameter is unset or set to 0 on all shard servers in the shard cluster, the shard cluster performs poorly. If the values of the SHARD_ID configuration parameter are not unique on all shard servers in a shard cluster, shard queries fail.

To reset the value if the SHARD_ID configuration parameter is set to a positive integer, edit the onconfig file and then restart the database server.

Related tasks:
Creating a shard cluster
Related information:
onmode -wf, -wm: Dynamically change certain configuration parameters

SMX_COMPRESS configuration parameter

Use the SMX_COMPRESS configuration parameter to specify the level of compression that the database server uses before sending data from the source database server to the target database server.

Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data. The SMX_COMPRESS configuration parameter values of the two servers are compared and changed to the higher compression values.

onconfig.std value
SMX_COMPRESS 0
values
-1 = The source database server never compresses the data, regardless of whether or not the target site uses compression.
0 = The source database server compresses the data only if the target database server expects compressed data.

1 = The database server performs a minimum amount of compression.

9 = The database server performs the maximum possible compression.

takes effect
After you edit your onconfig file and restart the database server.
When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
When you reset the value in memory by running the **onmode -wm** command.

Note: From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.

Related reference:
onmode -wf, -wm: Dynamically change certain configuration parameters

SMX_NUMPIPES configuration parameter

The SMX_NUMPIPES configuration parameter sets the number of pipes for server multiplexer group (SMX) connections.

onconfig.std value
: SMX_NUMPIPES 1

values
: 1 - 32767 = The number of network pipes for SMX connections.

takes effect
: After you edit your onconfig file and restart the database server.
: When you reset the value dynamically in your onconfig file by running the **onmode -wf** command.
: When you reset the value in memory by running the **onmode -wm** command.

Usage

High-availability clusters and parallel sharded queries use SMX connections. If the lag time between servers is too long, increase the number of SMX pipes.

Note:
From 14.10xC6 onwards, Enterprise Replication uses SMX connection for communicating with peer servers.

CDR_ALARMS Environment Variable

Enables Enterprise Replication event alarms.

default value
: 30-39,47,48,50,71,73-75

range of values
: integers in these ranges: 30-39, 47-71, 73-75

separator
: comma (,) to separate individual numbers or hyphen (-) to separate a range of numbers

takes effect
: When the database server is shut down and restarted.

Set the CDR_ALARMS environment variable to the Enterprise Replication event alarms that you want to receive. Enterprise Replication event alarms that are not set by CDR_ALARMS are disabled.

Use the CDR_ENV configuration parameter to set this environment variable in the onconfig file.

Related tasks:
Enabling or Disabling Enterprise Replication Event Alarms

CDR_ATSRISNAME_DELIM Environment Variable

Specifies the delimiter to use to separate the parts of the time portion of ATS and RIS file names that are in text format.

default value
: On UNIX: a colon (:)
: On Windows: a period (.)

range of values
: a single character

takes effect
: when Enterprise Replication is initialized

ATS and RIS files in XML format always use a period (.) as the delimiter.

For example, the default file name for an ATS file in text format on UNIX might look like this: **ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6**. If CDR_ATSRISNAME_DELIM is set to a period (.), then the same file name would look like this: **ats.g_beijing.g_amsterdam.D_2.000529_23.27.16.6**.

Related concepts:
ATS and RIS File Names

CDR_DISABLE_SPOOL Environment Variable

Controls the generation of ATS and RIS files.

default value
0
range of values
0 Allow ATS and RIS file generation
1 Prevent ATS and RIS file generation

takes effect
when Enterprise Replication is initialized

The CDR_DISABLE_SPOOL environment variable controls whether ATS and RIS files are generated. Set CDR_DISABLE_SPOOL to 1 if you do not want ATS or RIS files to be generated under any circumstances.

Related concepts:
Failed Transaction (ATS and RIS) Files
Related tasks:
Disabling ATS and RIS File Generation

CDR_LOGDELTA Environment Variable

Determines when the send and receive queues are spooled to disk as a percentage of the logical log size.

default value
30
range of values
positive numbers
takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_LOGDELTA environment variable determines when the send and receive queues are spooled to disk as a percentage of the logical log size. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter.
Important: Do not use the CDR_LOGDELTA environment variable unless instructed to do so by Technical Support.

CDR_PERFLOG Environment Variable

Enables queue tracing.

default value
0
range of values
positive number
takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_PERFLOG environment variable enables queue tracing. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter.
Important: Do not use the CDR_PERFLOG environment variable unless instructed to do so by Technical Support.

CDR_RMSCALEFACT Environment Variable

Sets the number of data sync threads started for each CPU VP.

default value
4
range of values
positive number
takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_RMSCALEFACT environment variable sets the number of data sync threads started for each CPU VP. Specifying a large number of threads can result in wasted resources. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter.
Important: Do not use the CDR_RMSCALEFACT environment variable unless instructed to do so by Support.

CDR_ROUTER Environment Variable

Disables intermediate acknowledgments of transactions in the hierarchical topologies.

default value
: 0

range of values
: any number

takes effect
: when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

When set to 1, the CDR_ROUTER environment variable disables intermediate acknowledgments of transactions in hierarchical topologies. The normal behavior for intermediate servers is to send acknowledgments if they receive an acknowledgment from the next server in the replication tree (can be a leaf server) or if the transaction is stored in the local queue. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see CDR_ENV Configuration Parameter.

If CDR_ROUTER is set at the hub server, an acknowledgment will be sent only if the hub server receives acknowledgment from all of its leaf servers. Transactions will not be acknowledged even if they are stored in the local queue of the hub server.

If CDR_ROUTER is not set at hub server, the hub server will send an acknowledgment if the transaction is stored in the local queue at the hub server or if the hub server received acknowledgment from all of its leaf servers.

Important: Do not use the CDR_ROUTER environment variable unless instructed to do so by Technical Support.

CDRSITES_10X Environment Variable

Works around a malfunction in version reporting for fix pack versions of 10.00 servers.

units
: ***cdrIDs***, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values
: positive numbers

takes effect
: when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_10X** in the ONCONFIG file, or immediately for the following actions:

  - Adding a value using the **cdr add onconfig** command
  - Removing a value using the **cdr remove onconfig** command
  - Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve Versions 10.00.xC1 or 10.00.xC3 servers, the NIF does not properly report its version when it responds to a new server with a fix pack version of 10.00.xC4 or later. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server.

To prevent this malfunction, if you have Version 10.00.xC1 or 10.00.xC3 servers in your Enterprise Replication environment, set the **CDRSITES_10X** environment variable with the CDR_ENV configuration parameter for these servers.

Note: You can only set the **CDRSITES_10X** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_10X** as a standard environment variable.
The ***cdrID*** is the unique identifier for the database server in the Options field of the SQLHOSTS file (**i = *unique_ID***).

For example, suppose that you have 5 database servers, Version 10.00.xC1, whose ***cdrID*** values range from 2 through 10 (***cdrID*** = 2, 3, 8, 9, and 10).

If you upgrade database server ***cdrID*** 8 to Version 10.00.xC4, you must set the **CDRSITES_10X** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.00.xC4 database server online:

```
CDR_ENV CDRSITES_10x=2,3,9,10
```


CDRSITES_731 Environment Variable

Works around a malfunction in version reporting for post-7.3x, 7.20x, or 7.24x version servers.

units
: ***cdrIDs***, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values
: positive numbers

takes effect
: when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_731** in the ONCONFIG file, or immediately for the following actions:

  - Adding a value using the **cdr add onconfig** command
  - Removing a value using the **cdr remove onconfig** command
  - Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve post 7.3x, 7.20x, or 7.24x servers, the NIF does not properly report its version when it responds to a new server. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server. If a 10.0, 9.40, or 9.30 server tries to synchronize with a 7.3x, 7.20x, or 7.24x server, the older server responds to the 10.0, 9.40, or 9.30 server that it is a 10.0, 9.40, or 9.30 server and will subsequently fail.

To prevent this malfunction, if you have Version 7.3x, 7.20x, or 7.24x servers in your Enterprise Replication environment, set the **CDRSITES_731** environment variable with the CDR_ENV configuration parameter for these servers.

Note: You can only set the **CDRSITES_731** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_731** as a standard environment variable.
For example, suppose that you have 5 database servers, Version 7x servers whose ***cdrID*** values range from 1 through 7 (***cdrID*** = 1, 4, 5, 6, and 7).

If you upgrade database server ***cdrID*** 6 to Version 10.0, 9.40, or 9.30, you must set the **CDRSITES_731** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.0, 9.40, or 9.30 database server online:

```
CDR_ENV CDRSITES_731=1,4,5,7
```


CDRSITES_92X Environment Variable

Works around a malfunction in version reporting for 9.21 or 9.20 servers.

units
: ***cdrIDs***, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values
: positive numbers

takes effect
: when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_92X** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve 9.21 or 9.20 servers, the NIF does not properly report its version when it responds to a new server. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server. If a 10.0/9.40/9.30 server tries to synchronize with a 9.21 or 9.20 server, the older server responds to the 10.0, 9.40, or 9.30 server that it is a 10.0, 9.40, or 9.30 server and will subsequently fail.

To prevent this malfunction, if you have Version 9.21 or 9.20 servers in your Enterprise Replication environment, set the **CDRSITES_92X** environment variable

Note: You can only set the **CDRSITES_92X** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_92X** as a standard environment variable.
For example, suppose that you have 5 database servers, Version 9.21 or 9.20 whose ***cdrID*** values range from 2 through 10 *(cdrIDs* = 2, 3, 8, 9, and 10).

If you upgrade database server ***cdrID*** 8 to Version 10.0, 9.40, or 9.30, you must set the **CDRSITES_92X** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.0, 9.40, or 9.30 database server online:

```
CDR_ENV CDRSITES_92x=2,3,9,10
```


Grid routines

Grid routines are used to create and maintain the grid and to administer servers in the grid by propagating commands from a source server to all other servers in the grid.

- ifx_get_erstate() function
  The **ifx_get_erstate()** function indicates whether replication is enabled for the transaction in which it is run.
- ifx_grid_connect() procedure
  The **ifx_grid_connect()** procedure opens a connection to the grid. Through an **ifx_grid_connect()** procedure, you can run routines and data definition language (DDL) commands on a source server, and then propagate the routines or commands to the other grid servers.
- ifx_grid_copy() procedure
  The **ifx_grid_copy()** procedure copies non-database, external files from a grid database server to other nodes in the same grid.
- ifx_grid_disconnect() procedure
  The **ifx_grid_disconnect()** procedure closes a connection to the grid.
- ifx_grid_execute() procedure
  The **ifx_grid_execute()** procedure propagates the execution of a routine or data manipulation language (DML) SQL statement to all servers in the grid.
- ifx_grid_function() function
  The **ifx_grid_function()** function propagates the execution of a function to all servers in the grid.
- ifx_grid_procedure() procedure
  The **ifx_grid_procedure()** procedure propagates the execution of a procedure to all servers in the grid.
- ifx_grid_redo() procedure
  The **ifx_grid_redo()** procedure reruns commands that were run through the grid and failed on one or more servers in the grid.
- ifx_grid_release() function
  The **ifx_grid_release()** function propagates deferred DDL statements that were run on the local grid server, but deferred from running on the other grid servers.

- ifx_grid_remove() function
  The **ifx_grid_remove()** function removes any DDL statements that are deferred from propagation to grid servers.
- ifx_grid_purge() procedure
  The **ifx_grid_purge()** procedure deletes metadata about commands that have been run through the grid.
- ifx_gridquery_skipped_nodes() function
  The **ifx_gridquery_skipped_nodes()** function returns the name of a server that was unavailable during a grid or shard query.
- ifx_gridquery_skipped_node_count() function
  The **ifx_gridquery_skipped_node_count()** function returns the number of servers that were unavailable during a grid or shard query.
- ifx_node_id() function
  The **ifx_node_id()** function returns the ID of the grid server on which the function is run.
- ifx_node_name() function
  The **ifx_node_name()** function returns the name of the grid server on which the function is run.

ifx_get_erstate() function

The **ifx_get_erstate()** function indicates whether replication is enabled for the transaction in which it is run.

Syntax

```
>>-EXECUTE FUNCTION--ifx_get_erstate--(--)--INTO--data_var--;-->
```

Element	Purpose	Restriction
data_var	Variable to receive the value that the function returns	

Usage

Use the **ifx_get_erstate()** function to obtain the state of replication within a transaction. You can use the state information saved in the variable as input to the **ifx_set_erstate()** procedure.

Return value

A return value of 1 indicates that the current transaction is replicating data.

A return value of 0 indicates that the current transaction is not replicating data.

Example

The following example obtains the replication state and stores it in the curstate variable:

```
EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
```

Related concepts:
Recapture replicated transactions
Related tasks:
Enabling replication within a grid transaction
Related reference:
ifx_set_erstate() procedure

ifx_grid_connect() procedure

The **ifx_grid_connect()** procedure opens a connection to the grid. Through an **ifx_grid_connect()** procedure, you can run routines and data definition language (DDL) commands on a source server, and then propagate the routines or commands to the other grid servers.

Syntax

```
>>-EXECUTE PROCEDURE--ifx_grid_connect--(--'--grid_name--'------>

>--+-+--------------+-+--------------+-+--)--;-----------------><
   | '-,--'--tag--'-' '-,--ER_enable-' |
   '-,--'--tag--'--,--defer------------'
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.

Element	Purpose	Restrictions
ER_enable	Enable or disable the creation of a replicate and replicate set and starting replication for any tables that are created while the connection to the grid is open. Optionally suppress any errors that might be raised when the procedure is run.	Valid values are: • 0 = Default. Enterprise Replication is disabled. • 1 = Enterprise Replication is enabled. • 2 = Enterprise Replication is disabled and errors are suppressed. • 3 = Enterprise Replication is enabled and errors are suppressed.
defer	Run DDL statements on the local server but delay the propagation of the statements to other servers in the grid. Optionally enable the creation of a replicate and replicate set and starting replication.	Valid values are: • 4 = Defer the propagation of DDL statements. • 5 = Defer the propagation of DDL statements and enable Enterprise Replication. Use this value when you run DDL statements on existing replicated tables.
tag	A character string to identify grid operations.	Must be unique among grid sessions with deferred DDL statements that are outstanding.

Usage

Use the **ifx_grid_connect()** procedure start a grid connection. All DDL SQL statements and routines that you run in the grid connect are propagated to all the servers in the grid. Use the **ifx_grid_disconnect()** procedure to close the grid connect and disable grid propagation. If the databases on your replication servers have different schemas or data, a DDL statement that is run through a grid might have different results on each server. In a replication system, when you run a statement locally, the results are replicated to the other replication servers. When you run a statement through a grid, that statement is simultaneously run on each server.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr enable grid** command.

You must connect to a database before you run the **ifx_grid_connect()** procedure. If you are planning to create a database, you can connect to the **sysmaster** database.

If you enable Enterprise Replication, when you create a table through the grid, a replicate is created that contains the newly created table with all the servers in the grid as participants. The replicate belongs to a replicate set that has the same name as the grid. When you create a replicated table through the grid, the ERKEY shadow columns are added automatically.

If you run the **ifx_grid_connect()** procedure automatically as part of the **sysdbopen()** procedure, set the *ER_enable* argument to 2 or 3 to suppresses errors that might prevent the session from accessing the database.

You can defer the propagation of DDL statements to other servers in the grid by setting the *defer* argument. The DDL statements are queued for propagation but not sent to other grid servers until you run the **ifx_grid_release()** function.

You cannot perform the following actions in the context a grid connection:

- Propagate data manipulation language statements through a grid.
- Replicate a database object that exists on a server in the grid.
- Use the `@servername` syntax while connected to the grid.
- Drop a replicated column through a grid. To drop a replicated column, you must manually remaster the replicate and then drop the column.
- Renaming a replicated database. You must manually rename the database on each participant server.

Example 1: Create a table

In the following example, a grid connection is opened that enables the propagation of only DDL statements, a table is created on all servers in the grid, and then the grid connection is closed:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1');

CREATE TABLE special_offers(
    offer_description   varchar(255),
    offer_startdate     date,
    offer_enddate       date,
    offer_rules         lvarchar);

EXECUTE PROCEDURE ifx_grid_disconnect();
```

In this example, the data in the **special_offers** table is not replicated.

Example 2: Create a replicated table

In the following example, a grid connection is opened that enables the propagation of DDL statements and the replication of data, a table is created on all servers in the grid, and then the grid connection is closed:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE TABLE special_offers(
          offer_description     varchar(255),
    offer_startdate     date,
```

```
    offer_enddate       date,
    offer_rules         lvarchar)
    WITH CRCOLS;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

A replicate for the **special_offers** table is created with timestamp conflict resolution and replication of the data in the table is started.

Example 3: Alter a replicated table to add a column

The following example alters the **special_offers** table to add a column and remasters the replicate on all participants that are members of the grid:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

ALTER TABLE special_offers ADD (
          offer_exceptions      varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Example 4: Alter a replicated table to add a column that is not replicated

The following example alters the **special_offers** table to add a column whose data is not replicated:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 0);

ALTER TABLE special_offers ADD (
          local_promotions      varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

The column **local_promotions** is added to the **special_offers** table on all grid servers, but the data in the **local_promotions** column is not replicated.

