Get started developing an integration solution in IBM Integration Bus v10

IBM Hybrid Integration ID team
Published on April 1, 2015 / Updated on March 22, 2016

IBM Integration Bus provides a flexible environment in which you can develop integration solutions to transform, enrich, route, and process your business messages and data. You can integrate client applications that use different protocols and message formats.

An integration solution is the container for the resources that you develop to process your business messages and data.

You design and develop integration solutions in IBM Integration Toolkit.

Step 1: Choose the implementation style of your integration solution
When you plan an integration solution, your first step is to decide the implementation style for your solution. In IBM Integration Bus, you can implement an integration solution as an application, an integration service, or a REST API.

An application allows you to connect your client applications whether they have or do not have defined interfaces. An application is a container for all the resources that are required to create an integration solution.

An integration service is a specialized application with a defined interface that acts as a container for a Web Services solution. External applications can call the operations by using a SOAP/HTTP binding, or from a JavaScript program by using a JavaScript API binding.

A REST API is a specialized application that can be used to expose RESTful web services. The REST API describes a set of resources, and a set of operations that can be called on those resources. External applications can call the operations in a REST API from any HTTP client, including client-side JavaScript code running in a web browser. (NEW in IBM Integration Bus v10)

Back to top

Step 2: Decide whether you want to use one or more libraries as part of your solution.
Libraries allow reuse of common assets between integration solutions. You can use them to group related code, data, or both.

Recommendation: Use libraries to group together resources of the same type or function, for reuse or ease of management.

1 comment on"Get started developing an integration solution in IBM Integration Bus v10"

Marimuthu Udayakumar April 01, 2015
Really nice demonstration to understand REST call in IIB.

Reply (Edit)

In IBM Integration Bus, there are two types of libraries that you can use:

- Shared library: A shared library typically contains reusable helper routines and resources such as ESQL modules, message definitions, message maps (graphical data maps), subflows, and Java™ utilities.

  > Recommendation: Use a shared library when you want to develop a set of resources that are common to multiple integration solutions, and you just want to update once the resources and make them available to all the applications where they are being used.

  - A shared library can be referenced by one or more applications.
  - Shared libraries are new in IBM Integration Bus v10.
  - Shared libraries are deployed once, as a separate entity. Each application, that references a shared library, can reference the resources in that deployed shared library. If that shared library is updated, the changes are immediately picked up by all referencing applications.
  - A shared library must be deployed before you can deploy an application that references it.
  - For more information, see Shared libraries

- Static library: A static library typically contains reusable helper routines and resources such as message flow, subflows, ESQL modules, message definitions, message maps (graphical data maps), and Java™ utilities.

  > Recommendation: Use a static library when you want to develop a set of resources that are common to multiple integration solutions, and you want to control when those changes are applied at runtime in each one of the consuming applications.

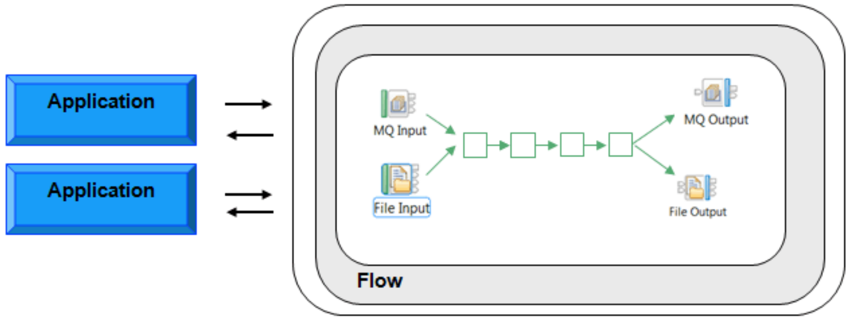
  - A static library can be referenced by one or more applications.
  - Static libraries were introduced in WebSphere Message Broker v8.
  - Each application that references a static library will deploy its own copy of the resources from the library.
  - For more information, see Static libraries

Back to top

Step 3: Plan the resources to implement your integration solution requirements

Depending on your choice in step 1, you need different resources to implement an integration solution:

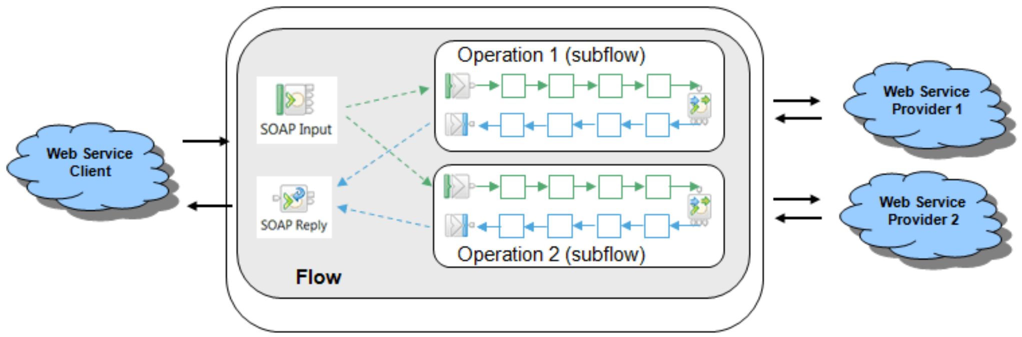
- To implement an application, you need one or more message flows and you might need to define message models for your data.
  - Message flows represent your processing steps.
  - Message models represent your input and output data to the solution.



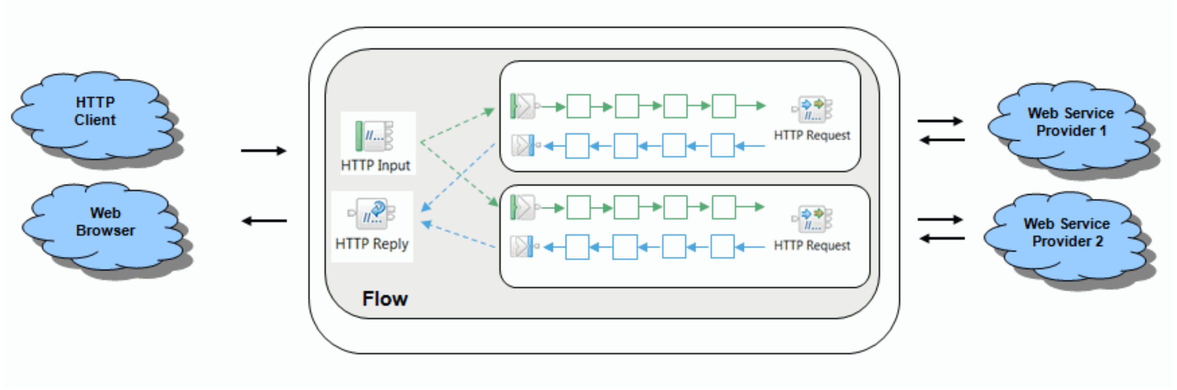
To implement an integration service, you need a WSDL file, or you can create one in IBM Integration Toolkit, that defines the interface, then you need to implement its operations.

The WSDL file provides the operations exposed by the service and the message models that represent the input and output data of each operation. (You also can create an integration service from scratch in IBM Integration Toolkit.)

Operations are implemented as subflows. Each subflow contains the processing logic associated with an operation.



To implement a REST API, you must first define the resources and operations that will be exposed by the API. Those definitions must be specified in a Swagger document, an open specification for defining REST APIs. Then, you need to implement a subflow per operation. Each subflow contains the processing logic associated with an operation.



IBM Integration Bus manages different sets of resources to integrate your applications and data:

Message model

A message model is used to represent some message formats. The message models are all based on World Wide Web Consortium (W3C) XML Schema 1.0 (XSD). IBM Integration Bus uses models that are based on XML Schema to describe the structure of different types of message formats, including message formats that are not XML. You can define XML schemas, DFDL models, and WebSphere Adapter schemas.
For more information, see The message model

Message flows

A message flow is a sequence of processing steps that run in an integration node when an input message is received. You can configure message flows to use one or both of the supported communication models, point-to-point and publish/subscribe. You can use message flow nodes, subflows, or a combination of both to define the processing steps.
For more information, see Message flows

Subflows

To reuse common application logic in multiple solutions, reduce development time, and increase maintainability of your message flows, you can use subflows. (Consider a subflow as analogous to a programming macro, or to inline code that is written once but used in many places.)
For more information, see Subflows

WSDL file

IBM Integration Bus supports the following WSDL styles:

- document-literal (style="document", use="literal")
- rpc-literal (style="rpc", use="literal")
- rpc-encoded (style="rpc", use="encoded")

Recommendation: Use the document-literal style for new integration services. It is the style that is least likely to cause interoperability problems.

