IBM INFORMIX V.14.10.XC4 - WIRE LISTENER AND JAVA

v.1

Agenda

- Wire listener enhancements
- JDBC enhancements
- J/Foundation enhancements
- The CDC Java transaction engine

Wire listener enhancements

Wire listener enhancements

- The wire listener was introduced in Informix v.12.10.xC3

- The wire listener is a Java pre-processor that manages the communication interface between non-SQL applications and the instance
  - It is the communication interface for the MongoDB API, the REST API and the MQTT API

- While the wire listener can use any network-based instance connection, the best practice is to create one or more dedicated instance connections for non-SQL applications
  - As a result, you need to add a network-based `DBSERVERALIAS` to `$ONCONFIG` and `$SQLHOSTS`

NoSQL interface connection

```
##########################################

SERVERNUM 1
DBSERVERNAME inst_1
DBSERVERALIASES mon_inst_1
FULL_DISK_INIT  0

##########################################
# Network Configuration Parameters
##########################################
##########################################

NETTYPE soctcp,2,50,NET
LISTEN_TIMEOUT 60
MAX_INCOMPLETE_CONNECTIONS 1024
```

Since `DBSERVER` and `DBSERVERALIAS` are using the same communication protocol, only one `NETTYPE` needs to be configured

Wire listener enhancements

- You don't need to make any special changes to `$SQLHOSTS` for non-SQL connectivity
  - They're just an instance alias connection
  - One caveat – if the non-SQL app is co-resident on the same server as the Informix instance it's connecting to, there is greater stability when the instance alias definition uses local loopback rather than the resolvable server name
    - An explicit server name can be set with the `listener.hostName` parameter in the properties file

```
inst_1          onsoctcp        ifmx-svr        inst1_tcp
mon_inst_1      onsoctcp        127.0.0.1       moninst1_tcp
```

- `/etc/services`

```
#### Modifications for Informix services
inst1_tcp       60000/tcp               ## for the main Informix instance
moninst1_tcp    60001/tcp               ## for the NoSQL interface
```

Wire listener enhancements

- Although an additional connection interface and its port are defined and enabled in the instance (using `DBSERVERALIAS` and `$SQLHOSTS` entries), non-SQL apps do not use this port to connect to the instance

- There is a connection properties file for the wire listener with a number of parameters including the listening port for inbound connections and the outbound port for connections to the instance
  - The outbound, instance connection url

```
#
url=jdbc:informix-sqli://localhost:60001/sysmaster:INFORMIXSERVER=mon_inst_1;USE
R=ifxjson;PASSWORD=in4mix12
```

  - The listener port number for application connections

```
## Description: The port number to listen on for incoming connections from clien
ts.
#
# Use of a port number below 1024 may require additional operating system privil
eges
# be granted to the user starting the JSON listener.
#
listener.port=27017
```

Wire listener enhancements

- Here is how non-SQL applications connect to an Informix instance
  - `$SQLHOSTS` and `/etc/services` information

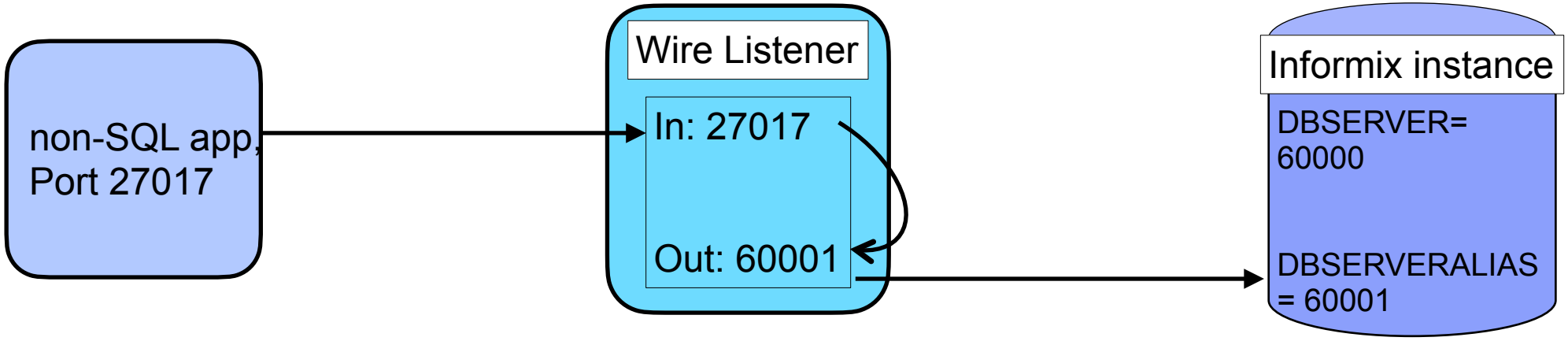
```
inst_1          onsoctcp        ifmx
mon_inst_1      onsoctcp        127.
```

```
#### Modifications for Informix services
inst1_tcp       60000/tcp               ## for the main Info
moninst1_tcp    60001/tcp               ## for the NoSQL int
```