Example 5: Defer propagation of a DDL statement

The following example defers propagation of the ALTER operation across **grid1**:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1','tag1',4);

ALTER TABLE special_offers ADD (
          local_restrictions     varchar(255));
```

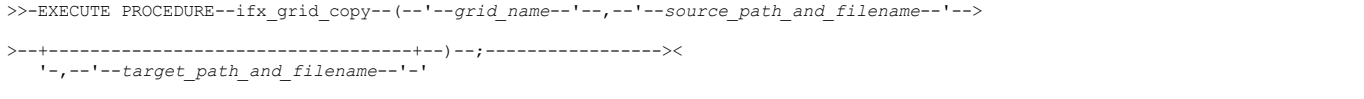
The column **local_restrictions** is added to the **special_offers** table on the local server only. The ALTER operation is queued for propagation to the other grid servers.

Related concepts:
Grid queries
Related tasks:
Propagating database object changes
Creating replicated tables through a grid
Adding an existing replicate to a grid replicate set by altering a table
Related reference:
ifx_grid_disconnect() procedure
cdr enable grid
ifx_grid_release() function
ifx_grid_remove() function
cdr delete grid

ifx_grid_copy() procedure

The **ifx_grid_copy()** procedure copies non-database, external files from a grid database server to other nodes in the same grid.

Syntax



Element	Purpose	Restriction
grid_name	The name of the database server's grid.	

Element	Purpose	Restriction
source_path_and_filename	The file path, relative to the value of the GRIDCOPY_DIR configuration parameter on the source database server, and name of the file you want to send to the other nodes of the grid.	The file must be located relative to the directory specified by the GRIDCOPY_DIR configuration parameter on the source server. By default, the GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR.
target_path_and_filename	The file path, relative to values of the GRIDCOPY_DIR configuration parameter on each other node of a grid, and the name each file will have on the other nodes of the grid.	If you do not specify *target_path_and_filename*, the file is copied to *source_path_and_filename* on each node, relative to the GRIDCOPY_DIR configuration parameter value on that node.

Usage

The **ifx_grid_copy()** procedure copies a file, along with the file's permissions, group, and owner values, from a directory on a grid server to specified destinations on each other node that is currently part of the grid. Group and owner values, rather than group ID and user ID values, are copied because group IDs and user IDs can have different values on different servers. If the file's group or owner values have not been defined on a node receiving a copied file, the copy fails on that node.

You can run this procedure only on an authorized database server and as an authorized user, as specified by the **cdr grid enable** command.

Only nodes that are members of a grid at the time the **ifx_grid_copy()** is run receive the copied file. Grid nodes added after **ifx_grid_copy()** procedures complete are not updated with files previously copied to other nodes.

Nonexistent directories that are specified by the *source_path_and_filename* or *target_path_and_filename* values of the **ifx_grid_copy()** command are created during the **ifx_grid_copy()** procedure.

Wildcard characters in file names are not supported.

Example 1: Copying a file to the same location on database servers of a grid

The GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR/tmp on all database servers of a grid named **grid1**. The following command copies the file script1.exe from $INFORMIXDIR/tmp/bin on the source database server to $INFORMIXDIR/tmp/bin on all other nodes of **grid1**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid1", "bin/script1.exe");
```

Example 2: Copying a file to the same relative locations on other database servers of a grid

The GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR/tmp on the source database server and $INFORMIXDIR/copies on each other node of a grid named **grid2**. The following command copies the file script2.exe from $INFORMIXDIR/tmp/bin on the source database server to $INFORMIXDIR/copies/bin on all other nodes of **grid2**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid2", "bin/script2.exe");
```

Example 3: Copying a file to different relative locations on the other database servers of a grid

The GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR/tmp on all database servers of a grid named **grid3**. The following command copies the file script3.exe from $INFORMIXDIR/tmp/bin on the source server to $INFORMIXDIR/tmp/copies on all other nodes of **grid3**.

```
EXECUTE PROCEDURE ifx_grid_copy ("grid3", "bin/script3.exe", "copies/script3.exe");
```

Example 4: Changing the name of file copied throughout a grid

The GRIDCOPY_DIR configuration parameter is set to $INFORMIXDIR/tmp on all database servers of a grid named **grid4**. The following command copies the file script4.exe in $INFORMIXDIR/tmp on the source server to $INFORMIXDIR/tmp on all other nodes of **grid4**. The copied file is named script4_copy.exe on the other nodes of **grid4**.

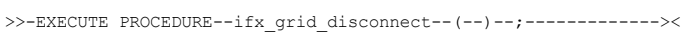
```
EXECUTE PROCEDURE ifx_grid_copy ("grid4", "script4.exe", "script4_copy.exe");
```

Related tasks:
Propagating external files through a grid
Related reference:
GRIDCOPY_DIR Configuration Parameter

ifx_grid_disconnect() procedure

The **ifx_grid_disconnect()** procedure closes a connection to the grid.

Syntax



Usage

Use the **ifx_grid_disconnect()** procedure to disable the propagation of DDL statements and commands to servers in the grid, which was enabled by the **ifx_grid_connect()** procedure. If you do not use the **ifx_grid_disconnect()** procedure, propagation through the grid is stopped when the database is closed or the connection is closed.

You must run this routine as an authorized user on an authorized grid server, as specified by the **cdr grid enable** command.

Example

The following example shows how to close a connection to the grid after opening a connection:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');
EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related tasks:
Propagating database object changes
Related reference:
ifx_grid_connect() procedure

ifx_grid_execute() procedure

The **ifx_grid_execute()** procedure propagates the execution of a routine or data manipulation language (DML) SQL statement to all servers in the grid.

Syntax

```
>>-EXECUTE PROCEDURE--ifx_grid_execute--(--'--grid_name--'------>

>--,--'--statement_text--'--+---------------+--)--;-------------><
                            '-,--'--tag--'-'
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
statement_text	The text format of the routine or SQL statement to be run.	Bound data items cannot be included in procedure text.
tag	A character string to identify grid operations.	

Usage

Use the **ifx_grid_execute()** procedure to run a routine or DML SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. The output of the routine, if any, is not returned to the client application. The results of routines or statements that are performed within the context of the **ifx_grid_execute()** procedure are not replicated. The **ifx_grid_execute()** procedure effectively runs routines and statements with a BEGIN WORK WITHOUT REPLICATION statement. Do not use the **ifx_grid_execute()** procedure to populate tables that are already involved in replication. Although you can use the **ifx_grid_execute()** procedure to run a DML statement, for example, to delete many rows from a table, in general use Enterprise Replication to replicate changes to replicated data. You can run DML statements on any type of table, including raw tables, virtual tables, and external tables.

You cannot run the **ifx_grid_execute()** procedure from within a transaction. When you run SQL administration API commands from the **ifx_grid_execute()** procedure, you must use double quotation marks around the SQL administration API function arguments and single quotation marks around the **ifx_grid_execute()** procedure arguments.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example, run from the **sysadmin** database, uses an SQL administration API command to create a dbspace on every server in the grid:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1',
                'admin("create dbspace", "dbspace3",
                "$INFORMIXDIR/WORK/dbspace3", "500 M")');
```

The following example drops the logical logs from the chunk number 3 from all the servers in the grid:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1', 'SELECT task("drop log", number) FROM
     sysmaster:syslogfil where chunk = 3;');
```

Related tasks:
Administering servers in the grid with the SQL administration API
Propagating updates to data
Related reference:
ifx_grid_procedure() procedure
ifx_grid_function() function

ifx_grid_function() function

The **ifx_grid_function()** function propagates the execution of a function to all servers in the grid.

Syntax

```
>>-EXECUTE FUNCTION--ifx_grid_function--(--'--grid_name--'------>

>--,--'--function_text--'--+--------------+--)--;-------------><
                           '-,--'--tag--'-'
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
function_text	The text format of the function to be run.	
tag	A character string to identify grid operations.	

Usage

Use the **ifx_grid_function()** function to run a function or SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. The output of the function is returned to the client application as an LVARCHAR data type with comma-delimited text. You can also view the output with the **cdr list grid** command with the **--verbose** option. You cannot run the **ifx_grid_function()** function from within a transaction.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example runs a function named **load_function()**:

```
EXECUTE FUNCTION ifx_grid_function('grid1', 'load_function(2000)');
```

Related tasks:
Administering servers in the grid with the SQL administration API
Related reference:
ifx_grid_execute() procedure

ifx_grid_procedure() procedure

The **ifx_grid_procedure()** procedure propagates the execution of a procedure to all servers in the grid.

Syntax

```
>>-EXECUTE PROCEDURE--ifx_grid_procedure--(--'--grid_name--'---->

>--,--'--procedure_text--'--+--------------+--)--;------------><
                           '-,--'--tag--'-'
```

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	
procedure_text	The text format of the procedure to be run.	Bound data items cannot be included.	
tag	A character string to identify grid operations.		

Usage

Use the **ifx_grid_procedure()** procedure to run a procedure or SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. You cannot run the **ifx_grid_procedure()** procedure from within a transaction.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example runs a procedure named **myloadprocedure()**:

```
EXECUTE PROCEDURE ifx_grid_procedure('grid1',
        'myloadprocedure(2000)', 'mytag');
```

Related reference:
ifx_grid_execute() procedure

ifx_grid_redo() procedure

The **ifx_grid_redo()** procedure reruns commands that were run through the grid and failed on one or more servers in the grid.

Syntax

```
>>-EXECUTE PROCEDURE--ifx_grid_redo--(--'grid_name'------------->

>--+-------------------------------------------------------------------------------+-->
   '-,'source_server'-+-------------------------------------------------------+-'
                      '-,'target_server'-+-------------------------------+-'
                                         '-,'tag'-+--------------------+-'
                                                  '-,'command_ID'-+----------+-'
                                                                  '-,'force'-'

>--)--;-------------------------------------------------------><
```

Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
command_ID	One or more ID numbers of the command to rerun on the grid.	Separate multiple ID numbers with a comma or specify a range with a hyphen (-). Can be NULL.
source_server	The replication server from which the routine was run.	Can be NULL.
tag	A character string identifying the grid operations to rerun.	Must be an existing tag. Can be NULL.
target_server	The replication server on which to rerun the routine.	Can be NULL.

Usage

Commands that you run through the grid might fail on one or more servers in the grid. Use the **ifx_grid_redo()** procedure to rerun commands that failed. For example, if you create a fragmented table through the grid and one of the grid servers does not have one of the dbspaces into which the table is fragmented, the command fails on that server. After you add the required dbspace to the server, run the **ifx_grid_redo()** procedure to create the fragmented table on that server.

You can specify from which source server the commands were run, the command ID, the target server on which the commands failed, or the identifying tag for commands that failed.

Use the force argument to rerun commands that succeeded.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example reruns failed commands on every server in the grid on which those commands failed:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1');
```

The following example reruns the command with the ID of 21 that originated server **cdr1** on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1', 'cdr1', 'cdr4', NULL, '21');
```

The following example reruns all commands that failed on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1', NULL, 'cdr4');
```

Related tasks:
Rerunning failed grid routines

ifx_grid_release() function

The **ifx_grid_release()** function propagates deferred DDL statements that were run on the local grid server, but deferred from running on the other grid servers.

Syntax

```
>>-EXECUTE FUNCTION--ifx_grid_release--(--'--grid_name--'--,--'--tag--'--)--;-><
```

Element	Purpose	Restrictions
grid_name	The name of the grid that a DDL statement is queued to propagates across	Must be the name of an existing grid.

Element	Purpose	Restrictions
tag	A character string to identify grid operations.	Must be the same value as the *tag* argument that was included in the **ifx_grid_connect()** procedure with the *defer* argument.

Usage

If you deferred the propagation of DDL statements by running the **ifx_grid_connect()** procedure with the *defer* argument, DDL statements are run on the local server but deferred from propagating across the grid. Run the **ifx_grid_release()** function to propagate the DDL statements to the other grid servers. You can run the **ifx_grid_release()** command at any time after the grid session in which the statements were deferred.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr enable grid** command.

Returns

The number of DDL statements that are released.

Example

The following example defers propagation of the ALTER operation across **grid1**:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1','tag1',4);

ALTER TABLE special_offers ADD (
          local_restrictions     varchar(255));
```

The following statement releases the queued ALTER operation:

```
EXECUTE FUNCTION ifx_grid_release('grid1','tag1');
```

The **local_restrictions** column is added to the **special_offers** table on the other grid servers.

Related concepts:
Grid queries
Related reference:
ifx_grid_connect() procedure
ifx_grid_remove() function
Related information:
SELECT_GRID session environment option
SELECT_GRID_ALL session environment option
GRID clause

ifx_grid_remove() function

The **ifx_grid_remove()** function removes any DDL statements that are deferred from propagation to grid servers.

Syntax

```
>>-EXECUTE FUNCTION--ifx_grid_remove--(--'--grid_name--'--,--'--tag--'--)--;-><
```

Element	Purpose	Restrictions
grid_name	The name of the grid that has a deferred DDL statement.	Must be the name of an existing grid.
tag	A character string to identify grid operations.	Must be the same value as the *tag* argument that was included in the **ifx_grid_connect()** procedure with the *defer* argument.

Usage

If you deferred the propagation of DDL statements by running the **ifx_grid_connect()** procedure with the *defer* argument, DDL statements are run on the local server but deferred from propagating across the grid. If you decide to not propagate the deferred DDL statements, run the **ifx_grid_remove()** function to remove the deferred DDL statement. You can run the **ifx_grid_remove()** command at any time after the grid session in which the statements were deferred. The **ifx_grid_remove()** command does not roll back the DDL statements on the local server.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr enable grid** command.

Returns

The number of DDL statements that are removed.

Example

The following example defers propagation of the ALTER operation across **grid1**:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1','tag1',4);

ALTER TABLE special_offers ADD (
          local_restrictions     varchar(255));
```

The following statement removes the queued ALTER operation:

```
EXECUTE FUNCTION ifx_grid_remove('grid1','tag1');
```

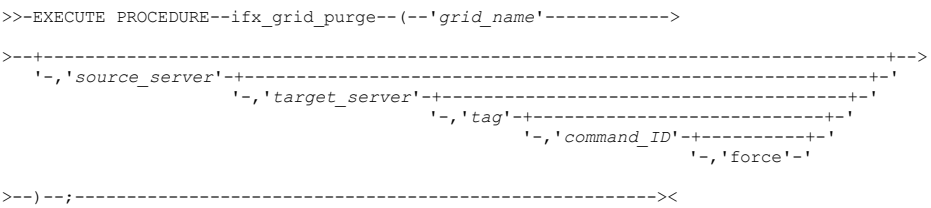
The **local_restrictions** column remains in the **special_offers** table on the local server but is not added to the special_offers tables on the other grid servers.

Related concepts:
Grid queries
Related reference:
ifx_grid_connect() procedure
ifx_grid_release() function
Related information:
SELECT_GRID session environment option
SELECT_GRID_ALL session environment option
GRID clause

ifx_grid_purge() procedure

The **ifx_grid_purge()** procedure deletes metadata about commands that have been run through the grid.

Syntax



Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
command_ID	One or more ID numbers of the command to purge.	Separate multiple ID numbers with a comma or specify a range with a hyphen (-). Can be NULL.
source_server	The replication server on which the routine originated.	Can be NULL.
tag	A character string identifying the grid operations to purge.	Must be an existing tag. Can be NULL.
target_server	The replication server on which the routine was run.	Can be NULL.

Usage

Use the **ifx_grid_purge()** procedure to delete the history of commands successfully run from the grid. Accumulated command history can significantly increase the size of the **syscdr** database.

Use the force argument to delete the history of all commands, including those that failed.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example deletes the history for commands that ran successfully:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1');
```

The following example deletes the history for commands, including those that failed:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', NULL, NULL, NULL, NULL, 'force');
```

The following example deletes the command history with the ID of 21 that originated server **cdr1** and ran on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', 'cdr1', 'cdr4', NULL, '21');
```

The following example deletes all commands that ran successfully on server **cdr4**:

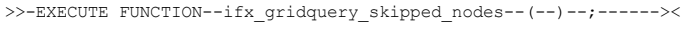
```
EXECUTE PROCEDURE ifx_grid_purge('grid1', NULL, 'cdr4');
```

Related concepts:
Grid maintenance

ifx_gridquery_skipped_nodes() function

The **ifx_gridquery_skipped_nodes()** function returns the name of a server that was unavailable during a grid or shard query.

Syntax



Usage

If you set the GRID_NODE_SKIP option of the SET ENVIRONMENT statement to `'on'`, any servers that are unavailable when a grid or shard query is run are skipped. Run the **ifx_gridquery_skipped_nodes()** function to return the name of a skipped server.

Use the **ifx_gridquery_skipped_nodes()** function with the **ifx_gridquery_skipped_node_count()** function. Run the **ifx_gridquery_skipped_node_count()** function to determine how many servers were skipped, and then run the **ifx_gridquery_skipped_nodes()** function the same number of times as the number of skipped servers.

Return value

An LVARCHAR string that supplies the name of a skipped server.

A return value of `0` indicates no skipped servers.

Example

The following statement returns the number of skipped nodes in the grid or shard query:

```
EXECUTE FUNCTION ifx_gridquery_skipped_node_count();
```

```
2
```

The following statements return the names of the two skipped nodes:

```
EXECUTE FUNCTION ifx_gridquery_skipped_nodes();
```

```
server1
```

```
EXECUTE FUNCTION ifx_gridquery_skipped_nodes();
```

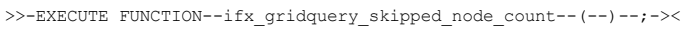
```
server2
```

Related concepts:
Examples of grid queries
Grid queries
Related reference:
ifx_gridquery_skipped_node_count() function
Related information:
GRID_NODE_SKIP session environment option
GRID clause

ifx_gridquery_skipped_node_count() function

The **ifx_gridquery_skipped_node_count()** function returns the number of servers that were unavailable during a grid or shard query.

Syntax



Usage

If you set the GRID_NODE_SKIP option of the SET ENVIRONMENT statement to `'on'`, any servers that are unavailable when a grid or shard query is run are skipped. Run the **ifx_gridquery_skipped_node_count()** function to return the number of a skipped servers.

Use the **ifx_gridquery_skipped_node_count()** function with the **ifx_gridquery_skipped_nodes()** function. Run the **ifx_gridquery_skipped_node_count()** function to determine how many servers were skipped, and then run the **ifx_gridquery_skipped_nodes()** function the same number of times as the number of skipped servers.

After a grid or shard query is run, the next SELECT statement that is run resets skipped node count, so **ifx_gridquery_skipped_node_count()** must be used in an EXECUTE FUNCTION statement.

Return value

An integer that indicates the number of skipped servers.

0 indicates that no grid or shard servers were skipped.

Example

The following statement returns the number of skipped nodes in the grid or shard query:

```
EXECUTE FUNCTION ifx_gridquery_skipped_node_count();
```

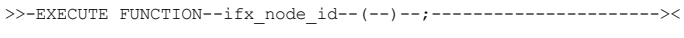
```
2
```

Related concepts:
Examples of grid queries
Grid queries
Related reference:
ifx_gridquery_skipped_nodes() function
Related information:
GRID_NODE_SKIP session environment option
GRID clause

ifx_node_id() function

The **ifx_node_id()** function returns the ID of the grid server on which the function is run.

Syntax



Usage

Use the **ifx_node_id()** function in the context of a grid query to return the ID of each server on which the grid query is run. Include the **ifx_node_id()** function in the SELECT statement of a grid query. The server ID is returned as a result column to identify the origin of the other results of the query.

If you run the **ifx_node_id()** function outside of the context of a grid query, the function returns the ID of the local server, unless you prefix the remote database and server name, for example: **db@serv4:ifx_node_id()**.

Example

The following grid query selects the server ID and the total sales from a grid named **SE_USA** and groups the results by the server ID:

```
SELECT ifx_node_id() AS ifx_node_id, sum(amt) AS total_sales
FROM sales GRID ALL 'SE_USA'
GROUP BY ifx_node_id;

ifx_node_id      total_sales

          1         $2100.00
          2         $2160.00
          3         $2000.00
          4         $2040.00
```

Related concepts:
Examples of grid queries
Grid queries
Related reference:
cdr define region
cdr delete region
ifx_node_name() function
Related information:
GRID clause

ifx_node_name() function

The **ifx_node_name()** function returns the name of the grid server on which the function is run.

Syntax

```
>>-EXECUTE FUNCTION--ifx_node_name--(--)--;--------------------><
```

Usage

Use the **ifx_node_name()** function in the context of a grid query to return the name of each server on which the grid query is run. Include the **ifx_node_name()** function in the SELECT statement of a grid query. The server name is returned as a result column to identify the origin of the other results of the query.

If you run the **ifx_node_name()** function outside of the context of a grid query, the function returns the name of the local server, unless you prefix the remote database and server name, for example: **db@serv4:ifx_node_name()**.

Example

The following grid query selects the server name and the total sales from a grid named **SE_USA** and groups the results by the server name:

```
SELECT ifx_node_name() AS node, sum(amt) AS total_sales
FROM sales GRID ALL 'SE_USA'
GROUP BY node;

node         Atlanta
total_sales  $2100.00

node         Birmingham
total_sales  $2160.00

node         Nashville
total_sales  $2000.00

node         Jacksonville
total_sales  $2040.00
```

Related concepts:
Examples of grid queries
Grid queries
Related reference:
cdr define region
cdr delete region
ifx_node_id() function
Related information:
GRID clause

Enterprise Replication routines

Enterprise Replication routines used to control if a replicated transaction is recaptured.

- ifx_get_erstate() function
  The **ifx_get_erstate()** function indicates whether replication is enabled for the transaction in which it is run.
- ifx_set_erstate() procedure
  The **ifx_set_erstate()** procedure controls whether database operations are replicated.

ifx_get_erstate() function

The **ifx_get_erstate()** function indicates whether replication is enabled for the transaction in which it is run.

Syntax

```
>>-EXECUTE FUNCTION--ifx_get_erstate--(--)--INTO--data_var--;-->
```

Element	Purpose	Restriction
data_var	Variable to receive the value that the function returns	

Usage

Use the **ifx_get_erstate()** function to obtain the state of replication within a transaction. You can use the state information saved in the variable as input to the **ifx_set_erstate()** procedure.

Return value

A return value of 1 indicates that the current transaction is replicating data.

A return value of 0 indicates that the current transaction is not replicating data.

Example

The following example obtains the replication state and stores it in the curstate variable:

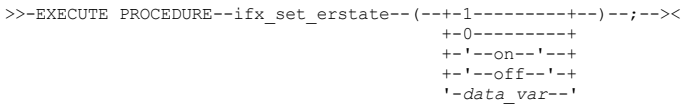
```
EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
```

Related concepts:
Recapture replicated transactions
Related tasks:
Enabling replication within a grid transaction
Related reference:
ifx_set_erstate() procedure

ifx_set_erstate() procedure

The **ifx_set_erstate()** procedure controls whether database operations are replicated.

Syntax



Element	Purpose	Restriction
data_var	Variable holding the value that a function returned	

Usage

Use the **ifx_set_erstate()** procedure to enable or disable replication during a transaction. During normally replicated transactions, use the **ifx_set_erstate()** procedure to enable the recapture of a transaction after it has been replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Replication can only be enabled on tables that are participants in an existing replicate. To enable replication, set the **ifx_set_erstate()** procedure to `1` or `'on'`. To disable replication, set the **ifx_set_erstate()** procedure to `0` or `'off'`. To set replication to a previous state that was saved by the **ifx_get_erstate()** function, set the **ifx_set_erstate()** procedure to the name of the variable returned by the **ifx_get_erstate()** function.

Example

The following example enables replication in the transaction:

```
EXECUTE PROCEDURE ifx_set_erstate(1);
```

The following example resets the replication state to a previous state that was saved by the **ifx_get_erstate()** function in the curstate variable:

```
EXECUTE PROCEDURE ifx_set_erstate(curstate);
```

Related concepts:
Recapture replicated transactions
Related tasks:
Enabling replication within a grid transaction
Related reference:
ifx_get_erstate() function

onstat -g commands for Enterprise Replication

You can monitor and debug Enterprise Replication activity using **onstat -g** commands.

The **onstat** utility reads shared-memory structures and provides statistics about the database server that are accurate at the instant that the command executes. The system-monitoring interface (SMI) also provides information about the database server. For general information about **onstat** and SMI, refer to the *IBM® Informix® Administrator's Reference*. For information on SMI tables specific to Enterprise Replication, see SMI Tables for Enterprise Replication Reference.

- Threads shown by the onstat -g ath command
  The threads that Enterprise Replication uses are shown by the **onstat -g ath** command.
- onstat -g cat: Print ER global catalog information
  Prints information from the Enterprise Replication global catalog.
- onstat -g cdr: Print ER statistics
  Prints the output for all of the Enterprise Replication statistics commands.
- onstat -g cdr config: Print ER settings
  Prints the settings of Enterprise Replication configuration parameters and environment variables that can be set with the CDR_ENV configuration parameter.
- onstat -g ddr: Print status of ER log reader
  Prints the status of the Enterprise Replication database log reader.
- onstat -g dss: Print statistics for data sync threads
  Prints detailed statistical information about the activity of individual data sync threads.
- onstat -g dtc: Print statistics about delete table cleaner
  Prints statistics about the delete table cleaner.
- onstat -g grp: Print grouper statistics
  Prints statistics about the grouper.
- onstat -g nif: Print statistics about the network interface
  Prints statistics about the network interface.
- onstat -g que: Print statistics for all ER queues
  Prints statistics that are common to all queues.
- onstat -g rcv: Print statistics about the receive manager
  Prints statistics about the Enterprise Replication receive manager. The receive manager is a set of service routines between the receive queues and data sync. The **onstat -g rcv** command is used primarily as a debugging tool and by Software Support. If you suspect that acknowledgment messages are not being applied, you can run this command.
- onstat -g rep: Prints the schedule manager queue
  Prints events that are in the queue for the schedule manager.
- onstat -g rqm: Prints statistics for RQM queues
  Prints statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM).
- onstat -g sync: Print statistics about synchronization
  Prints statistics about the active synchronization process.

Related concepts:
Monitor and troubleshooting Enterprise Replication
Related information:
onstat -k command: Print active lock information
onstat -g ath command: Print information about all threads

Threads shown by the onstat -g ath command

The threads that Enterprise Replication uses are shown by the **onstat -g ath** command.

The following table summarizes the threads that Enterprise Replication uses. You can use this information about threads when you evaluate memory use.

Table 1. Enterprise Replication threads

Number of Threads	Thread Name	Thread Description
1	ddr_snoopy	Performs physical I/O from logical log, verifies potential replication, and sends applicable log-record entries to Enterprise Replication.
1	preDDR	Runs during queue recovery to monitor the log and blocks user transactions if the log position advances too far before replication resumes.
1	CDRGfan	Receives log entries and passes entries to evaluator thread
n	CDRGeval*n*	Evaluates log entry to determine whether to replicated it. The value of *n* is the number of evaluator threads specified by the CDR_EVALTHREADS configuration parameter. This thread also compresses committed transactions and queues completed replication messages.
1 per large transaction	CDRPager	Performs the physical I/O for the temporary smart large object that holds paged transaction records. Grouper paging is activated for a transaction when its size is 10 percent of the value of the SHMVIRTSIZE or CDR_QUEUEMEM configuration parameters or when it includes more than 100,000 records.
1	CDRCparse	Parses all SQL statements for replicate definitions.
1 per connection	CDRNsT*n*CDRNsA*n*	Sending thread for site.
1 per connection	CDRNr*n*	Receiving thread for site.
2...*n*	CDRACK_*n*	Accepts acknowledgments from site. At least 2, up to a maximum of the number of active connections.
# CPUs...	CDRD_*n*	Replays transaction on the target system (data sync thread). At least one thread is created for each CPU virtual processor (VP). The maximum number of threads is 4*(number of CPU VPs).
1	CDRSchedMgr	Schedules internal Enterprise Replication events.

Number of Threads	Thread Name	Thread Description
0 or 1	CDRM_Monitor	Monitors and adjusts data sync performance for optimum performance (on the target).
0 or 1	CDRDTCleaner	Deletes rows from the deleted rows shadow table when they are no longer needed.

Related information:
onstat -g ath command: Print information about all threads

onstat -g cat: Print ER global catalog information

Prints information from the Enterprise Replication global catalog.

```
                  .-full-----.
>>-onstat -g cat--+----------+--------------------------------><
                  +-replname-+
                  +-repls----+
                  '-servers--'
```

Modifier	Description
replname	The name of a replicate

Usage

The global catalog contains a summary of information about the defined servers, replicates, and replicate sets on each of the servers within the domain. If a replicated table is undergoing an alter operation, the **onstat -g cat** command shows that it is in alter mode. For example, use this command to determine:

- How many servers and how many replicates are configured
- Which table matches a given replicate
- Whether a server is a root or leaf server
- The current bitmap mask for the specified server. You can use the bitmap mask with the output from the **onstat -g rqm** command to determine which server Enterprise Replication is waiting on for an acknowledgment.

You can set the scope of the output by specifying one of the following options to **onstat -g cat**:

- **full**: (Default) Prints expanded information for both replicate servers and replicates.
- *replname*: Prints information about the specified replicate only.
- **repls**: Prints information about replicates only.
- **servers**: Prints information about servers only.

This sample output from the **onstat -g cat repls** command shows that the table **tab** is in alter mode. The replicate **rep1** is defined on this table, its replicate ID is 6553601.

```
GLOBAL-CATALOG CACHE STATISTICS
REPLICATES
-------------------
Parsed statements:
    Id 6553601 table tab
    Id 6553602 table tab12
Inuse databases: test(2)
  Name: rep1, Id: 6553601 State: ACTIVE Flags: 0x800000 ALTERMODE
         use 0 lastexec Wed Dec 31 18:00:00 1969
    Local Participant: test:nagaraju.tab
    Attributes: TXN scope, Enable ATS, Enable RIS, all columns
         sent in updates
    Conflict resolution: [TIMESTAMP]
    Column Mapping: ON, columns INORDER, offset 8, uncomp_len 12
    Column Name Verifcation: ON
    No Replicated UDT Columns
  Name: rep12, Id: 6553602 State: ACTIVE Flags: 0x800000 use 0
         lastexec Wed Dec 31 18:00:00 1969
    Local Participant: test:nagaraju.tab12
    Attributes: TXN scope, Enable ATS, Enable RIS, all columns
         sent in updates
    Conflict resolution: [TIMESTAMP]
    Column Mapping: ON, columns INORDER, offset 8, uncomp_len 2064
    Column Name Verifcation: ON
    No Replicated UDT Columns
```

The following replicate information shows that the replicate belongs to a grid replicate set. `UTF8` indicates that code set conversion between replicates is enabled.

```
Name: grid_6553604_100_3, Id: 6553605 State: ACTIVE Flags: 0x900000 UTF8 GRID
        use 0 lastexec Wed Dec 31 18:00:00 1969

        Local Participant: tdb:nagaraju.t1
        Attributes: ROW scope, Enable RIS, all columns sent in updates
        Conflict resolution[Prim::Sec]: [ALWAYSAPPLY]
        Column Mapping: OFF
        Column Name Verifcation: ON
        No Replicated UDT Columns
```

This sample output from the **onstat -g cat servers** command shows that the server **g_bombay** and **g_delhi** are active; neither one is a hub or a leaf server, and both have ATS and RIS files that are generated in XML format.

```
GLOBAL-CATALOG CACHE STATISTICS

SERVERS
-------------------
  Current server  : Id 200, Nm g_bombay
  Last server slot: (0, 2)
  # free slots    : 0
  Broadcast map   : <[0005]>
  Leaf server map : <[0000]>
  Root server map : <[0006]>
  Adjacent server map: <[0004]>
    Id: 200, Nm: g_bombay, Or: 0x0002, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 00, ishub: FALSE, isleaf: FALSE
      subtree map: <empty>
      atsrisformat=xml

    Id: 100, Nm: g_delhi, Or: 0x0004, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 100, ishub: FALSE, isleaf: FALSE
      subtree map: <empty>
      atsrisformat=xml
```

Related tasks:
Viewing grid information
Related reference:
onstat -g cdr: Print ER statistics

onstat -g cdr: Print ER statistics

Prints the output for all of the Enterprise Replication statistics commands.

```
>>-onstat-- -g--cdr-----------------------------------------><
```

Usage

The output of the **onstat -g cdr** command is a combination of the following Enterprise Replication **onstat** command outputs:

- **onstat -g cat**
- **onstat -g grp**
- **onstat -g que**
- **onstat -g rqm**
- **onstat -g nif all**
- **onstat -g rcv**
- **onstat -g dss**
- **onstat -g dtc**
- **onstat -g rep**

Related reference:
onstat -g cat: Print ER global catalog information
onstat -g grp: Print grouper statistics
onstat -g que: Print statistics for all ER queues
onstat -g rqm: Prints statistics for RQM queues
onstat -g nif: Print statistics about the network interface
onstat -g rcv: Print statistics about the receive manager
onstat -g dss: Print statistics for data sync threads
onstat -g dtc: Print statistics about delete table cleaner
onstat -g rep: Prints the schedule manager queue

onstat -g cdr config: Print ER settings

Prints the settings of Enterprise Replication configuration parameters and environment variables that can be set with the CDR_ENV configuration parameter.

This command has the following formats:

```
onstat -g cdr config
onstat -g cdr config long
onstat -g cdr config parameter_name
onstat -g cdr config parameter_name long
onstat -g cdr config CDR_ENV
onstat -g cdr config CDR_ENV long
onstat -g cdr config CDR_ENV variable_name
onstat -g cdr config CDR_ENV variable_name long
```

The **long** option prints additional information about settings that can be useful for Support.

The following table describes *parameter_name* and *variable_name*.

Modifier	Description
parameter_name	The name of an Enterprise Replication configuration parameter
variable_name	The name of an Enterprise Replication environment variable

If you use **onstat -g cdr config** without any options, the settings of all Enterprise Replication configuration parameters and environment variables are included in the output. If you specify the CDR_ENV configuration parameter without an environment variable name, all Enterprise Replication environment variables are included in the output.

The following sample output of the **onstat -g cdr config ENCRYPT_CDR** command shows the setting of the ENCRYPT_CDR configuration parameter:

```
onstat -g cdr config ENCRYPT_CDR


ENCRYPT_CDR configuration setting:          0
```

The following sample output of the **onstat -g cdr config CDR_ENV** command shows the settings of all Enterprise Replication environment variables:

```
onstat -g cdr config CDR_ENV

CDR_ENV environment variable settings:
    CDR_LOGDELTA:
        CDR_LOGDELTA configuration setting:          0
    CDR_PERFLOG:
        CDR_PERFLOG configuration setting:           0
    CDR_ROUTER:
        CDR_ROUTER configuration setting:            0
    CDR_RMSCALEFACT:
        CDR_RMSCALEFACT configuration setting:       0
    CDRSITES_731:
        CDRSITES_731 configuration setting:            [None configured]
    CDRSITES_92X:
        CDRSITES_92X configuration setting:            [None configured]
    CDRSITES_10X:
        CDRSITES_10X configuration setting:            [None configured]
```

The following sample output of the **onstat -g cdr config** command shows the settings of all Enterprise Replication configuration parameters and CDR_ENV environment variables:

```
onstat -g cdr config

CDR_DBSPACE:
    CDR_DBSPACE configuration setting:                rootdbs
CDR_DSLOCKWAIT:
    CDR_DSLOCKWAIT configuration setting:             5
CDR_EVALTHREADS:
    CDR_EVALTHREADS configuration setting:            1, 2
CDR_MAX_DYNAMIC_LOGS:
    CDR_MAX_DYNAMIC_LOGS configuration setting:       0
CDR_NIFCOMPRESS:
    CDR_NIFCOMPRESS configuration setting:            0
CDR_QDATA_SBSPACE:
    CDR_QDATA_SBSPACE configuration setting:          cdrsbsp
CDR_QHDR_DBSPACE:
    CDR_QHDR_DBSPACE configuration setting:           rootdbs
CDR_QUEUEMEM:
    CDR_QUEUEMEM configuration setting:               4096
CDR_SERIAL:
    CDR_SERIAL configuration setting:                 0, 0
CDR_SUPPRESS_ATSRISWARN:
    CDR_SUPPRESS_ATSRISWARN configuration setting:    [None suppressed]
ENCRYPT_CDR:
    ENCRYPT_CDR configuration setting:                0
ENCRYPT_CIPHERS:
    ENCRYPT_CIPHERS configuration setting:            [None configured]
ENCRYPT_MAC:
    ENCRYPT_MAC configuration setting:                [None configured]
ENCRYPT_MACFILE:
    ENCRYPT_MACFILE configuration setting:            [None configured]
ENCRYPT_SWITCH:
    ENCRYPT_SWITCH configuration setting:             0,0
CDR_ENV environment variable settings:
    CDR_LOGDELTA:
        CDR_LOGDELTA configuration setting:           0
    CDR_PERFLOG:
        CDR_PERFLOG configuration setting:            0
    CDR_ROUTER:
        CDR_ROUTER configuration setting:             0
    CDR_RMSCALEFACT:
        CDR_RMSCALEFACT configuration setting:        0
    CDRSITES_731:
        CDRSITES_731 configuration setting:             [None configured]
    CDRSITES_92X:
        CDRSITES_92X configuration setting:             [None configured]
    CDRSITES_10X:
        CDRSITES_10X configuration setting:             [None configured]
```

Related tasks:
Dynamically Modifying Configuration Parameters for a Replication Server

onstat -g ddr: Print status of ER log reader

Prints the status of the Enterprise Replication database log reader.

The **ddr**, or **ddr_snoopy**, is an internal component of Enterprise Replication that reads the log buffers and passes information to the grouper.

You can use the information from the **onstat -g ddr** command to monitor *replay position* in the log file and ensure replay position is never overwritten (which can cause loss of data). The replay position is the point from where, if a system failure occurs, Enterprise Replication starts re-reading the log information into the log update buffers. All the transactions generated before this position at all the target servers have been applied by Enterprise Replication or safely stored in stable queue space. As messages are acknowledged or stored in the stable queue, the replay position should advance. If you notice that replay position is not advancing, this can mean that the stable queue is full or a remote server is down.

The **onstat -g ddr** output shows you a snapshot of the replay position, the *snoopy position*, and the *current position*. The snoopy position identifies the position of the **ddr_snoopy** thread in the logical logs. The **ddr_snoopy** has read the log records up until this point. The current position is the position where the server has written its last logical log record.

If log reading is blocked, data might not be replicated until the problem is resolved. If the block is not resolved, the database server might overwrite the read (**ddr_snoopy**) position, which means that data will not be replicated. If this occurs, you must manually resynchronize the source and target databases.

To avoid these problems, follow these guidelines:

- Have 24 hours of online log space available.
- Keep the log file size consistent. Instead of having a single large log file, implement several smaller ones.
- Avoid switching logical logs more than once per hour.
- Keep some distance between LTXHWM (long-transaction high-watermark) and LTXEHWM (long-transaction, exclusive-access, high-watermark).

You can configure one or more actions to occur if the current position reaches the log needs position by setting the CDR_LOG_LAG_ACTION configuration parameter.

The following sample output from the **onstat ddr** command shows the replay position, snoopy position, and current position highlighted.