For more information, see Creating a solution based on WSDL or XSD files

Swagger document

To build a REST API, you must provide a Swagger document. (A Swagger document is the REST API equivalent of a WSDL document for a SOAP-based web service.)
IBM Integration Bus supports version 2.0 of the Swagger specification.

The Swagger document specifies the list of resources that are available in the REST API and the operations that can be called on those resources. The Swagger document also specifies the list of parameters to an operation, including the name and type of the parameters, whether the parameters are required or optional, and information about acceptable values for those parameters. Additionally, the Swagger document can include JSON Schema that describes the structure of the request body that is sent to an operation in a REST API, and it describes the structure of any response bodies that are returned from an operation.

For more information, see Swagger

Back to top

Step 4: Plan the message models required by your integration solution
The next step is to identify the input and output data to your integration solution.
IBM Integration Bus supplies a range of parsers to parse and write message formats. Some message formats are self-defining and can be parsed without reference to a model. However, most message formats are not self-defining, and a parser must have access to a predefined model that describes the message, if it is to parse the message correctly.

A message model is used to represent a message format. The message models are all based on World Wide Web Consortium (W3C) XML Schema 1.0 (XSD). IBM Integration Bus uses models that are based on XML Schema to describe the structure of all kinds of message formats, including message formats that are not XML.
You can define XML schemas, and DFDL models.

XML Schema is an international standard that defines a language for describing the structure of XML documents.

Data Format Description Language 1.0 (DFDL) is an open standard modeling language from the Open Grid Forum (OGF) that builds upon the features of XML Schema 1.0 in order to model and validate all kinds of general text and binary data. It uses standard XSD model objects to describe the logical structure of data, together with DFDL annotations that describe the physical text or binary representation of data. IBM Integration Bus uses DFDL schema files to describe text and binary data, including industry standard formats.

For more information, see Get started with DFDL.

Recommendation: To make full use of the facilities that are offered by IBM Integration Bus, model your message formats.

Applications typically use a combination of message formats, including those message formats that are defined by the following structures or standards:

- Comma Separated Values (CSV)
- COBOL, C, PL/I, and other language data structures
- Industry standards such as SWIFT, X12 or HL7
- XML including SOAP
- JSON

Most message formats are not self-defining, and a parser must have access to a predefined model that describes the message, if it is to parse the message correctly. Even if your messages are self-defining, and do not require modeling, message modeling is necessary for IBM Integration Bus to understand data formats.

An example of a self-defining message format is XML. In XML, the message itself contains metadata in addition to data values, and it is this metadata that enables an XML parser to understand an XML message even if no model is available. Another example of a self-defining format is JSON.

Examples of messages that do not have a self-defining message format are CSV text messages, binary messages that originate from a COBOL program, and SWIFT formatted text messages. None of these message formats contain sufficient information to enable a parser to fully understand the message. In these cases, a model is required to describe them.

A parser is a program that interprets the physical bit stream of an incoming message, and creates an internal logical representation of the message in a tree structure. The parser also regenerates a bit stream for an outgoing message from the internal message tree representation.

In IBM Integration Bus, a message is described as a tree because messages are typically hierarchical in structure. The way in which the parser interprets the bit stream is unique to that parser; therefore, the logical message tree that is created from the bit stream varies from parser to parser.

A parser is called when the bit stream that represents an input message is converted to the internal form, also known as the logical tree. This invocation of the parser is known as parsing.

A parser is also called when a logical tree that represents an output message is converted into a bit stream. This action by the parser is known as writing. Typically, an output message is generated by an output node at the end of the message flow.

Each parser is suited to a particular class of messages, known as a message domain. The message domain identifies the parser that is used to parse and write instances of the message.

For example:

You can use XML domains to parse and write messages that conform to the W3C XML standard.

You can use the DFDL domain to parse and write general text and binary message formats, including industry standards.

This is a list of some of the message domains supported in IBM Integration Bus:

- DFDL parser and domain
- JSON parser and domain
- XMLNSC parser and domain
- SOAP parser and domain

For a list of available parsers, see Available parsers.