  - The wire listener properties file

```
# be granted to the us
#
listener.port=27017
```

```
#
url=jdbc:informix-sqli://localhost:60001/sysmaster:INFORMIXSERVER=mon_inst_1;USE
R=ifxjson;PASSWORD=in4mix12
```



Wire listener enhancements

- The user ID used in the connection URL obviously must be a valid user on the system and have privileges in the databases

- In earlier Informix versions, when installing in “Typical” mode, the `ifxjson` user ID was automatically created as an ID to use for this functionality
  - The user ID was made part of the DBSA group because it executed administrative actions
  - The user ID was granted `Replication` privileges in the instance so it could execute sharding and other ER-related operations
  - This is no longer the case

- You do not have to use `ifxjson` as the user ID, any other valid ID may be used provided it has sufficient privileges granted to it
  - If you want to use, `ifxjson`, you’ll have to create it manually as shown on the following slides
    - Actually, this is the best practice for any ID used for the wire listener

Wire listener enhancements

- To manually create the `ifxjson` ID then grant instance privileges:
- The best practice is to configure the ID with a “no login” shell to prevent its use to log into the host system
  - This is important since the ID and password are in clear text in the connection properties file

1001	informix	informix	/home/infor...	/bin/bash
1002	ifxjson	ifxjson	/home/ifxjson	
1003	ihqadmin	ihqadmin	/home/ihqad...	/bin/bash

- Then using the Sysadmin API, grant the `Replication` privilege to this non-Informix user ID
  - The ability to grant specific privileges to users was added in Informix v.12.10

```
execute function [task | admin] (“grant admin”, “ifxjson”, “replication”)
```

Wire listener enhancements

- Okay, enough of the background information, what's new in Informix v.14.10.xC4?
- First, is current API level support
  - The wire listener now supports MongoDB API versions 3.2, 3.4, 3.6, 4.0, and 4.2
- Prior to xC4, the wire listener only supported API versions that were 5 years old or older
  - All of these went out-of-support several years ago
  - We now support up to, and including, the latest API version (4.2)
- While we now support the latest API versions, we do not have 100% full feature compatibility
  - We have implemented the most commonly used syntax
  - If you use a API command that is not supported, the wire listener will give you a good, rather than cryptic error message

Wire listener enhancements

- When using the wire listener for the MongoDB API, part of configuring its properties file is selecting which API version to use
  - While xC4 supports the latest API versions, for in-place upgrade compatibility reasons, the default API version is still 2.4
    - This will change in the next major release

```
## Key:         mongo.api.version
## Type:        string
## Default:     2.4
## Description: MongoDB API version that should define the listener responses.
#
# Specifies the MongoDB version that listener responses should be compatible wit
h. Options are "2.4", "2.6", "3.0", "3.2", "3.4", "3.6", "4.0", or "4.2"
#
# mongo.api.version=2.4
```

    - Obviously you can set this to what ever you need for your applications

Wire listener enhancements

- The second wire listener enhancement focuses on how REST and MQTT sessions authenticate with the instance

- Prior to xC4, the two options were either through PAM (password authentication module) or a MongoDB Java connection URL
  - The PAM had to use MongoDB style authentication as well
  - Neither of these worked well with these types of applications
    - The applications had to create users and store/hash passwords using MongoDB style operations
    - Why force these application developers to use foreign syntax and workflow in their applications?

Wire listener enhancements

- With Informix v.14.10.xC4 there are three options for REST or MQTT authentication
  - The MongoDB style process
  - A PAM module
  - Regular, defined Informix user IDs
    - Can be O/S IDs or mapped IDs

- This third option enables a simpler and easier connection process which matches SQL applications and interfaces

- The REST and MQTT applications still have to use the wire listener for the communication interface
  - The wire listener passes the UID and password to the instance through the JDBC connection process
  - The instance perceives the connection as though it was a “regular” JDBC application

Wire listener enhancements

- The last wire listener enhancement is improved performance of REST-based operations
- Tests indicate up to 25% better performance for queries
  - Some of this boost comes from improvements in the JDBC 4.50.JC4 driver
    - More on this in a moment
- The rest of performance increase comes from a modification in the wire listener properties file depending on how the listener is configured