```
DDR -- Running --

# Event  Snoopy   Snoopy   Replay   Replay   Current  Current
Buffers   ID      Position  ID      Position   ID     Position
2064     35       2ae050   34       121018   55       290000

Log Pages Snooped:
      From      From      From Staging     Tossed
     Cache      Disk      File             (LBC full)
         0         0       19704           0

CDR log records ignored   : 0
DDR log lag state   : On
Current DDR log lag action   : logstage
DDR log staging disk space usage  :0.26%
Maximum disk space allowed for log staging :1048576 KB
Maximum disk space ever used for log staging :2746.98 KB
Current staged log file count :21
Total dynamic log requests: 0

DDR events queue

Type   TX id    Partnum  Row id
```

Related reference:
cdr view

onstat -g dss: Print statistics for data sync threads

Prints detailed statistical information about the activity of individual data sync threads.

The data sync thread applies the transaction on the target server. Statistics include the number of applied transactions and failures and when the last transaction from a source was applied.

The **onstat -g dss** command has the following formats:

```
onstat -g dss
onstat -g dss modifier
```

The following table describes the values for *modifier*.

Modifier	Action
UDR	Prints summary information about any UDR invocations by the data sync threads.
UDRx	Prints expanded information (including a summary of error information) about any UDR invocations by the data sync threads. The `Procid` column lists the UDR procedure ID.

In the following example, only one data sync thread is currently processing the replicated data. It has applied a total of one replicated transaction and the transaction was applied at 2004/09/13 18:13:10. The Processed Time field shows the time when the last transaction was processed by this data sync thread.

```
-- Up 00:00:28 -- 28672 Kbytes
DS thread statistic
cmtTime    Tx       Tx        Tx       Last Tx
```

```
Name       < local  Committed Aborted  Processed  Processed Time
---------- -------  --------- -------  ---------  -----------------
CDRD_1     0        1         0        1          (1095117190) 2004/09/13
18:13:10
           Tables (0.0%):
           Databases: test
CDR_DSLOCKWAIT = 1
CDR_DSCLOSEINTERVAL = 60
```

Related reference:
onstat -g cdr: Print ER statistics

onstat -g dtc: Print statistics about delete table cleaner

Prints statistics about the delete table cleaner.

The delete table cleaner removes rows from the delete table when they are no longer needed.

The **onstat -g dtc** command is used primarily as a debugging tool and by Software Support.

In the following example, the thread name of the delete table cleaner is **CDRDTCleaner**. The total number of rows deleted is **1**. The last activity on this thread occurred at 2010/08/13 18:47:19. The delete table for replicate **rep1** was last cleaned at 2010/08/13 18:28:25.

```
-- Up 00:59:15 -- 28672 Kbytes
-- Delete Table Cleanup Status as of (1095119368) 2010/08/13 18:49:28
thread           = 49 <CDRDTCleaner>
      rows deleted     = 1
      lock timeouts    = 0
      cleanup interval = 300
      list size        = 3
      last activity    = (1095119239) 2010/08/13 18:47:19
```

```
Id      Database                        Last Cleanup Time
        Replicate            Server              Last Log Change
========================================================
000001  test                            (1095118105) 2010/09/13
  18:28:25
        rep1                 g_bombay           (1095118105) 2010
        /08/13 18:28:25
        rep1                 g_delhi            (1095118105) 2010
        /08/13 18:28:25
000002  test                            <never cleaned>
```

Related reference:
onstat -g cdr: Print ER statistics

onstat -g grp: Print grouper statistics

Prints statistics about the grouper.

The grouper evaluates the log records, rebuilds the individual log records into the original transaction, packages the transaction, and queues the transaction for transmission.

The**onstat -g grp** command is used primarily as a debugging tool and by Software Support.

The **onstat -g grp** command can take an optional modifier. The following table describes the values for the modifier.

Table 1. Modifiers for the onstat -g grp command

Modifier	Action
A	Prints all the information printed by the G, T, P, E, R, and S modifiers
E	Prints grouper evaluator statistics
Ex	Prints grouper evaluator statistics, expands user-defined routine (UDR) environments
G	Prints grouper general statistics
L	Prints grouper global list
Lx	Prints grouper global list, expands open transactions
M	Prints grouper compression statistics
Mz	Clears grouper compression statistics
P	Prints grouper table partition statistics
pager	Prints grouper paging statistics
R	Prints grouper replicate statistics

Modifier	Action
S	Prints grouper serial list head (The serial list head is the first transaction in the list, that is, the next transaction that will be placed in the send queue.)
Sl	Prints grouper serial list (The serial list is the list of transactions, in chronological order.)
Sx	Prints grouper serial list, expands open transactions
T	Prints grouper transaction statistics
UDR	Prints summary information about any UDR invocations by the grouper threads
UDRx	Prints expanded information (including a summary of error information) about any UDR invocations by the grouper threads The `Procid` column lists the UDR procedure ID.

The following sample shows output for the **onstat -g grp** command:

```
Grouper at 0xb014018:
Last Idle Time: (1095122236) 2010/09/13 19:37:16
RSAM interface ring buffer size: 528
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 48
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 5
Log update buffer memory in use: 0
Max log update buffer memory used at once: 320
Updates from Log: 16
Log update links allocated: 512
Blob links allocated: 0
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
Last Tx to Queuer began : (1095118105) 2010/09/13 18:28:25
Last Tx to Queuer ended : (1095118105) 2010/09/13 18:28:25
Last Tx to Queuer log ID, position: 12,23
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 2
Tx returned from Queuer: 2
Events sent to Queuer: 7
Events returned from Queuer: 7
Total rows sent to Queuer: 2
Open Tx array size: 1024
Table 'tab' at 0xae8ebb0 [ CDRShadow ]
Table 'tab12' at 0xae445e0 [ CDRShadow ]

Grouper Table Partitions:
  Slot  312...
    'tab' 1048888
  Slot  770...
    'tab12' 3145730
  Slot 1026...
    'tab12' 4194306
Repl links on global free list: 2
Evaluators: 3
  Evaluator at 0xb03d030 ID 0 [Idle:Idle] Protection:unused
    Eval iteration: 1264
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xb03d080
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xb03d0d8 ID 1 [Idle:Idle] Protection:unused
    Eval iteration: 1265
    Updates evaluated: 2
    Repl links on local free list: 254
    UDR environment table at 0xb03d128
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xb03d180 ID 2 [Idle:Idle] Protection:unused
    Eval iteration: 1266
    Updates evaluated: 4
    Repl links on local free list: 256
    UDR environment table at 0xb03d1d0
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
        Total Free Repl links 768

Replication Group 6553601 at 0xb0a8360
  Replication at 0xb0a82b0 6553601:6553601 (tab) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
```

```
      In Order: offset 8, size 10
Replication Group 6553602 at 0xb0a8480
  Replication at 0xb0a83d0 6553602:6553602 (tab12)[Ignore Stopped NotifyDS FullRowOn]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 16
```

The following example shows output for the **onstat -g grp E** command. The field **Evaluators: 4** indicates that there are four evaluation threads configured for the system.

```
Repl links on global free list: 0 Evaluators: 4
  Evaluator at 0xba71840 ID 0 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba71890
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0
  Evaluator at 0xba718f0 ID 1 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba71940
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0

  Evaluator at 0xba8c260 ID 2 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba8c2b0
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0
  Evaluator at 0xbaac2a0 ID 3 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xbaac2f0
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0
Total Free Repl links 1024
```

The following example shows output for the **onstat -g grp G** command.

```
Grouper at 0xb8ab020:
Last Idle Time: (1095115397) 2010/09/13 17:43:17
RSAM interface ring buffer size: 1040
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 64
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 1
Log update buffer memory in use: 0
Max log update buffer memory used at once: 64
Updates from Log: 1
Log update links allocated: 512
Blob links allocated: 0
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
```

The following example shows output for the **onstat -g grp P** command. In the following example, the grouper is evaluating rows for the **account**, **teller**, and **customer** tables.

```
Table 'teller' at 0xb851480 [ CDRShadow VarChars ]
Table 'account' at 0xb7faad8 [CDRShadow VarChars VarUDTs Floats
      Blobs]
Table 'customer' at 0xbbe67a8 [CDRShadow VarChars VarUDTs]
Grouper Table Partitions:
  Slot  387...
    'account' 1048707
  Slot  389...
    'teller' 1048709
  Slot  394...
    'customer' 1048714
```

The following example shows output for the **onstat -g grp pager** command. The sample output shows the grouper large transaction evaluation statistics.

```
Grouper Pager statistics:
Number of active big transactions: 0
```

```
Total number of big transactions processed: 0
Spool size of the biggest transaction processed: 0 Bytes
```

The following example shows output for the **onstat -g grp R** command. In this example, the grouper is configured to evaluate rows for replicates with IDs **6553601** and **6553602** (you can use the **onstat -g cat repls** command to obtain the replicate names). The **Ignore** attribute of replicate ID **6553602** shows that the grouper is currently not evaluating rows for this replicate. This can happen if the replicate state is not ACTIVE. You can obtain the replicate state using the **onstat -g cat repls** command.

```
Replication Group 6553601 at 0xb0a8360
  Replication at 0xb0a82b0 6553601:6553601 (tab) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 10
Replication Group 6553602 at 0xb0a8480
  Replication at 0xb0a83d0 6553602:6553602 (tab12)[Ignore Stopped NotifyDS FullRowOn]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 16
```

The following example shows output for the **onstat -g grp T** command. In this example, the grouper evaluated and queued 1 transaction to the send queue. The **Tx sent to Queuer** field shows the total number of transactions evaluated and queued to the send queue for propagating to all the replicate participants. The **Total rows sent to Queuer** field shows the total number of rows queued to the send queue for propagating to all the replicate participants.

```
Last Tx to Queuer began : (1095116676) 2010/09/13 18:04:36
Last Tx to Queuer ended : (1095116676) 2010/09/13 18:04:36
Last Tx to Queuer log ID, position: 5,3236032
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 1
Tx returned from Queuer: 0
Events sent to Queuer: 0
Events returned from Queuer: 0
Total rows sent to Queuer: 1
Open Tx array size: 1024
```

Related reference:
onstat -g cdr: Print ER statistics

onstat -g nif: Print statistics about the network interface

Prints statistics about the network interface.

```
>>-onstat -g nif--+-----------+--------------------------------><
                  +-all-------+
                  +-sites-----+
                  +-server_ID-+
                  '-sum-------'
```

The output shows which sites are connected and provides a summary of the number of bytes sent and received by each site. This can help you determine if a site is not sending or receiving bytes.

The **onstat -g nif** option is used primarily as a debugging tool and by Software Support.

The following table describes the options for **onstat -g nif** command:

Table 1. Options for the onstat -g nif command

Option	Action
all	Prints the sum and the sites.
sites	Prints the NIF site context blocks.
server_ID	Prints information about the replication server with that server ID.
sum	Prints the sum of the number of buffers sent and received for each site.

Example Output

The following example shows output for the **onstat -g nif** command. In this example, the local server is connected to the server group **g_bombay** and its CDR ID is **200**. The connection status is running. The connection between the two servers is running, but the replication state on the **g_bombay** server is suspended. The server group **g_bombay** internal NIF version is **9**. The local server has sent three messages to the server **g_bombay** and it has received two messages from **g_bombay**.

```
$ onstat -g nif

NIF anchor Block: af01610
       nifGState          RUN
          RetryTimeout    300

CDR connections:
 Id    Name        State     Version    Sent    Received
 ------------------------------------------------------
 200 g_bombay      RUN,SUSPEND    9        3          2
```

Output Description

NIF anchor Block
: The address of the network storage block.

nifGState
: The connection state.

RetryTimeout
: The number of seconds before Enterprise Replication attempts to retry a dropped connection.

Id
: The Enterprise Replication ID number for the server.

Name
: The name of the server group.

State
: The connection state between the local server and the listed server. If multiple states are shown the second state designates the replication state.

Version
: The internal version number of the NIF component on the listed server.

Sent
: The number of messages the local server has sent to the listed server.

Received
: The number of messages received by the local server from the listed server.

Related reference:
onstat -g cdr: Print ER statistics

onstat -g que: Print statistics for all ER queues

Prints statistics that are common to all queues.

The queuer manages the logical aspects of the queue. The RQM (reliable queue manager) manages the physical queue.

The **onstat -g que** command is used primarily as a debugging tool and by Software Support.

In the following example, **Element high water mark** shows the maximum size of the transaction buffer header data (metadata) allowed in memory, shown in kilobytes. **Data high water mark** shows the maximum size of transactions for user data allowed in memory, shown in kilobytes.

```
CDR Queuer Statistics:
  Queuer state             : 2
  Local server             : 100
  Element high water mark  : 131072
  Data high water mark     : 131072
  # of times txns split    : 0
  Total # of split txns    : 0
  allowed log delta        : 30
  maximum delta detected   : 4
  Control Key              : 0/00000007
  Synchronization Key      : 0/00000003
Replay Table:
  Replay Posn (Disk value): 12/00000018 (12/00000018)
  Replay save interval     : 10
  Replay updates           : 10
  Replay # saves           : 17
  Replay last save time    : (1095118157) 2010/09/13 18:29:17
Send Handles
  Server ID                : 200
  Send state,count         : 0,0
  RQM hdl for trg_send: Traverse handle (0xaf8e018) for thread CDRACK_0 at Head_of_Q,
      Flags: None
  RQM hdl for control_send:  Traverse handle (0xaf74018)
    for thread CDRACK_0 at Head_of_Q,  Flags: None
  RQM hdl for sync_send: Traverse handle (0xadc6018) for thread CDRACK_0 at Head_of_Q,
      Flags: None
  Server ID                : 200
  Send state,count         : 0,0
  RQM hdl for trg_send: Traverse handle (0xac8b018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  RQM hdl for control_send: Traverse handle (0xb1ce018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  RQM hdl for sync_send: Traverse handle (0xadc5018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  Server ID                : 200
  Send state,count         : 0,0
  RQM hdl for trg_send: Traverse handle (0xaea71d8) for thread CDRNsA200 at Head_of_Q,
      Flags: None
  RQM hdl for ack_send: Traverse handle (0xae8c1d8) for thread CDRNsA200 at Head_of_Q,
      Flags: None
  RQM hdl for control_send: Traverse handle (0xae9e1d8) for thread CDRNsA200 at Head_of_Q,
      Flags: None
```

Related reference:
onstat -g cdr: Print ER statistics

onstat -g rcv: Print statistics about the receive manager

Prints statistics about the Enterprise Replication receive manager. The receive manager is a set of service routines between the receive queues and data sync. The **onstat -g rcv** command is used primarily as a debugging tool and by Software Support. If you suspect that acknowledgment messages are not being applied, you can run this command.

Syntax

```
>>-onstat -g--rcv--+----------+--------------------------------><
                   +-serverid-+
                   '-full-----'
```

The *serverID* option specifies a replication server.

The **full** option prints all statistics.

The following table describes the fields in the Receive Manager global section of the **onstat -g rcv** command output.

Table 1. Receive Manager section of the onstat -g rcv output

Field	Description
cdrRM_DSParallelPL	Shows the current level of Apply Parallelism, 0 (zero) being the highest
cdrRM_DSNumLockTimeout cdrRM_DSNumLockRB cdrRM_DSNumDeadLocks	Indicate the number of collisions between various apply threads
cdrRM_acksinList	Shows acknowledgments that are received but not yet processed

The Receive Parallelism Statistics section of the **onstat -g rcv** command output shows a summary of the data sync threads by source server.

Table 2. Receive Parallelism Statistics section of the onstat -g rcv output

Field	Description
Server	Source server ID
Concur	Number of transactions currently being applied in parallel
Tot.Txn.	Total number of transactions that are applied from this source server
Pending	Number of current transactions in the pending list for this source server
Active	Number of current transactions currently being applied from this source server
MaxPnd	Maximum number of transactions in the pending list queue
MaxAct	Maximum number of transactions in the active list queue
AvgPnd	Average depth of the pending list queue
AvgAct	Average depth of the active list queue
CommitRt	Commit rate of transaction from this source server, based on transactions per second

The Statistics by Source section of the **onstat -g rcv** command output shows the following information for each source server. For each replicate ID:

- The number of transactions that are applied from the source servers
- The number of inserts, deletes, and updates within the applied transactions
- The timestamp of the most recently applied transaction on the target server
- The timestamp of the commit on the source server for the most recently applied transaction

The following example shows output for the **onstat -g rcv full** command.