Wire listener enhancements

- The `listener.type` parameter determines how a wire listener will communicate with applications
  - The choices are `mongo` (default), `rest`, and `mqtt`

```
## Key:         listener.type
## Type:        string(mongo,rest,mqtt)
## Default:     mongo
## Description: The type of listener to start.
#
# A mongo listener supports clients that use the Mongo driver protocol.
# A rest listener supports clients that use REST over HTTP.
# A mqtt listener support client that use the MQTT message protocol.
#
# listener.type=mongo
```

  - Depending on whether you select `rest` or `mongo`, the wire listener modifies the `response.documents.size.maximum` parameter automatically

Wire listener enhancements

- The `response.documents.size.maximum` parameter determines the amount of data that can be passed back to the application
  - The MongoDB API has a fixed limit of 1,048,578 bytes
    - If the result set is larger than that, it must be passed back in multiple packets, each processed separately by the application
  - The REST API does not have this limit
- If the listener type is set to `rest`, this parameter is automatically disabled by the listener so the full result set can be passed back resulting in faster application performance
  - You can over-ride this by entering a fixed value in this parameter but chances are you won't get as much of a performance improvement

JDBC enhancements

JDBC enhancements

- Part of the work released in the xC4 JDBC version is to lay the foundation for future enhancements supporting TimeSeries and other UDTs

- In JDBC 4.50.xC4 TypeMaps and Global TypeMaps are available
  - A TypeMap is how the UDTs in Informix are mapped to custom Java classes in the application for use
  - For example, in Informix there is the `bson` data type which is a binary type
    - In JDBC, there is a `bson` class which supports sending and receiving data within this data type
    - The application can use this class

- Prior to xC4, there were two hard coded TypeMaps in the JDBC driver and customers had to create their own TypeMaps for other data types
  - We offered support for custom TypeMaps in the driver for years

JDBC enhancements

- While this process of creating and supporting TypeMaps works with persistent connections, pooled connections or connections that drop and return often have trouble setting or changing their TypeMap
- JDBC 4.50.xC4 introduces a Global TypeMap building infrastructure
  - It includes all the supported Informix UDTs
  - Customers can add their own types
- When customers use this functionality and define their own TypeMaps, when a new JDBC connection is established, the driver will use the global Builder to automatically assign the UDTs to classes
  - This enables you to define a type once but use it in any map for any connection

JDBC enhancements

- As mentioned, the JDBC driver comes with 4 bundled UDTs, 2 are new types
  - `bson`
  - `json` — implemented as a wrapper around the `lvarchar` the instance sends
  - `binaryVar`
  - `binary18`

JDBC enhancements

- JDBC 4.50.xC4 also saw performance enhancements in several areas
  - Since JDBC is the foundation for almost all the Informix Java functionality (including IHQ), any improvements in it result in performance improvements throughout the engine

- Some of the optimizations and improvements in this driver include
  - Optimizing column lookups — the `ResultSet.getObject()` calls are significantly faster
  - Faster instantiation of `bson` objects within the driver
  - Encoding to and from `bson` (from/to `json`) is at least 5% faster if not more
  - Improvements of 8% or more in connecting a JDBC session to the instance
  - Inserts of `byte` or `text` data types are about 7% faster
  - The creation and use of prepared statements is about 4% faster

JDBC enhancements

- The final category of JDBC enhancements is new connection parameters
  - These are compatibility features for applications which must interact with Informix and other database servers or tools
  - `METADATA_UPPERCASE_VALUES` — sets the metadata results returned to uppercase
  - `AUTO_CASE_SCHEMA` — configures JDBC whether to “case” the schema in upper or lower case OR to return the schema as it was saved
  - `CURSOR_HOLDABILITY` — specifically for `mode ansii` databases, does the driver hold cursors open over a `commit`? The default is `off`.
    - This has always been supported within the driver with the `Connection.setHoldability()` property
      - However pre-compiled applications without access to the source code couldn’t use the feature
    - Now the functionality is available through the URL or datasource call

J/Foundation enhancements

J/Foundation enhancements

- J/Foundation didn't see as many improvements as the client drivers but it did see an expansion of the number of pre-built UDRs
- These UDRs leverage the functionality in the JDBC drivers so there are a lot of features and are fairly easy to use

- The new J/Foundation features include
  - A registry of functions so users can see what functionality is available to them
    - For example, you can create a function that calls the registry function

```
------------------------ my_test@inst_1 --------- Press CTRL-W for Help -------

CREATE FUNCTION genFunctionStatements() RETURNS LVARCHAR(30000) EXTERNAL NAME
'com.informix.judrs.JFoundation.generateCreateFunctionStatements()'
language java;
```

    - Then use it

```
----------------------- my_test@inst_1 ----

execute function genFunctionStatements()
```

J/Foundation enhancements

- The new J/Foundation features include
  - A registry of functions so users can see what functionality is available to them
  - There are functions which output the `grant` and `drop` commands as well
  - With these functions, you can recreate or add some or all of the builtin Java UDRs in another instance

```
(expression)  -- com.informix.judrs.JFoundation
              CREATE FUNCTION generateCreateFunctionStatements() RETURNS LVARCH
              AR EXTERNAL NAME 'com.informix.judrs.JFoundation.generateCreateFu
              nctionStatements()' LANGUAGE JAVA;
              CREATE FUNCTION generateCreateFunctionStatements(LVARCHAR) RETURN
              S LVARCHAR EXTERNAL NAME 'com.informix.judrs.JFoundation.generate
              CreateFunctionStatements(java.lang.String)' LANGUAGE JAVA;
              CREATE FUNCTION generateGrantStatements() RETURNS LVARCHAR EXTERN
              AL NAME 'com.informix.judrs.JFoundation.generateGrantStatements()
              ' LANGUAGE JAVA;
              CREATE FUNCTION generateDropFunctionStatements() RETURNS LVARCHAR
               EXTERNAL NAME 'com.informix.judrs.JFoundation.generateDropFuncti
              onStatements()' LANGUAGE JAVA;
              -- com.informix.judrs.JFoundationMemory
              CREATE FUNCTION getTotalMemory() RETURNS BIGINT EXTERNAL NAME 'co
              m.informix.judrs.JFoundationMemory.getTotalMemory()' LANGUAGE JAV
              A;
              CREATE FUNCTION getMaxMemory() RETURNS BIGINT EXTERNAL NAME 'com.
              informix.judrs.JFoundationMemory.getMaxMemory()' LANGUAGE JAVA;
              CREATE FUNCTION getFreeMemory() RETURNS BIGINT EXTERNAL NAME 'com
              .informix.judrs.JFoundationMemory.getFreeMemory()' LANGUAGE JAVA;

              -- com.informix.judrs.Explain
              CREATE FUNCTION getExplain(LVARCHAR) RETURNS LVARCHAR EXTERNAL NA
              ME 'com.informix.judrs.Explain.getExplain(java.lang.String)' LANG
              UAGE JAVA;
              -- com.informix.judrs.IfxStrings
              CREATE FUNCTION replaceAll(LVARCHAR, LVARCHAR, LVARCHAR) RETURNS
              LVARCHAR EXTERNAL NAME 'com.informix.judrs.IfxStrings.replaceAll(
              java.lang.String,java.lang.String,java.lang.String)' LANGUAGE JAV
              A;
              CREATE FUNCTION encodeBase64(BLOB) RETURNS LVARCHAR EXTERNAL NAME
               'com.informix.judrs.IfxStrings.encodeBase64(java.sql.Blob)' LANG
              UAGE JAVA;
              CREATE FUNCTION getUUID() RETURNS LVARCHAR EXTERNAL NAME 'com.inf
              ormix.judrs.IfxStrings.getUUID()' LANGUAGE JAVA;
              -- com.informix.judrs.LargeObjects
              CREATE FUNCTION lobSize(BLOB) RETURNS BIGINT EXTERNAL NAME 'com.i
              nformix.judrs.LargeObjects.lobSize(java.sql.Blob)' LANGUAGE JAVA;
```

J/Foundation enhancements

- The new J/Foundation features include
  - Access to query explain plans
    - Typically query explain plans are output on the server
  - There is a new function that executes a query, captures the query plan information from the file on the server, returns the file as a `lvarchar`, then deletes the server side file
  - For example
    - Build this function and use it

```
------------------------- stores@inst_1 ----------- Press CTRL-W for Help ------

CREATE FUNCTION getMyExplain(LVARCHAR) RETURNS LVARCHAR(30000) EXTERNAL NAME
'com.informix.judrs.Explain.getExplain(java.lang.String)' LANGUAGE JAVA;
```

```
execute function getMyExplain("select a.customer_num, order_num, order_date
from customer a, orders b
where a.customer_num = b.customer_num order by a.customer_num, b.order_num");
```