```
Receive Manager global  block 0D452018
    cdrRM_inst_ct:                    5
    cdrRM_State:              00000000
    cdrRM_numSleepers:        3
    cdrRM_DsCreated:          3
    cdrRM_MinDSThreads:       1
    cdrRM_MaxDSThreads:       4
    cdrRM_DSBlock             0
    cdrRM_DSParallelPL        0
    cdrRM_DSFailRate          0.000000
    cdrRM_DSNumRun:           35
    cdrRM_DSNumLockTimeout    0
    cdrRM_DSNumLockRB         0
    cdrRM_DSNumDeadLocks      0
    cdrRM_DSNumPCommits       0
    cdrRM_ACKwaiting          0
    cdrRM_totSleep:           77
    cdrRM_Sleeptime:          153
    cdrRM_Workload:           0
    cdrRM_optscale:           4
    cdrRM_MinFloatThreads:    2
    cdrRM_MaxFloatThreads:    7
    cdrRM_AckThreadCount:     2
    cdrRM_AckWaiters:         2
    cdrRM_AckCreateStamp:Wed Sep 08 11:47:49 2010
    cdrRM_DSCreateStamp: Wed Sep 08 14:16:35 2010
    cdrRM_acksInList:         0
```

```
    cdrRM_BlobErrorBufs:      0

        Receive Parallelism Statistics
        Server Concur Tot.Txn. Pending Active MaxPnd MaxAct  AvgPnd  AvgAct CommitRt
             1      8       1       0      0      1      1    1.00    1.00     0.06

        Tot Pending:0     Tot Active:0  Avg Pending:1.00    Avg Active:1.00
        Commit Rate:0.06

        Time Spent In RM Parallel Pipeline Levels
        Lev. TimeInSec  Pcnt.
           0        40 100.00%
           1         0   0.00%
           2         0   0.00%

        Statistics by Source

        Server 1
        Repl      Txn     Ins     Del     Upd Last Target Apply   Last Source Commit
        65551       1       0       0       2 2012/10/29 09:52:23 2012/10/29 09:52:22

        No Replicates Currently Being Throttled
```

If a replicate encounters a deadlock situation or otherwise reduces the degree of parallelism by which transactions are applied, the Statistics by Source section shows the replicate and the maximum number of concurrent transactions that are possible.

```
        Statistics by Source

        Server 1
        Repl      Txn     Ins     Del     Upd Last Target Apply   Last Source Commit
        65551       1       0       0       2 2012/10/29 09:52:23 2012/10/29 09:52:22

        Replicates Being Throttled
        Repid      Max
                   Concurrent
        65551      3
```

If the replicate includes a **TimeSeries** column, a TimeSeries Statistics by Source section shows statistics about the time series elements that are applied on target servers:

```
TimeSeries Statistics by Source

Server 100
Repl        Txn   TSIns   TSDel   TSCmd Last Target Apply   Last Source Commit
65536       672     672       0       0 2012/08/27 15:04:33 2012/08/27 15:04:32
```

Related reference:
onstat -g cdr: Print ER statistics

onstat -g rep: Prints the schedule manager queue

Prints events that are in the queue for the schedule manager.

The **onstat -g rep** command is used primarily as a debugging tool and by Software Support.

The**onstat -g rep** command takes an optional replicate name to limit the output to those events originated by the specified replicate.

The following example shows sample output for the **onstat -g rep** command:

```
Schedule manager Cb: add7e18 State: 0x8100 <CDRINIT,CDRRUNNING>

Event        Thread             When
================================================
CDRDS          CDREvent       00:00:20
```

Related reference:
cdr swap shadow
onstat -g cdr: Print ER statistics

onstat -g rqm: Prints statistics for RQM queues

Prints statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM).

The RQM manages the insertion and removal of items to and from the various queues. The RQM also manages spooling of the in-memory portions of the queue to and from disk. The **onstat -g rqm** command displays the contents of the queue, size of the transactions in the queue, how much of the queue is in memory and on disk, the location of various handles to the queue, and the contents of the various progress tables. You can choose to print information for all queues or for just one queue by using one of the modifiers described below.

If a queue is empty, no information is printed for that queue.

The **onstat -g rqm** can take an optional modifier. The following table describes the values for the modifier.

Table 1. Values for the modifier to the onstat -g rqm command

Modifier	Action
ACKQ	Prints the ack send queue
CNTRLQ	Prints the control send queue
RECVQ	Prints the receive queue
SBSPACES	Prints detailed statistical information about the sbspaces configured for CDR_QDATA_SBSPACE.
SENDQ	Prints the send queue
SYNCQ	Prints the sync send queue
FULL	Prints full information about every in-memory transaction for every queue
BRIEF	Prints a brief summary of the number of transactions in each of the queues and the replication servers for which the data is queued Use this modifier to quickly identify sites where a problem exists. If large amounts of data are queued for a single server, then that server is probably down or off the network.
VERBOSE	Prints all the buffer headers in memory

When you specify a modifier to select a specific queue, the command prints all the statistics for that queue and information about the first and last in-memory transactions for that queue. When you select the SBSPACES modifier, the command prints information about the sbspaces being used for replication, including how full those sbspaces are.

The other modifiers of the **onstat -g rqm** command are used primarily as a debugging tool and by Technical Support.

The output for the SENDQ modifier contains the following sections:

- The current statistics section (`Transaction spool name` through `Pending Txn Data`): Contains information about the current contents of the queue, such as how many bytes are contained in the queue, how many transactions are in the queue, how many transactions are currently in memory, how many have been spooled to disk, how many exist only on disk, and so on. The `Insert Stamp` field value is used to maintain the order of the transactions within the queue. The `Size of Data in queue` field shows the size of the queue when combining the in-memory transactions with the spool-only transactions. The `Pending Txn Buffers` field contains information about transactions that are in the process of being queued into the send queue.
- The historical statistics section (`Max Real memory data used` through `Total Txn Lookups`): contains a summary of what has been placed in the queue in the past. The `Max Real memory data used` field contains the largest in memory size of the queue. The `Total Txn Recovered` field shows the transactions that existed only in the spool when the server was started. The `Total Txns deleted` field shows the number of transactions that have been removed from the queue. The `Total Txns duplicated` field contains the number of times attempted to queue a transaction that had already been processed. The `Total Txn Lookups` field is a counter of the number of times that an Enterprise Replication thread attempted to read a transaction.
- The `Progress Table` section: contains information on what is currently queued, to which server it is queued for, and what has been acknowledged from each of the participants of the replicate. The first part of the progress table section is a summary. Below the summary section is a list of the servers and group entries that contain what is currently queued for each server, what has been sent to the remote server, and what has been acknowledged from the remote server. The contents of the `ACKed` and `Sent` columns contains the key of the last transaction that was acknowledged from the remote server or sent to that server. The key is a multi-part number consisting of *source_node*/*unique_log_id*/*logpos*/*incremental number*. The transaction section contains the first and last transaction in the queue that are currently in memory. The `NeedAck` field shows from which server the transaction is waiting for an acknowledgment. You can use this bitmap mask with the output from the **onstat -g cat** command to determine the name of the server which server Enterprise Replication is waiting on for an acknowledgment.
- The `Transverse handle` section: contains the position within the queue that any thread is currently processing. Each thread that attempts to read a transaction from the queue, or to place a transaction into the queue must first allocate a handle. This handle is used to maintain the positioning within the queue.

The following example shows output for the **onstat -g rqm SENDQ** command.

```
> onstat  -g rqm SENDQ


CDR Reliable Queue Manager (RQM) Statistics:

RQM Statistics for Queue (0xb956020) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 9/0
 Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
 Txns in queue:              0
 Log Events in queue:        0
 Txns in memory:             0
 Txns in spool only:         0
 Txns spooled:               0
 Unspooled bytes:            0
 Size of Data in queue:      0 Bytes
 Real memory in use:         0 Bytes
 Pending Txn Buffers:        0
 Pending Txn Data:           0 Bytes
 Max Real memory data used: 385830 (4194304) Bytes
 Max Real memory hdrs used  23324 (4194304) Bytes
 Total data queued:         531416 Bytes
 Total Txns queued:         9
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        9
 Total Txns duplicated:     0
 Total Txn Lookups:         54

 Progress Table:
        Progress Table is Stable
                On-disk table name.............:          spttrg_send
                Flush interval (time)..........:          30
                Time of last flush.............:          1207866706
                Flush interval (serial number):           1000
                Serial number of last flush...:           1
                Current serial number.........:           5
```

```
Server    Group Bytes Queued      Acked                        Sent
-------------------------------------------------------------------------------
    20  0xa0002      12 effffffff/effffffff/effffffff/effffffff -  a/e/1510a1/0
    20  0xa0003                  0 a/e/4ca1b8/0                 -  a/e/4ca1b8/0
    30  0xa0004                  0 a/e/4ca1b8/0                 -  a/e/4ca1b8/0
    20  0xa0004                  0 a/e/4ca1b8/0                 -  a/e/4ca1b8/0
    20  0xa0001                  0 a/d/6e81f8/0                 -  a/d/6e81f8/0


First Txn (0x0D60C018) Key:  1/9/0x000d4bb0/0x00000000
 Txn Stamp: 1/0, Reference Count: 0.
 Txn Flags: Notify
 Txn Commit Time: (1094670993) 2004/09/08 14:16:33
 Txn Size in Queue: 5908
 First Buf's (0x0D31C9E8) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Stream
 NeedAck: Waiting for Acks from <[0004]>
 No open handles on txn.

 Last Txn (0x0D93A098) Key:  1/9/0x00138ad8/0x00000000
 Txn Stamp: 35/0, Reference Count: 0.
 Txn Flags: Notify
 Txn Commit Time: (1094671237) 2004/09/08 14:20:37
 Txn Size in Queue: 6298
 First Buf's (0x0D92FFA0) Queue Flags: Resident
 First Buf's Buffer Flags: TRG, Stream
 NeedAck: Waiting for Acks from <[0004]>
 Traverse handle (0xbca1a18) for thread CDRGeval0 at Head_of_Q,  Flags: None
 Traverse handle (0xb867020) for thread CDRACK_1 at Head_of_Q,  Flags: None
 Traverse handle (0xbcbd020) for thread CDRGeval1 at Head_of_Q,  Flags: None
 Traverse handle (0xbd08020) for thread CDRGeval3 at Head_of_Q,  Flags: None
 Traverse handle (0xbe511c8) for thread CDRGeval2 at Head_of_Q,  Flags: None
 Traverse handle (0xbe58158) for thread CDRACK_0 at Head_of_Q,  Flags: None
```

The following output is an example of the **onstat -g rqm SBSPACES** command.

```
onstat -g rqm sbspaces

Blocked:DDR


RQM Space Statistics for CDR_QDATA_SBSPACE:
------------------------------------------
name/addr      number     used       free     total    %full    pathname
0x46581c58     5          311        1        312      100      /tmp/amsterdam_sbsp_base
amsterdam_sbsp_base5      311        1        312      100

0x46e54528     6          295        17       312      95       /tmp/amsterdam_sbsp_2
amsterdam_sbsp_26         295        17       312      95

0x46e54cf8     7          310        2        312      99       /tmp/amsterdam_sbsp_3
amsterdam_sbsp_37         310        2        312      99

0x47bceca8     8          312        0        312      100      /tmp/amsterdam_sbsp_4
amsterdam_sbsp_48         312        0        312      100
```

In this example, the sbspaces are all either full or nearly full.

Related tasks:
Monitoring Disk Usage for Send and Receive Queue Spool
Related reference:
onstat -g cdr: Print ER statistics

onstat -g sync: Print statistics about synchronization

Prints statistics about the active synchronization process.

The following example shows output for the **onstat -g sync** command.

```
Prim   Sync   St.  Shadow Flag Stat Block  EndBlk
Repl   Source      Repl              Num    Num