J/Foundation enhancements

- The new J/Foundation features include
  - Access to query explain plans

```
(expression)
              QUERY: (OPTIMIZATION TIMESTAMP: 07-02-2020 14:44:50)
              ------
              select a.customer_num, order_num, order_date from customer a, ord
              ers b where a.customer_num = b.customer_num order by a.customer_n
              um, b.order_num

              Estimated Cost: 11
              Estimated # of Rows Returned: 23
              Temporary Files Required For: Order By

                1) informix.b: SEQUENTIAL SCAN

                2) informix.a: INDEX PATH

                  (1) Index Name: informix. 100_1
                      Index Keys: customer_num   (Key-Only)  (Serial, fragments
              : ALL)
                      Lower Index Filter: informix.a.customer_num = informix.b.
              customer_num
              NESTED LOOP JOIN


              ----------
              Procedure: informix.getmyexplain
              Query statistics:
              -----------------

                Table map :
                ----------------------------
                Internal name     Table name
                ----------------------------
                t1                b
                t2                a

                type     table  rows_prod  est_rows  rows_scan  time       est_
              cost
                ------------------------------------------------------------
              ----
                scan     t1     23         23        23         00:00.00   3
```

J/Foundation enhancements

- The new J/Foundation features include
  - New utilities for LOBs
    - These are fairly easy to add and use because of JDBC functionality
    - These utilities can:
      - Return the size of a LOB
      - Concatenate a string with a CLOB to create a new CLOB
        - This can be inserted into the table or used elsewhere
      - Append data to a CLOB
      - Dump a CLOB's contents in 32k chunks
      - Create a CLOB from any string

J/Foundation enhancements

- The new J/Foundation features include
  - New utilities for LOBs
  - For example:

```
CREATE FUNCTION lobSize(BLOB) RETURNS BIGINT EXTERNAL NAME
'com.informix.judrs.LargeObjects.lobSize(java.sql.Blob)' LANGUAGE JAVA;

CREATE FUNCTION lobSize(CLOB) RETURNS BIGINT EXTERNAL NAME
'com.informix.judrs.LargeObjects.lobSize(java.sql.Clob)' LANGUAGE JAVA;

CREATE FUNCTION concat(CLOB, LVARCHAR) RETURNS CLOB EXTERNAL NAME
'com.informix.judrs.LargeObjects.concat(java.sql.Clob,java.lang.String)'
LANGUAGE JAVA;

CREATE FUNCTION append(CLOB, LVARCHAR) RETURNS CLOB EXTERNAL NAME
'com.informix.judrs.LargeObjects.append(java.sql.Clob,java.lang.String)'
LANGUAGE JAVA;

CREATE FUNCTION toString(CLOB) RETURNS LVARCHAR EXTERNAL NAME
'com.informix.judrs.LargeObjects.toString(java.sql.Clob)' LANGUAGE JAVA;
```

J/Foundation enhancements

- The new J/Foundation features include
  - New utilities for LOBs
  - Some examples

```
create table test_tab
(col1 int,
 col2 varchar(10),
 col3 clob)
```

```
insert into test_tab values (1, "first", toClob("Hello world!"));
insert into test_tab values (2, "second", toClob("Hello moon!"));
```

```
select col1, col2, lobsize(col3) from test_tab
```

```
col1 col2                    (expression)

   1 first                             12
   2 second                            11
```

J/Foundation enhancements

- The new J/Foundation features include
  - New utilities for LOBs
  - Some examples

```
execute function concat((select col3 from test_tab where col1 = 1), "more text")
```

```
(expression)
Hello world!more text
```

  - But this just created a new CLOB, it didn’t persist the data at all

J/Foundation enhancements

- The new J/Foundation features include
  - New utilities for LOBs
  - Some examples

```
execute function append((select col3 from test_tab where col1 = 2), " A lot more text!")
```

```
(expression)
Hello moon! A lot more text!
```

```
execute function toString((select col3 from test_tab where col1 = 1));
execute function toString((select col3 from test_tab where col1 = 2));
```

```
(expression)  Hello world!



(expression)  Hello moon! A lot more text!
```

J/Foundation enhancements

- These functions enable you to perform quick and dirty functional testing with CLOBs and some BLOBs without having to be a Java programmer or recompiling an application

The CDC Java transaction engine

CDC Java API enhancements

- With renewed support for Informix by the IBM CDC products, there is a new Java CDC transaction engine

- One of the “interesting” aspects of CDC is you get all sorts of records including those that really don’t need to be sent
  - `begin work` messages
  - DML operations (you need these!)
  - `commit work` messages
  - Rollbacks
  - Timeouts
  - And more

- Informix has superior technology with smart triggers or async post commit triggers
  - It just gives you the data changes, the stuff that matters, not all the other garbage

CDC Java API enhancements

- So Informix has built a transaction engine for CDC
  - It only provides the committed data
  - You can specify which record types you want
    - Just inserts, updates, deletes?
    - Include timeout messages?

Questions