655361 20     0    1310729 2    0    592    600
```

Output Description

Prim Repl
: Replicate number of the replicate being synchronized

Sync Source
: Source server of the sync

St
: Sync replicate state

Shadow Repl
: The shadow replicate used to perform the sync

Flag
: Internal flags:

  - 0x02 = external sync
  - 0x04 = shutdown request has been issued
  - 0x08 = abort has occurred
  - 0x010 = a replicate stop has been requested
  - 0X020 = shadow or primary replicate has been deleted

Stat
: Resync job state

Block num
: Last block applied on targets (on source always 0)

EndBlock Num
: Last block in resync process. Marks the end of the sync scan on the target. A value of' -2 indicates that the scan is still in progress, and the highest block number is not yet known.

Additional fields for forwarded rows:

ServID
: Server where forwarded row originated

fwdLog ID
: Originator's log ID of the forwarded row

fwdLog POS
: Originator's log position of the forwarded row

endLog ID
: Operation switches back to normal at this point

endLog POS
: Operation switches back to normal at this log position

complete flag
: Set to 1 after normal processing resumes for the originating source

syscdr Tables

These tables in the **syscdr** database contain progress information about consistency checking and synchronization operations.

- The replcheck_stat Table
  The **replcheck_stat** table contains the progress information for consistency check and synchronization operations that specified a progress report task name.
- The replcheck_stat_node Table
  The **replcheck_stat_node** table contains the progress information for the consistency check and synchronization operations with progress report task names on a particular replication server.

The replcheck_stat Table

The **replcheck_stat** table contains the progress information for consistency check and synchronization operations that specified a progress report task name.

Column	Type	Purpose
replcheck_id	serial	Unique key for the task and replicate combination.
replcheck_name	varchar(32)	The task name.
replcheck_replname	varchar(128)	The replicate name.
replcheck_type	char(1)	The task type: • C = consistency check • S = synchronization
replcheck_numrows	integer	The total number of rows in the table.
replcheck_rows_processed	integer	The number of rows that were processed to correct inconsistent rows.
replcheck_status	char(1)	The status of this task: • A = Aborted • D = Defined • R = Running • C = Completed • F = Completed, but inconsistent • W = Pending complete

Column	Type	Purpose
replcheck_start_time	datetime year to second	The time that the sync or check task for the replicate started running.
replcheck_end_time	datetime year to second	The time that sync or check task for the replicate completed.

Related reference:
The replcheck_stat_node Table
cdr stats check
cdr stats sync

The replcheck_stat_node Table

The **replcheck_stat_node** table contains the progress information for the consistency check and synchronization operations with progress report task names on a particular replication server.

Column	Type	Purpose
replnode_replcheck_id	integer	Server group ID (CDR ID).
replcheck_node_id	integer	Unique key for the task, replicate, and server combination.
replcheck_order	integer	A number to provide consistent ordering for display purposes.
replcheck_node_name	varchar(128)	The name of the replication server.
replnode_table_owner	varchar(128)	The owner of table being synchronized or checked.
replnode_table_name	varchar(128)	The name of the table being synchronized or checked.
replnode_row_count	integer	The number of rows in the participant.
replnode_processed_rows	integer	The number of rows processed to correct inconsistent rows.
replnode_missing_rows	integer	The number of rows on the reference server that do not exist on the target server.
replnode_extra_rows	integer	The number of rows on the target server that do not exist on the reference server.
replnode_mismatched_rows	integer	The number of rows on the target server that are not consistent with the corresponding rows on the reference server.
replnode_extra_child_rows	integer	The number of child rows that required processing on the target nodes.

Related reference:
The replcheck_stat Table
cdr stats check
cdr stats sync

SMI Tables for Enterprise Replication Reference

The system-monitoring interface (SMI) tables in the **sysmaster** database provide information about the state of the database server. Enterprise Replication uses the following SMI tables.

- The syscdr_ats Table
  The **syscdr_ats** table contains the first ten lines of the transaction header for each ATS file.
- The syscdr_atsdir Table
  The **syscdr_atsdir** table contains information about the contents of the ATS directory.
- The syscdr_ddr Table
  The **syscdr_ddr** table contains information about the status of log capture and the proximity or status of transaction blocking (DDRBLOCK) or transaction spooling.
- The syscdr_nif Table
  The **syscdr_nif** table contains information about network connections and the flow of data between Enterprise Replication servers.
- The syscdr_rcv Table
  The **syscdr_rcv** table contains information about transactions being applied on target servers and acknowledgments being sent from target servers.
- The syscdr_ris Table
  The **syscdr_ris** table contains the first ten lines of the transaction header for each RIS file.
- The syscdr_risdir Table
  The **syscdr_risdir** table contains information about the contents of the RIS directory.
- The syscdr_rqm Table
  The **syscdr_rqm** table contains statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM).

- The syscdr_rqmhandle Table
  The **syscdr_rqmhandle** table contains information about which transaction is being processed in each queue. The handle marks the position of the thread in the queue.
- The syscdr_rqmstamp Table
  The **syscdr_rqmstamp** table contains information about which transaction is being added to each queue.
- The syscdr_state Table
  The **syscdr_state** table contains status on Enterprise Replication, data capture, data apply, and the network between the servers.
- The syscdrack_buf Table
  The **syscdrack_buf** table contains information about the buffers that form the acknowledgment queue.
- The syscdrack_txn Table
  The **syscdrack_txn** table contains information about the acknowledgment queue.
- The syscdrctrl_buf Table
  The **syscdrctrl_buf** table contains buffers that provide information about the control queue. The control queue is a stable queue that contains control messages for the replication system.
- The syscdrctrl_txn Table
  The **syscdrctrl_txn** table contains information about the control queue. The control queue is a stable queue that contains control messages for the replication system.
- The syscdrerror Table
  The **syscdrerror** table contains information about errors that Enterprise Replication has encountered.
- The syscdrlatency Table
  The **syscdrlatency** table contains statistics about Enterprise Replication latency (the time it takes to replicate transactions).
- The syscdrpart Table
  The **syscdrpart** table contains participant information.
- The syscdrprog Table
  The **syscdrprog** table lists the contents of the Enterprise Replication progress tables.
- The syscdrq Table
  The **syscdrq** table contains information about Enterprise Replication queues.
- The syscdrqueued Table
  The **syscdrqueued** table contains data-queued information.
- The syscdrrecv_buf Table
  The **syscdrrecv_buf** table contains buffers that provide information about the data-receive queue.
- The syscdrrecv_stats Table
  The **syscdrrecv_stats** table contains statistics about the receive manager. The receive manager is a set of service routines between the receive queues and data sync.
- The syscdrrecv_txn Table
  The **syscdrrecv_txn** table contains information about the data receive queue. The receive queue resides in memory.
- The syscdrrepl Table
  The **syscdrrepl** table contains replicate information.
- The syscdrreplset Table
  The **syscdrreplset** table contains replicate set information.
- The syscdrs Table
  The **syscdrs** table contains information about database servers in an Enterprise Replication domain.
- The syscdrsend_buf Table
  The **syscdrsend_buf** table contains buffers that give information about the send queue.
- The syscdrsend_txn Table
  The **syscdrsend_txn** table contains information about the send queue.
- The syscdrserver Table
  The **syscdrserver** table contains information about database servers declared to Enterprise Replication.
- The syscdrsync_buf Table
  The **syscdrsync_buf** table contains buffers that give information about the synchronization queue. Enterprise Replication uses this queue only when defining a replication server and synchronizing its global catalog with another replication server.
- The syscdrsync_txn Table
  The **syscdrsync_txn** table contains information about the synchronization queue. This queue is currently used only when defining a replication server and synchronizing its global catalog with another replication server. The synchronization queue is an in-memory-only queue.
- The syscdrtsapply table
  The **syscdrtsapply** table lists statistics about the time series elements that are applied on target servers.
- The syscdrtx Table
  The **syscdrtx** table contains information about Enterprise Replication transactions.
- Enterprise Replication Queues

Related concepts:
Monitor and troubleshooting Enterprise Replication

The syscdr_ats Table

The **syscdr_ats** table contains the first ten lines of the transaction header for each ATS file.

Column	Type	Description
ats_ris	integer	Pseudo row ID.
ats_file	char(128)	ATS file name.
ats_sourceid	integer	CDRID of source server.
ats_source	char(128)	Source server name.

Column	Type	Description
ats_committime	char(20)	Time when the transaction was committed on the source server.
ats_targetid	integer	CDRID of the target server.
ats_target	char(128)	Target server name.
ats_receivetime	char(20)	Time when the transaction was received on the target server .
ats_risfile	char(128)	Corresponding RIS file name.
ats_line1	char(200)	The first line of the transaction header information.
ats_line2	char(200)	The second line of the transaction header information.
ats_line3	char(200)	The third line of the transaction header information.
ats_line4	char(200)	The fourth line of the transaction header information.
ats_line5	char(200)	The fifth line of the transaction header information.
ats_line6	char(200)	The sixth line of the transaction header information.
ats_line7	char(200)	The seventh line of the transaction header information.
ats_line8	char(200)	The eighth line of the transaction header information.
ats_line9	char(200)	The ninth line of the transaction header information.
ats_line10	char(200)	The tenth line of the transaction header information.

The syscdr_atsdir Table

The **syscdr_atsdir** table contains information about the contents of the ATS directory.

Column	Type	Description
atsd_rid	integer	Pseudo row ID
atsd_file	char(128)	ATS file name
atsd_mode	integer	File mode
atsd_size	integer	File size in bytes
atsd_atime	datetime	Last access time
atsd_mtime	datetime	Last modified time
atsd_ctime	datetime	Create time

The syscdr_ddr Table

The **syscdr_ddr** table contains information about the status of log capture and the proximity or status of transaction blocking (DDRBLOCK) or transaction spooling.

Column	Type	Description
ddr_state	char(24)	The current state of log capture: • Running = Log capture is running normally • Down = Log capture is not running • Uninitialized = The server is not a source server for replication
ddr_snoopy_loguniq	integer	The current log ID at which transactions are being captured for replication
ddr_snoopy_logpos	integer	The current log position at which transactions are being captured for replication
ddr_replay_loguniq	integer	The current log ID at which transactions have been applied
ddr_replay_logpos	integer	The current log position at which transactions have been applied. This is the position from which the log would need to be replayed to recover Enterprise Replication if Enterprise Replication or the database server shut down.
ddr_curr_loguniq	integer	The current log ID
ddr_curr_logpos	integer	The current log position
ddr_logsnoop_cached	integer	The number of log pages that log capture read from its cache
ddr_logsnoop_disk	integer	The number of times that log capture had to read log pages from disk
ddr_log_tossed	integer	The number of log pages that could not be stored in the cache because the log capture buffer cache was full

Column	Type	Description
ddr_logs_ignored	integer	The number of log records that were ignored because they were extensible log records unknown to Enterprise Replication
ddr_dlog_requests	integer	The number of times that a dynamic log was requested to be created to prevent DDRBLOCK state
ddr_total_logspace	integer	The total number of log pages in the replication system
ddr_logspace2wrap	integer	The number of log spaces until log capture runs into a log wrap
ddr_logpage2block	integer	The number of log pages until log capture runs into a DDRBLOCK state
ddr_logneeds	integer	The number of log pages necessary to prevent a log wrap to avoid a DDRBLOCK state
ddr_logcatchup	integer	The number of log pages necessary to process before going out of a DDRBLOCK state
ddr_loglag_state	char(10)	The state of DDR log lag: on or off
ddr_cur_loglag_act	char(24)	The action being taken to prevent log wrapping
ddr_logstage_diskusage	float	The amount of used log staging disk space as a percentage of the total space
ddr_logstage_hwm4disk	integer	The maximum allowable disk space for log staging in KB
ddr_logstage_maxused	float	The maximum disk space ever used for log staging in KB
ddr_logstage_lfile_cnt	integer	The number of staged log files

The syscdr_nif Table

The **syscdr_nif** table contains information about network connections and the flow of data between Enterprise Replication servers.

Column	Type	Description
nif_connid	integer	The CDRID of the peer node
nif_connname	char(24)	The name (group name) of the peer node
nif_state	char(24)	The status of the Enterprise Replication network: • Admin Close = Enterprise Replication was stopped by user by issuing the **cdr stop** command • Connected = The connection is active • Connecting = The connection is being established • Disconnected = The connection was explicitly disconnected • Local server = The connection is to the local server. • Logic Error = The connection disconnected due to an error during message transmission • Never Connected = The servers have never had an active connection • Start Error = The connection disconnected due to an error while starting a thread to receive remote messages • Timeout = The connection attempt has timed out, but will be reattempted
nif_connstate	char(24)	The connection state: • ABORT = The connection is being aborted. • BLOCK = The connection has been blocked from transmitting data by the other server. • INIT = The connection is being initialized. • INTR = The connection has been interrupted. • RUN = The connection is active. • SHUT = The connection is shutting down in an orderly way. • SLEEP = The connection is waiting to receive data. • STOP = The connection is stopped. • SUSPEND = The connection has been suspended. • TIMEOUT = The connection has timed out.
nif_version	integer	The network protocol of this connection used to convert the message formats between dissimilar releases of the server, for example, IBM® Informix® 7 and IBM Informix 9
nif_msgsent	integer	Number of messages sent to the peer server
nif_bytessent	integer	Number of bytes sent to the peer server
nif_msgrcv	integer	Number of messages received from the peer server
nif_bytesrcv	integer	Number of bytes received from the peer server
nif_compress	integer	Compression level for communications • -1 = no compression • 0 = compress only if the target server expects compression • 1 - 9 = increasing levels of compression
nif_sentblockcnt	integer	Number of times a flow block request was sent to the peer server to delay sending any further replicated transactions for a short time because the receive queue on the target server is full

Column	Type	Description
nif_rcvblockcnt	integer	Number of times a flow block request was received from the peer server
nif_trgsend_stamp1	integer	Stamp 1 of the last transaction sent to the peer server
nif_trgsend_stamp2	integer	Stamp 2 of the last transaction sent to the peer server
nif_acksend_stamp1	integer	Stamp 1 of the last acknowledgment sent to the peer server
nif_acksend_stamp2	integer	Stamp 2 of last acknowledgment sent to the peer server
nif_ctrlsend_stamp1	integer	Stamp 1 of the last control message sent to the peer server
nif_ctrlsend_stamp2	integer	Stamp 2 of the last control message sent to the peer server
nif_syncsend_stamp1	integer	Stamp 1 of the last sync message sent to the peer server
nif_syncsend_stamp2	integer	Stamp 2 of the last sync message sent to the peer server
nif_starttime	datetime	Time that the connection was established
nif_lastsend	datetime	Time of the last message sent to the peer server

The syscdr_rcv Table

The **syscdr_rcv** table contains information about transactions being applied on target servers and acknowledgments being sent from target servers.

Column	Type	Description
rm_state	char(100)	The status of the receive manager and apply threads: • Running = Transaction apply is running normally • Down = Transaction apply is not running • Uninitialized = The server is not a source server for replication
rm_num_sleepers	integer	Number of data sync threads currently suspended
rm__num_dsthreads	integer	The current number of data sync threads
rm_min_dsthreads	integer	Minimum number of data sync threads
rm_max_dsthreads	integer	Maximum number of data sync threads
rm_ds_block	integer	If `1`, the data sync is currently blocked to try to avoid causing a DDRBLOCK state
rm_ds_parallel	integer	The degree to which transactions are applied in parallel (0 through 3, inclusive): • `0` = the highest degree of parallelism • `3` = serial apply (no parallelism)
rm_ds_failrate	float	A computed weighted ratio that is used to determine when to change the degree of apply parallelism based on the rate of transactions that could not be applied
rm_ds_numrun	integer	Number of transactions run
rm_ds_lockout	integer	Number of lock timeouts encountered
rm_ds_lockrb	integer	Number of forced rollbacks due to having to switch to serial apply
rm_ds_num_deadlocks	integer	Number of deadlocks encountered
rm_ds_num_pcommits	integer	Number of out-of-order commits that have occurred
rm_ack_waiting	integer	Number of acknowledgments that are waiting for a log flush to return to the source server
rm_tosleep	integer	Total times that the data sync threads have become suspended
rm_sleeptime	integer	Total time that the data sync threads have been suspended
rm_workload	integer	The current workload
rm_optscale	integer	Factor determining how many data sync threads will be allowed per CPU VP
rm_min_fthreads	integer	Minimum acknowledgment threads
rm_max_fthreads	integer	Maximum acknowledgment threads
rm_ack_start	char(64)	Time when the acknowledgment threads started
rm_ds_start	char(64)	Time when the data sync threads started
rm_pending_acks	integer	Number of acknowledgments on the source that have not yet been processed
rm_blob_error_bufs	integer	Number of smart large objects that could not be successfully applied

The syscdr_ris Table

The **syscdr_ris** table contains the first ten lines of the transaction header for each RIS file.

Column	Type	Description
ris_rid	integer	Pseudo row ID.
ris_file	char(128)	RIS file name.
ris_sourceid	integer	CDRID of source server.
ris_source	char(128)	Source server name.
ris_committime	char(20)	Time when the transaction was committed on the source server.
ris_targetid	char(128)	CDRID of the target server.
ris_target	integer	Target server name.
ris_receivetime	char(20)	Time when the transaction was received on the target server.
ris_atsfile	char(128)	Corresponding ATS file.
ris_line1	char(200)	The first line of the transaction header information.
ats_line2	char(200)	The second line of the transaction header information.
ats_line3	char(200)	The third line of the transaction header information.
ats_line4	char(200)	The fourth line of the transaction header information.
ris_line5	char(200)	The fifth line of the transaction header information.
ris_line6	char(200)	The sixth line of the transaction header information.
ris_line7	char(200)	The seventh line of the transaction header information.
ris_line8	char(200)	The eighth line of the transaction header information.
ris_line9	char(200)	The ninth line of the transaction header information.
ris_line10	char(200)	The tenth line of the transaction header information.

The syscdr_risdir Table

The **syscdr_risdir** table contains information about the contents of the RIS directory.

Column	Type	Description
risd_rid	integer	Pseudo row ID
risd_file	char(128)	RIS file name
risd_mode	integer	File mode
risd_size	integer	File size in bytes
risd_atime	datetime	Last access time
risd_mtime	datetime	Last modified time
risd_ctime	datetime	Create time

The syscdr_rqm Table

The **syscdr_rqm** table contains statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM).

The RQM manages the insertion and removal of items to and from the various queues. The RQM also manages spooling of the in-memory portions of the queue to and from disk.

Column	Type	Description
rqm_idx	integer	Index number
rqm_name	char(128)	Queue name
rqm_flags	integer	Flags
rqm_txn	integer	Transactions in queue
rqm_event	integer	Events in queue
rqm_txn_in_memory	integer	Transaction in memory
rqm_txn_in_spool_only	integer	Spool-only transactions
rqm_txn_spooled	integer	Spooled transactions

Column	Type	Description
rqm_unspooled_bytes	int8	Unspooled bytes
rqm_data_in_queue	int8	Data in queue
rqm_inuse_mem	int8	Real memory in use
rqm_pending_buffer	integer	Pending buffers
rqm_pending_data	int8	Pending buffers
rqm_maxmemdata	int8	Maximum memory in use by data
rqm_maxmemhdr	int8	Maximum memory in use by headers
rqm_totqueued	int8	Total data queued
rqm_tottxn	integer	Total transactions queued
rqm_totspooled	integer	Total transactions spooled
rqm_totrestored	integer	Total transactions stored
rqm_totrecovered	integer	Total transactions recovered
rqm_totspoolread	integer	Total rows read from spool
rqm_totdeleted	integer	Total transactions deleted
rqm_totduplicated	integer	Total transactions duplicates
rqm_totlookup	integer	Total transaction lookups

The syscdr_rqmhandle Table

The **syscdr_rqmhandle** table contains information about which transaction is being processed in each queue. The handle marks the position of the thread in the queue.

Column	Type	Description
rqmh_qidx	integer	The queue associated with this handle
rqmh_thread	char(18)	Thread owning the handle
rqmh_stamp1	integer	Stamp 1 of the last transaction this handle accessed
rqmh_stamp2	integer	Stamp 2 of the last transaction this handle accessed
rqmh_servid	integer	Part 1 of the transaction key
rqmh_logid	integer	Part 2 of the transaction key
rqmh_logpos	integer	Part 3 of the transaction key
rqmh_seq	integer	Part 4 of the transaction key

The syscdr_rqmstamp Table

The **syscdr_rqmstamp** table contains information about which transaction is being added to each queue.

Column	Type	Description
rqms_qidx	integer	Queue index corresponding to the queues: • 0 = Transaction Send Queue • 1 = Acknowledgment Send Queue • 2 = Control Send Queue • 3 = CDR Metadata Sync Send Queue • 4 = Transaction Receive Queue
rqms_stamp1	integer	Stamp 1 of the next transaction being put into the queue
rqms_stamp2	integer	Stamp 2 of the next transaction being put into the queue
rqms_cstamp1	integer	Communal stamp 1 used to identify the next transaction read from the receive queue
rqms_cstamp2	integer	Communal stamp 2 used to identify the next transaction read from the receive queue

The syscdr_state Table

The **syscdr_state** table contains status on Enterprise Replication, data capture, data apply, and the network between the servers.

Column	Type	Description
er_state	char(24)	The status of Enterprise Replication: • Abort = Enterprise Replication is aborting on this server. • Active = Enterprise Replication is running normally. • Down = Enterprise Replication is stopped on this server. • Dropped = The attempt to drop the **syscdr** database failed. • Init Failed = The initial start-up of Enterprise Replication on this server failed, most likely because of a problem on the specified global catalog synchronization server. • Initializing = Enterprise Replication is being defined. • Initial Startup = Enterprise Replication is starting for the first time on this server. • Shutting Down = Enterprise Replication is shutting down on this server. • Startup Blocked = Enterprise Replication cannot start because the server was started with the **oninit -D** command. • Synchronizing Catalogs = The server is receiving a copy of the **syscdr** database. • Uninitialized = The server does not have Enterprise Replication defined on it.
er_capture_state	char(24)	The current state of log capture: • Running = Log capture is running normally • Down = Log capture is not running • Uninitialized = The server is not a source server for replication
er_network_state	char(64)	The status of the Enterprise Replication network: • Running = Communication is running normally • Down = Communication is not running • Uninitialized = The server is not a source server for replication
er_apply_state	char(24)	The status of the receive manager and apply threads: • Running = Transaction apply is running normally • Down = Transaction apply is not running • Uninitialized = The server is not a source server for replication

The syscdrack_buf Table

The **syscdrack_buf** table contains information about the buffers that form the acknowledgment queue.

When the target database server applies transactions, it sends an acknowledgment to the source database server. When the source database server receives the acknowledgment, it can then delete those transactions from its send queue.

For information on the columns of the **syscdrack_buf** table, refer to Columns of the Buffer Tables.

The syscdrack_txn Table

The **syscdrack_txn** table contains information about the acknowledgment queue.

When the target database server applies transactions, it sends an acknowledgment to the source database server. When the source database server receives the acknowledgment, it can then delete those transactions from its send queue. The acknowledgment queue is an in-memory only queue. That is, it is a volatile queue that is lost if the database server is stopped.

For information on the columns of the **syscdrack_txn** table, refer to Columns of the Transaction Tables.

The syscdrctrl_buf Table

The **syscdrctrl_buf** table contains buffers that provide information about the control queue. The control queue is a stable queue that contains control messages for the replication system.

For information on the columns of the **syscdrctrl_buf** table, refer to Columns of the Buffer Tables.

The syscdrctrl_txn Table

The **syscdrctrl_txn** table contains information about the control queue. The control queue is a stable queue that contains control messages for the replication system.

For information on the columns of the **syscdrctrl_txn** table, refer to Columns of the Transaction Tables.

The syscdrerror Table

The **syscdrerror** table contains information about errors that Enterprise Replication has encountered.

Column	Type	Description
errornum	integer	Error number
errorserv	char(128)	Database server name where error occurred
errorseqnum	integer	Sequence number that can be used to prune single-error table
errortime	datetime year to second	Time error occurred
sendserv	char(128)	Database server name, if applicable, that initiated error behavior
reviewed	char(1)	• `Y` if reviewed and set by DBA • `N` if not reviewed
errorstmnt	text	Error description

The syscdrlatency Table

The **syscdrlatency** table contains statistics about Enterprise Replication latency (the time it takes to replicate transactions).

Column	Type	Description
source	integer	Source of transaction (**cdrid**)
replid	integer	Replicate ID
txncnt	integer	The number of transactions on this source replicate
inserts	integer	Number of INSERT statements
deletes	integer	Number of DELETE statements
updates	integer	Number of UPDATE statements
last_tgt_apply	integer	The time of the last transaction to be applied to the target (**cdrtime**)
last_src_apply	integer	The time of the last transaction to be applied on the source (**cdrtime**)

The syscdrpart Table

The **syscdrpart** table contains participant information.

Column	Type	Description
replname	lvarchar	Replicate name
servername	char(128)	Database server name
partstate	char(50)	Participant state: ACTIVE, INACTIVE
partmode	char(1)	• `P` = primary database server (read/write) • `R` = target database server (receive only)
dbsname	lvarchar	Database name
owner	lvarchar	Owner name
tabname	lvarchar	Table name

Column	Type	Description
pendingsync	integer	• 0 = the Pending Sync attribute is not set • 1 = the Pending Sync attribute is set, indicating that the participant is waiting to be synchronized after the replication server was enabled

The syscdrprog Table

The **syscdrprog** table lists the contents of the Enterprise Replication progress tables.

The progress tables keep track of what data has been sent to which servers and which servers have acknowledged receipt of what data. Enterprise Replication uses the transaction keys and stamps to keep track of this information.

The progress table is two dimensional. For each server to which Enterprise Replication sends data, the progress tables keep progress information on a per-replicate basis.

Column	Type	Description
dest_id	integer	Server ID of the destination server
repl_id	integer	The ID that Enterprise Replication uses to identify the replicate for which this information is valid
source_id	integer	Server ID of the server from which the data originated
key_acked_srv	integer	Last key for this replicate that was acknowledged by this destination
key_acked_lgid	integer	Logical log ID
key_acked_lgpos	integer	Logical log position
key_acked_seq	integer	Logical log sequence
tx_stamp_1	integer	Together with tx_stamp2, forms the stamp of the last transaction acknowledged for this replicate by this destination
tx_stamp_2	integer	Together with tx_stamp1, forms the stamp of the last transaction acknowledged for this replicate by this destination

The syscdrq Table

The **syscdrq** table contains information about Enterprise Replication queues.

Column	Type	Description
srvid	integer	The identifier number of the database server
repid	integer	The identifier number of the replicate
srcid	integer	The server ID of the source database server In cases where a particular server is forwarding data to another server, *srvid* is the target and *srcid* is the source that originated the transaction.
srvname	char(128)	The name of the database server
replname	char(128)	Replicate name
srcname	char(128)	The name of the source database server
bytesqueued	integer	Number of bytes queued

The syscdrqueued Table

The **syscdrqueued** table contains data-queued information.

Column	Type	Description
servername	char(128)	Sending to database server name
name	char(128)	Replicate name
bytesqueued	decimal(32,0)	Number of bytes queued for the server *servername*

The syscdrrecv_buf Table

The **syscdrrecv_buf** table contains buffers that provide information about the data-receive queue.

When a replication server receives replicated data from a source database server, it puts this data on the receive queue for processing. On the target side, Enterprise Replication picks up transactions from this queue and applies them on the target.

For information on the columns of the **syscdrrecv_buf** table, refer to Columns of the Buffer Tables.

The syscdrrecv_stats Table

The **syscdrrecv_stats** table contains statistics about the receive manager. The receive manager is a set of service routines between the receive queues and data sync.

Column	Type	Description
source	integer	The source server (**cdrid**)
txncnt	integer	Number of transactions from this source
pending	integer	The transaction currently pending on this source
active	integer	The transaction currently active on this source
maxpending	integer	Maximum pending transactions on this source
maxactive	integer	Maximum active transactions on this source
avg_pending	float	Average pending transactions on this source
avg_active	float	Average active transactions on this source
cmtrate	float	Average commit rate from this source

The syscdrrecv_txn Table

The **syscdrrecv_txn** table contains information about the data receive queue. The receive queue resides in memory.

When a replication server receives replicated data from a source database server, it puts the data in the receive queue and then applies the transactions on the target.

For information on the columns of the **syscdrrecv_txn** table, refer to Columns of the Transaction Tables.

The syscdrrepl Table

The **syscdrrepl** table contains replicate information.

Column	Type	Description
replname	lvarchar	Replicate name.
replstate	char(50)	Replicate state. For possible values, refer to cdr list server.
freqtype	char(1)	Type of replication frequency: • `C` = continuous • `I` = interval • `T` = time based • `M` = day of month • `W` = day of week
freqmin	smallint	The time for replication by minute: • Minutes after the hour that replication should occur. • Null if continuous.
freqhour	smallint	The time for replication by hour: • Hour that replication should occur. • Null if continuous.
freqday	smallint	Day of week or month replication should occur.

Column	Type	Description
scope	char(1)	Replication scope: • T = transaction • R = row-by-row
invokerowspool	char(1)	Whether Row Information Spooling is enabled: • Y = row spooling is enabled. • N = row spooling is disabled.
invoke transpool	char(1)	Whether Aborted Transaction Spooling is enabled: • Y = transaction spooling is enabled. • N = transaction spooling is disabled.
primresolution	char(1)	Type of primary conflict resolution: • A = always apply • D = delete wins • I = ignore • T = timestamp. • S = SPL routine
secresolution	char(1)	Type of secondary conflict resolution: • S = SPL routine • Null = not configured
storedprocname	lvarchar	SPL routine: • Name of SPL routine for secondary conflict resolution. • Null if not defined.
floattype	char(1)	Type of floating point number conversion: • C= converts floating point numbers to canonical format. • I= converts floating point numbers to IEEE format. • N = does not convert floating point numbers (sends in native format).
istriggerfire	char(1)	Whether triggers are enabled: • Y = triggers are enabled. • N = triggers are disabled.
isfullrow	char(1)	Whether to replicate full rows or only the changed columns: • Y = sends the full row and enables upserts. • N = sends only changed columns and disables upserts.
isgrid	char(1)	Whether the replicate belongs to a grid replicate set: • Y = the replicate belongs to a grid replicate set. • N = the replicate does not belong to a grid replicate set.

Related tasks:
Viewing grid information

The syscdrreplset Table

The **syscdrreplset** table contains replicate set information.

Column	Type	Description
replname	lvarchar	Replicate name
replsetname	lvarchar	Replicate set name
replsetattr	integer	Replicate set attributes: • 0x00200000U = The replicate set was created with a template. • 0x00000080U = The replicate set is exclusive.

The syscdrs Table

The **syscdrs** table contains information about database servers in an Enterprise Replication domain.

Column	Type	Description
servid	integer	Server identifier.
servname	char(128)	Database server name.
cnnstate	char(1)	Status of connection to this database server: • C = Connected • D = Connection disconnected (will be retried) • E = Error during connection • F = Connection failed • K = In the process of connecting • L = The connection is to the local server • R = Disconnected but will attempt to reconnect • T = Idle time-out caused connection to terminate • X = Connection closed by user command and unavailable until reset by user
cnnstatechg	integer	Time that connection state was last changed.
servstate	char(1)	Status of database server: • A = Active. The server is active and replicating data. • D = Deleted. The server has been deleted; it is not capturing or delivering data and the queues are being drained. • S = Suspended. Delivery of replication data to the server is suspended. • Q = Quiescent. The server is in the process of being defined. • U = Disabled. Only delete shadow tables are populated in this state.
ishub	char(1)	Whether the server is a hub server that forwards information to another replication server: • Y = Server is a hub • N = Server is not a hub
isleaf	char(1)	Whether the server is a leaf or a nonleaf server: • Y = Server is a leaf server • N = Server is not a leaf server
rootserverid	integer	The identifier of the root server.
forwardnodeid	integer	The identifier of the parent server.
timeout	integer	The number of minutes of idle time between replication servers before the connection is timed out.

Although not directly connected, a nonroot server is similar to a root server except it forwards all replicated messages through its parent (root) server. All nonroot servers are known to all root servers and other nonroot servers. A nonroot server can be a terminal point in a tree or it can be the parent for another nonroot server or a leaf server. Nonroot and root servers are aware of all replication servers in the replication environment, including all the leaf servers.

A leaf server is a nonroot server that has a partial catalog. A leaf server has knowledge only of itself and its parent server. It does not contain information about replicates of which it is not a participant. The leaf server must be a terminal point in a replication hierarchy.

Related concepts:
Hierarchical Routing Topology Terminology

The syscdrsend_buf Table

The **syscdrsend_buf** table contains buffers that give information about the send queue.

When a user performs transactions on the source database server, Enterprise Replication queues the data on the send queue for delivery to the target servers.

For information on the columns of the **syscdrsend_buf** table, refer to Columns of the Buffer Tables.

The syscdrsend_txn Table

The **syscdrsend_txn** table contains information about the send queue.

When a user performs transactions on the source database server, Enterprise Replication queues the data on the send queue for delivery to the target servers.

For information on the columns of the **syscdrsync_txn** table, refer to Columns of the Transaction Tables.

The syscdrserver Table

The **syscdrserver** table contains information about database servers declared to Enterprise Replication.

Column	Type	Description
servid	integer	Replication server ID
servername	char(128)	Database server group name
connstate	char(1)	Status of connection to this database server: • C = Connected • D = Connection disconnected (will be retried) • T = Idle time-out caused connection to terminate • X = Connection closed by user command and unavailable until reset by user
connstatechange	integer	Time that connection state was last changed
servstate	char(50)	Status of this database server: • A = Active • D = Disabled • S = Suspended • Q = Quiescent (initial sync state only)
ishub	char(1)	• Y = Server is a hub • N = Server is not a hub
isleaf	char(1)	• Y = Server is a leaf server • N = Server connection is not a leaf server
rootserverid	integer	The identifier of the root server
forwardnodeid	integer	The identifier of the parent server
idletimeout	integer	Idle time-out
atsdir	lvarchar	ATS directory spooling name
risdir	lvarchar	RIS directory spooling name

The syscdrsync_buf Table

The **syscdrsync_buf** table contains buffers that give information about the synchronization queue. Enterprise Replication uses this queue only when defining a replication server and synchronizing its global catalog with another replication server.

For information on the columns of the **syscdrsync_buf** table, refer to Columns of the Buffer Tables.

The syscdrsync_txn Table

The **syscdrsync_txn** table contains information about the synchronization queue. This queue is currently used only when defining a replication server and synchronizing its global catalog with another replication server. The synchronization queue is an in-memory-only queue.

For information on the columns of the **syscdrsync_txn** table, refer to Columns of the Transaction Tables.

The syscdrtsapply table

The **syscdrtsapply** table lists statistics about the time series elements that are applied on target servers.

Table 1. The syscdrtsapply table

Column	Type	Description
source	integer	CDRID of source server.
replid	integer	Replicate ID.
txncnt	integer	The number of transactions from this source server and replicate.
tsinserts	integer	The number of time series elements that were inserted.
tsdeletes	integer	The number of time series elements that were deleted.
tscmd	integer	The number of TimeSeries routines that inserted or deleted elements that are replicated.
last_tgt_apply	integer	The time when the most recent transaction was applied on a target server.
last_src_apply	integer	The time when the most recent transaction was applied on the source server.

The syscdrtx Table

The **syscdrtx** table contains information about Enterprise Replication transactions.

Column	Type	Description
srvid	integer	Server ID
srvname	char(128)	Name of database server from which data is received
txprocssd	integer	Transaction processed from database server *srvname*
txcmmtd	integer	Transaction committed from database server *srvname*
txabrtd	integer	Transaction aborted from database server *srvname*
rowscmmtd	integer	Rows committed from database server *srvname*
rowsabrtd	integer	Rows aborted from database server *srvname*
txbadcnt	integer	Number of transactions with source commit time (on database server *srvname*) greater than target commit time

Enterprise Replication Queues

One group of **sysmaster** tables shows information about Enterprise Replication queues. The **sysmaster** database reports the status of these queues in the tables that have the suffixes **_buf** and **_txn**.

The name of each table that describes an Enterprise Replication queue is composed of the following three pieces:

- **syscdr**, which indicates that the table describes Enterprise Replication
- An abbreviation that indicates which queue the table describes
- A suffix, either **_buf** or **_txn**, which specifies whether the table includes buffers or transactions

Selecting from these tables provides information about the contents of each queue. For example, the following SELECT statement returns a list of all transactions queued on the send queue:

```
SELECT * FROM syscdrsend_txn
```

The following example returns a list of all transactions queued on the in-memory send queue and returns the number of buffers and the size of each buffer for each transaction on the send queue:

```
SELECT cbkeyserverid,cbkeyid,cbkeypos,count(*),sum(cbsize)
   from syscdrsend_buf
   group by cbkeyserverid, cbkeyid, cbkeypos
   order by cbkeyserverid, cbkeyid, cbkeypos
```

All queues are present on all the replication servers, regardless of whether the replication server is a source or a target for a particular transaction. Some of the queues are always empty. For instance, the send queue on a target-only server is always empty.

Each queue is two-dimensional. Every queue has a list of transaction headers. Each transaction header in turn has a list of buffers that belong to that transaction.

- Columns of the Transaction Tables
  The transaction tables contain information about transactions that are in memory. They do not contain information about transactions that are spooled to disk.
- Columns of the Buffer Tables
  The buffer tables contain information about the buffers that form the transactions that are listed in the transaction tables.

Columns of the Transaction Tables

The transaction tables contain information about transactions that are in memory. They do not contain information about transactions that are spooled to disk.

The names of transaction tables end with **_txn**. All transaction tables have the same columns and the same column definitions.

The **ctstamp1** and **ctstamp2** columns combine to form the primary key for these tables.

Column	Type	Description
ctkeyserverid	integer	Server ID of the database server where this data originated. This server ID is the group ID from the sqlhosts file.
ctkeyid	integer	Logical log ID.
ctkeypos	integer	Position in the logical log on the source server for the transaction that is represented by the buffer.
ctkeysequence	integer	Sequence number for the buffer within the transaction.
ctstamp1	integer	Together with **ctstamp2**, forms an insertion stamp that specifies the order of the transaction in the queue.
ctstamp2	integer	Together with **ctstamp1**, forms an insertion stamp that specifies the order of the transaction in the queue.
ctcommittime	integer	Time when the transaction represented by this buffer was committed.
ctuserid	integer	Login ID of the user who committed this transaction.
ctfromid	integer	Server ID of the server that sent this transaction. Used only in hierarchical replication.

Related reference:
Columns of the Buffer Tables

Columns of the Buffer Tables

The buffer tables contain information about the buffers that form the transactions that are listed in the transaction tables.

The names of buffer tables end with **_buf**. All buffer tables contain the same columns and the same column definitions.

Column	Type	Description
cbflags	integer	Internal flags for this buffer.
cbsize	integer	Size of this buffer in bytes.
cbkeyserverid	integer	Server ID of the database server where this data originated. This server ID is the group ID from the sqlhosts file.
cbkeyid	integer	Login ID of the user who originated the transaction that is represented by this buffer.
cbkeypos	integer	Log position on the source server for the transaction that is represented by this buffer.
cbkeysequence	integer	Sequence number for this buffer within the transaction.
cbreplid	integer	Replicate identifier for the data in this buffer.
cbcommittime	integer	Time when the transaction represented by this buffer was committed.

The following columns combine to form the primary key for this table: **cbkeyserverid**, **cbkeyid**, **cbkeypos**, **cbkeysequence**.

The columns **cbkeyserverid**, **cbkeyid**, and **cbkeypos** form a foreign key that points to the transaction in the **_txn** table that the buffer represents.

Related reference:
Columns of the Transaction Tables

Replication Examples

This appendix contains simple examples of replication using the command-line utility (CLU).

The cdr utility documents the CLU.

- Replication Example Environment
  To run the replication examples in this publication, you must set up IBM® Informix® database servers. Each database server must be in a database server group.
- Primary-Target Example
  This is a simple example of *primary-target* replication.
- Update-Anywhere Example
  This example builds on the primary-target example and creates a simple *update-anywhere* replication.
- Hierarchy Example
  This example adds a replication tree to the fully-connected environment of the **usa**, **italy**, and **japan** replication servers.

Related tasks:
Preparing the Network Environment

Replication Example Environment

To run the replication examples in this publication, you must set up IBM® Informix® database servers. Each database server must be in a database server group.

The replication environment for the examples consists of:

- Three computers (**s1**, **s2**, and **s3**) that host the database servers **usa**, **italy**, and **japan**. Each computer has active network connections to the other two computers.
- The database servers **usa**, **italy**, and **japan** are members of the database server groups **g_usa**, **g_italy**, and **g_japan**.

The sqlhosts file for each database server must contain the following connectivity information.

```
g_usa     group     -      -          i=1
usa       ontlitcp  s1     techpubs1  g=g_usa
g_italy   group     -      -          i=8
italy     ontlitcp  s2     techpubs2  g=g_ital
g_japan   group     -      -          i=6
japan     ontlitcp  s3     techpubs6  g=g_japan
```

You must create an sbspace for the row data and set the CDR_QDATA_SBSPACE parameter to the location of that sbspace. For more information, see Setting Up Send and Receive Queue Spool Areas and CDR_QDATA_SBSPACE Configuration Parameter.

All commands in this example, except for creation of the sample databases on **italy** and **japan**, are issued from the computer **s1**.

The databases for the examples are identical to **stores_demo** databases with logging, as follows:

- Create a database named **stores** on the **usa** database server:

  ```
  s1> dbaccessdemo -log stores
  ```

- Create a database named **stores** on the **italy** database server:

  ```
  s1> rlogin s2
  s2> dbaccessdemo -log stores
  ```

- Create a database named **stores** on the **japan** database server:

  ```
  s1> rlogin s3
  s3> dbaccessdemo -log stores
  ```

For information about preparing data for replication, see Data Preparation Example.

Primary-Target Example

This is a simple example of *primary-target* replication.

In primary-target replication, only changes to the primary table are replicated to the other tables in the replicate. Changes to the secondary tables are not replicated.

In this example, define the **g_usa** and **g_italy** database server groups as Enterprise Replication servers and create a replicate, **repl1**.

To define the database server groups and create the replicate

1. Create and populate the database that defines the **usa** database server as a replication server:

   ```
   cdr define server --init g_usa
   ```

   Before replicating data, you must define the database servers as *replication servers*. A replication server is a database server that has an extra database that holds replication information.

   The **--init** option specifies that this server is a new replication server. When you define a replication server, you *must* use the name of the database server group (**g_usa**) rather than the database server name.

2. Display the replication server that you defined to verify that the definition succeeded:

   ```
   cdr list server
   ```

   The command returns the following information:

   ```
   SERVER     ID STATE     STATUS     QUEUE    CONNECTION CHANGED
   ------------------------------------------------------------
   g_usa       1 Active    Local      0
   ```

3. Define the second database server, **italy**, as a replication server:

   ```
   cdr define server --connect=italy --init \
   --sync=g_usa g_italy
   ```

   The **--connect** option allows you to define **italy** (on the **s2** computer) while working at the **s1** (**usa**) computer. The **--sync** option instructs the command to use the already-defined replication server (**g_usa**) as a pattern for the new definition. The **--sync** option also links the two replication servers into a replication environment. Tip: In all options except the **--connect** option, Enterprise Replication uses the name of the database server group to which a database server belongs, instead of the name of the database server itself.
4. Verify that the second definition succeeded:

   ```
   cdr list server
   ```

The command returns the following information:

```
SERVER    ID STATE    STATUS    QUEUE   CONNECTION CHANGED
-------------------------------------------------------------
g_italy    8 Active   Connected 0       JUN 14 14:38:44 2000
g_usa      1 Active   Local     0
```

5. Define the replicate **repl1**:

```
cdr define replicate --conflict=ignore repl1 \
"P stores@g_usa:informix.manufact" \
"select * from manufact" \
"R stores@g_italy:informix.manufact" \
"select * from manufact"
```

These lines are all one command. The backslashes (\) at the end of the lines indicate that the command continues on the next line.

This step specifies that conflicts should be ignored and describes two participants, **usa** and **italy**, in the replicate:

- The `P` indicates that in this replicate **usa** is a primary server. That is, if any data in the selected columns changes, that changed data should be sent to the secondary servers.
- The `R` indicates that in this replicate **italy** is a secondary server (receive-only). The specified table and columns receive information that is sent from the primary server. Changes to those columns on **italy** are *not* replicated.

6. Display the replicate that you defined, so that you can verify that the definition succeeded:

```
cdr list replicate
```

The command returns the following information:

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------------
REPLICATE:     repl1
STATE:         Inactive
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   g_usa:informix.manufact
               g_italy:informix.manufact
```

Step 5 *defines* a replicate but does not make the replicate active. The output of step 6 shows that the STATE of the replicate is INACTIVE.

7. Make the replicate active:

```
 cdr start repl1
```

8. Display the replicate so that you can verify that the STATE has changed to ACTIVE:

```
cdr list replicate
```

The command returns the following information:

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------------
REPLICATE:     repl1
STATE:         Active
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   g_usa:informix.manufact
               g_italy:informix.manufact
```

If any changes are made to the **manufact** table, the changes will be replicated to the other participants in the replicate.

Now you can modify the **manufact** table on the **usa** database server and see the change reflected in the **manufact** table on the **italy** database server.

To cause a replication

1. Use DB-Access to insert a value into the **manufact** table on **usa**:

```
INSERT INTO stores@usa:manufact \
VALUES ('AWN','Allwyn','8');
```

2. Observe the changes on **usa** and on **italy**:

```
SELECT * from stores@usa:manufact
SELECT * from stores@italy:manufact
```

In **repl1**, **usa** is the primary database server and **italy** is the target. Changes made to the **manufact** table on **italy** do not replicate to **usa**.

To not cause a replication

1. Use DB-Access to insert a value into the **manufact** table on **italy**:

```
INSERT INTO stores@italy:manufact \
VALUES ('ZZZ','Zip','9');
```

2. Verify that the change occurred on **italy** but did not replicate to the **manufact** table on **usa**:

```
SELECT * from stores@usa:manufact
SELECT * from stores@italy:manufact
```

Update-Anywhere Example

This example builds on the primary-target example and creates a simple *update-anywhere* replication.

In update-anywhere replication, changes to *any* table in the replicate are replicated to *all* other tables in the replicate. In this example, any change to the **stock** table of the **stores** database on any database server in the replicate will be replicated to the **stock** table on the other database servers.

In this example, define the **repl2** replicate.

To prepare for update-anywhere replication

1. Define the replicate, **repl2**:

   ```
   cdr define replicate --conflict=ignore repl2 \
   "stores@g_usa:informix.stock" "select * from stock" \
   "stores@g_italy:informix.stock" "select * from stock"
   ```

   These lines are all one command. The backslashes (\) at the end of the lines indicate that the command continues on the next line.

   This step specifies that conflicts should be ignored and describes two participants, **usa** and **italy** (including the table and the columns to replicate) in the replicate.

   Because neither `P` (primary) nor `R` (receive-only) is specified, the replicate is defined as update-anywhere. If any data in the selected columns changes, on either participant, that changed data should be sent to the other participants in the replicate.

2. Display all the replicates so that you can verify that your definition of **repl2** succeeded:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   -------------------------------------------------------------
   REPLICATE:     repl1
   STATE:         Active
   CONFLICT:      Ignore
   FREQUENCY:     immediate
   QUEUE SIZE:    0
   PARTICIPANT:   g_usa:informix.manufact
                  g_italy:informix.manufact

   REPLICATE:     repl2
   STATE:         Inactive
   CONFLICT:      Ignore
   FREQUENCY:     immediate
   QUEUE SIZE:    0
   PARTICIPANT:   g_usa:informix.stock
                  g_italy:informix.manufact
   ```

   Although this output shows that **repl2** exists, it does not show the *participant modifiers* (the SELECT statements) for **repl2**.

3. Display the participant modifiers for **repl2**:

   ```
   cdr list replicate repl2
   ```

   This command returns the following information:

   ```
   REPLICATE  TABLE                        SELECT
   -------------------------------------------------------------
   repl2      stores@g_usa:informix.stock   select * from stock
   repl2      stores@g_italy:informix.stock select * from stock
   ```

4. Add the **japan** database server to the replication system already defined in the previous example:

   ```
   cdr define server --connect=japan --init \
   --sync=g_usa g_japan
   ```

   You can use either **g_usa** or **g_italy** in the **--sync** option.

   Enterprise Replication maintains identical information on all servers that participate in the replication system. Therefore, when you add the **japan** database server, information about that server is propagated to all previously-defined replication servers (**usa** and **italy**).

5. Display the replication servers so that you can verify that the definition succeeded:

   ```
   cdr list server
   ```

   The command returns the following information:

   ```
   SERVER    ID STATE    STATUS    QUEUE   CONNECTION CHANGED
   -------------------------------------------------------------
   g_italy    8 Active   Connected 0       JUN 14 14:38:44 2000
   g_japan    6 Active   Connected 0       JUN 14 14:38:44 2000
   g_usa      1 Active   Local     0
   ```

6. Add the participant and participant modifier to **repl2**:

   ```
   cdr change replicate --add repl2 \
   "stores@g_japan:informix.stock" "select * from stock"
   ```

7. Display detailed information about **repl2** after adding the participant in step 6:

   ```
   cdr list replicate repl2
   ```

The command returns the following information:

```
REPLICATE  TABLE                         SELECT
-----------------------------------------------------------
repl2      stores@g_usa:informix.stock   select * from stock
repl2      stores@g_italy:informix.stock select * from stock
repl2      stores@g_japan:informix.stock select * from stock
```

8. Make the replicate active:

```
 cdr start repl2
```

9. Display a list of replicates so that you can verify that the STATE of **repl2** has changed to ACTIVE:

```
cdr list replicate
```

The command returns the following information:

```
CURRENTLY DEFINED REPLICATES
-----------------------------------------------------
REPLICATE:      repl1
STATE:          Active
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    g_usa:informix.manufact
                g_italy:informix.manufact

REPLICATE:      repl2
STATE:          Active
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    g_usa:informix.stock
                g_italy:informix.manufact
                g_japan:informix.manufact
```

Now you can modify the **stock** table on one database server and see the change reflected on the other database servers.

To cause a replication

1. Use DB-Access to insert a line into the **stock** table on **usa**:

```
INSERT INTO stores@usa:stock VALUES (401, "PRC", "ski boots", 200.00,
                "pair", "pair");
```

2. Observe the change on the **italy** and **japan** database servers:

```
SELECT * from stores@italy:stock;
SELECT * from stores@japan:stock;
```

To update the stock table on the japan database server

1. Use DB-Access to change a value in the **stock** table on **japan**:

```
UPDATE stores@japan:stock SET unit_price = 190.00
WHERE stock_num = 401;
```

2. Verify that the change has replicated to the **stock** table on **usa** and on **italy**:

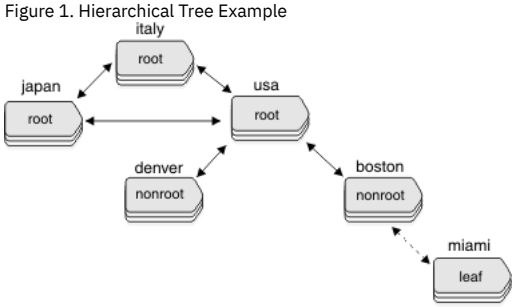
```
SELECT * from stores@usa:stock WHERE stock_num = 401;
SELECT * from stores@italy:stock WHERE stock_num = 401;
```


Hierarchy Example

This example adds a replication tree to the fully-connected environment of the **usa**, **italy**, and **japan** replication servers.

The nonroot servers **boston** and **denver** are children of **usa**. (The leaf server **miami** is a child of **boston**.) Figure 1 shows the replication hierarchy.

Figure 1. Hierarchical Tree Example



To try this example, you need to prepare three additional database servers: **boston**, **denver**, and **miami**. To prepare the database servers, use the techniques described in Replication Example Environment.

The following example defines a replication hierarchy that includes **denver**, **boston**, and **miami** and, whose root is **usa**.

To define a hierarchy

1. Add **boston** to the replication hierarchy as a nonroot server attached to the root server **usa**:

   ```
   cdr define server --connect=boston --nonroot --init \
   --sync g_usa g_boston
   ```

   The backslash (\) indicates that the command continues on the next line.

2. Add **denver** to the replication hierarchy as a nonroot server attached to the root server **usa**:

   ```
   cdr define server -c denver -I -N --ats=/ix/myats \
   -S g_usa g_denver
   ```

   This command uses short forms for the **connect**, **init**, and **sync** options. (For information about the short forms, refer to Option Abbreviations.) The command also specifies a directory for collecting information about failed replication transactions, **/ix/myats**.

3. List the replication servers as seen by the **usa** replication server:

   ```
   cdr list server
   ```

   The root server **usa** is fully connected to all the other root servers. Therefore **usa** knows the connection status of all other root servers and of its two child servers, **denver** and **boston**. The command returns the following information:

   ```
   SERVER     ID    STATE    STATUS    QUEUE    CONNECTION CHANGED
   -------------------------------------------------------------
   g_boston    3    Active   Connected     0    Aug 19 14:20:03 2000
   g_denver   27    Active   Connected     0    Aug 19 14:20:03 2000
   g_italy     8    Active   Connected     0    Aug 19 14:20:03 2000
   g_japan     6    Active   Connected     0    Aug 19 14:20:03 2000
   g_usa       1    Active   Local         0
   ```

4. List the replication servers as seen by the **denver** replication server:

   ```
   cdr list server --connect=denver
   ```

   The nonroot server **denver** has a complete global catalog of replication information, so it knows all the other servers in its replication system. However, **denver** knows the connection status only of itself and its parent, **usa**.

   The command returns the following information:

   ```
   SERVER     ID    STATE    STATUS    QUEUE    CONNECTION CHANGED
   -------------------------------------------------------------
   g_boston    3    Active                0
   g_denver   27    Active   Local        0
   g_italy     8    Active                0
   g_japan     6    Active                0
   g_usa       1    Active   Connected    0    Aug 19 14:20:03 2000
   ```

5. Define **miami** as a leaf server whose parent is **boston**:

   ```
   cdr define server -c miami -I --leaf -S g_boston g_miami
   ```

6. List the replication servers as seen by **miami**:

   ```
   cdr list server -c miami
   ```

   As a leaf replication server, **miami** has a limited catalog of replication information. It knows only about itself and its parent.

   The command returns the following information:

   ```
   SERVER     ID    STATE    STATUS    QUEUE    CONNECTION CHANGED
   -------------------------------------------------------------
   g_boston    3    Active   Connected     0    Aug19 14:35:17 2000
   g_miami     4    Active   Local         0
   ```

7. List details about the **usa** replication server:

   ```
   cdr list server g_usa
   ```

   The server is a *hub*; that is, it forwards replication information to and from other servers. It uses the default values for idle timeout, send queue, receive queue, and ATS directory.

   The command returns the following information:

   ```
   NAME        ID   ATTRIBUTES
   ---------------------------------------------------------------
   g_usa       1    timeout=15 hub sendq=rootdbs recvq=rootdbs atsdir=/tmp
   ```


Data sync warning and error messages

Data sync warning and error messages describe problems with replicated transactions.

You can suppress these messages from being written to the ATS and RIS files. You cannot suppress code 0, DSROWCOMMITTED, which indicates that the row was committed, or code 1, DSEROW, which indicates that an error occurred.

To specify which warnings and errors to suppress, use the CDR_SUPPRESS_ATSRISWARN configuration parameter. For more information, see CDR_SUPPRESS_ATSRISWARN Configuration Parameter.

Table 1. Data sync warning and error messages

Warning or Error Code	Number	Description
DSEReplInsertOrder	2	Warning: Insert exception, row already exists in target table, converted to update
DSEReplUpdateOrder	3	Warning: Update exception, row does not exist in target table, converted to insert
DSEReplDeleteOrder	4	Warning: Delete exception, row does not exist in target table, saved in delete table
DSEReplInsert	5	Error: Insert aborted, row already exists in target table
DSEReplUpdate	6	Error: Update aborted, row does not exist in target table
DSEReplDelete	7	Error: Delete aborted, row does not exist in target table
DSERowLength	8	Error: Length of replicated row is greater than row size in target table
DSEDbopType	9	Error: Unknown db operation
DSENoServerTimeCol	10	Error: Missing cdrserver and/or cdrtime columns in target table
DSEConflictRule	13	Error: Unknown conflict resolution rule defined
DSELostConflictRes	14	Error: Failed conflict resolution rule
DSENoServerName	15	Error: Global catalog cannot translate replicate server id to name
DSEColMap	16	Error: Unable to remap columns selected for replication
DSEColUncomp	17	Error: Invalid char/length in VARCHAR column
DSESPRetTypeOp	18	Error: Invalid data type or unknown operation returned by stored procedure
DSESPAbortRow	19	Error: Row aborted by stored procedure
DSESPSelCols	20	Error: Number of columns returned by stored procedure not equal to the number of columns in select statement
DSESPColTypeLen	21	Error: Invalid data type or length for selected columns returned by stored procedure
DSESPError	22	Error: Error returned by user's stored procedure
DSESPInternal	23	Error: Internal error (buffer too small for stored procedure arguments
DSENoMatchKeyInsert	24	Error: No matching key delete row for key insert
DSESql	25	Error: SQL error encountered
DSEIsam	26	Error: ISAM error encountered
DSELocalDReExist	27	Warning: Local delete row has been reinserted on local server
DSELocalDOddState	28	Warning: Unable to determine if the local delete row should be updated to the delete table
DSELocalDInDelTab	29	Warning: Row already exists in delete table for the given local delete row
DSEBlobOrder	30	Warning: Row failed conflict resolution rule but one or more blob columns were accepted
DSEBlobSetToNull	31	Warning: One or more blob columns were set to NULL because data could not be sent
DSEBlobKeepLocal	32	Warning: One or more blob columns were not changed because data could not be sent
DSEBlobInvalidFlag	33	Error: Invalid user action defined for blob columns
DSEBlobAbortRow	34	Error: Row aborted by user's stored procedure due to unsent blobs
DSESPBlobRetOp	35	Error: Invalid action returned by user's stored procedure on blob columns
DSEReplDel	36	
DSENoUDTHeader	37	
DSENoUDTTrailer	38	
DSEStreamHandle	39	
DSEAttachUDREnv	40	
DSECdrreceiveSetup	41	
DSECdrreceiveCall	42	
DSECdrreceiveRetCode	43	cdrreceive returned error
DSECdrreceiveRetGarbage	44	cdrreceive returned garbage
DSEStream	45	Error reading from stream
DSEStreamAborted	46	Stream aborted by sender
DSEValStore	47	
DSECdrreceiveRetType	48	cdrreceive returned wrong type
DSEStreamOptType	49	Unrecognized stream option
DSEStreamOptLen	50	Stream option has bad length
DSEStreamOptBitmap	51	Error in changed col bitmap
DSEUnStreamColl	52	Error while unstreaming collection

Warning or Error Code	Number	Description
DSEUnStreamRowType	53	Error while unstreaming rowtype
DSEStreamFormat	54	Unexpected or invalid data in stream
DSEStack	55	Out of stack space
DSEInternal	56	Generic internal problem
DSESmartBlobCreate	57	Error creating sblob
DSESmartBlobWrite	58	Error writing sblob
DSEStreamColConv	59	Error converting column data from the master dictionary formats to the local dictionary format
DSE2UTF8CodeSetConvErr	63	Error converting data from local database code set to UTF-8
DSEFromUTF8CodeSetConvErr	64	Error converting data from UTF-8 to local database code set.
DSE2UTF8CodeSetConvWarn	65	One or more characters were substituted during conversion from the local database code set to UTF-8.
DSEFromUTF8CodeSetConvWarn	66	One or more characters were substituted during conversion from UTF-8 to the local database code set.
DSETSSetup	67	Failed to setup environment for processing time series elements.
DSETSDelOp	68	Failed to apply time series delete statement. Statements include DelClip(), DelRange(), and DelTrim() operations.
DSETSElem	69	Failed to apply time series element.
DSEOpenTSCon	133	Failed to open time series container.