You can create a message model by using any of the following methods:

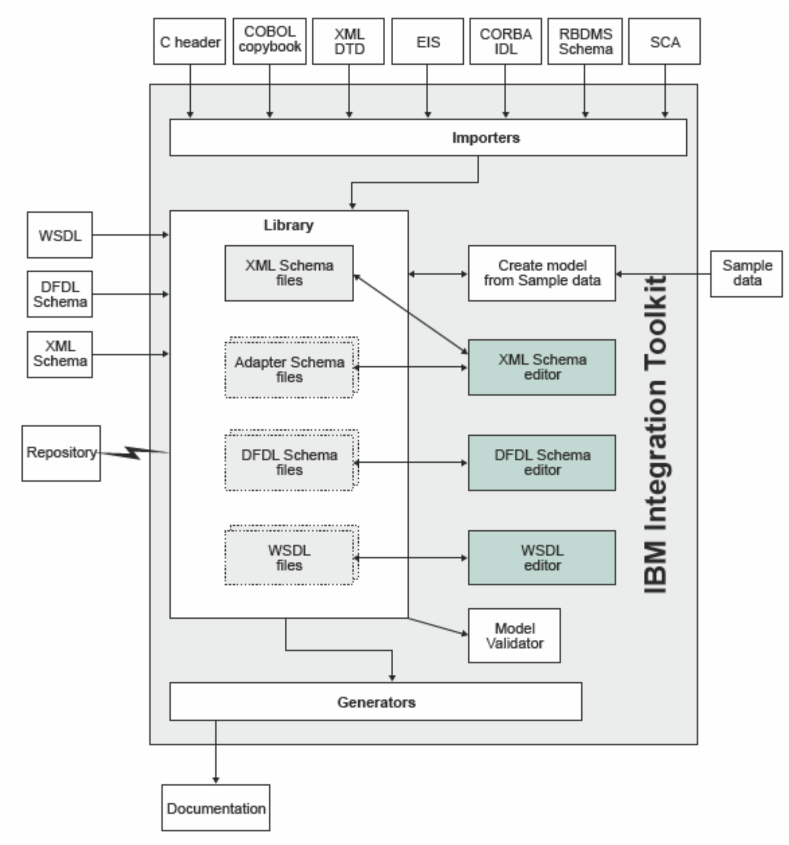
- Importing an application message format that is described by an XML Schema, XML DTD, C structure, COBOL structure, SCA import or export, or WSDL definition.
- By creating an empty message model file, then creating your message by using the XML editor, the WSDL editor, or the DFDL editor provided in the IBM Integration Toolkit.
- By using the Adapter Connection wizard to import EIS metadata.
- By creating a populated model file from example message data.

Message modeling has the following advantages:

- Runtime validation of messages. Without a message model, a parser cannot check whether input and output messages have the correct structure and data values.
- Enhanced parsing of XML messages. Although XML is self-defining, all data values are treated as strings if a message model is not used. If a message model is used, the parser is provided with the data type of data values, and can cast the data accordingly.
- Improved productivity when writing ESQL. When you are creating ESQL programs for IBM Integration Bus message flows, the ESQL editor can use message models to provide code completion assistance.
- Drag-and-drop operations on message maps. When you are creating message maps for IBM Integration Bus message flows, the Graphical Data Mapping editor uses the message model to populate its source and target views.
- Reuse of message models, in whole or in part, by creating additional messages that are based on existing messages.

Generation of documentation.

Provision of version control and access control for message models by storing them in a central repository.



Back to top

Step 5: Plan the resources to implement the processing logic in a message flow or subflow
To implement the processing logic in your integration solutions, you use message flow nodes and subflows.
A message flow node receives a message, performs a set of actions against the message, and optionally passes the original message, and none or more other messages, to the next node in the message flow. For more information, see Message flow nodes

A subflow is a sequence of processing steps that perform a task. You can use message flow nodes, subflows, or a combination of both to define the processing steps. It is designed to be embedded in a message flow or in another subflow. To connect your subflow to other nodes in the main flow, you can add Input and Output nodes to the subflow. For example, you might code some error processing in a subflow, or you can create a subflow to provide an audit trail (storing the entire message and writing a trace entry). For more information, see Designing a subflow

When you design your message flow steps, you must consider the following guidelines:

You must include at least one input node to receive messages. The input node that you select depends on the source of the input messages, and where in the flow you want to receive the messages. An input node checks when there is data for the message flow to process, reads that data from the transport or server, and presents that data to the rest of the flow for processing.

You must include processing nodes to transform a message from one format to another, or route a message along a particular path, or provide more complex options such as aggregation or filtering.

Some processing nodes allow you to use graphical data maps, Java code, .Net, an XSLT sheet, and others as part of your processing logic.

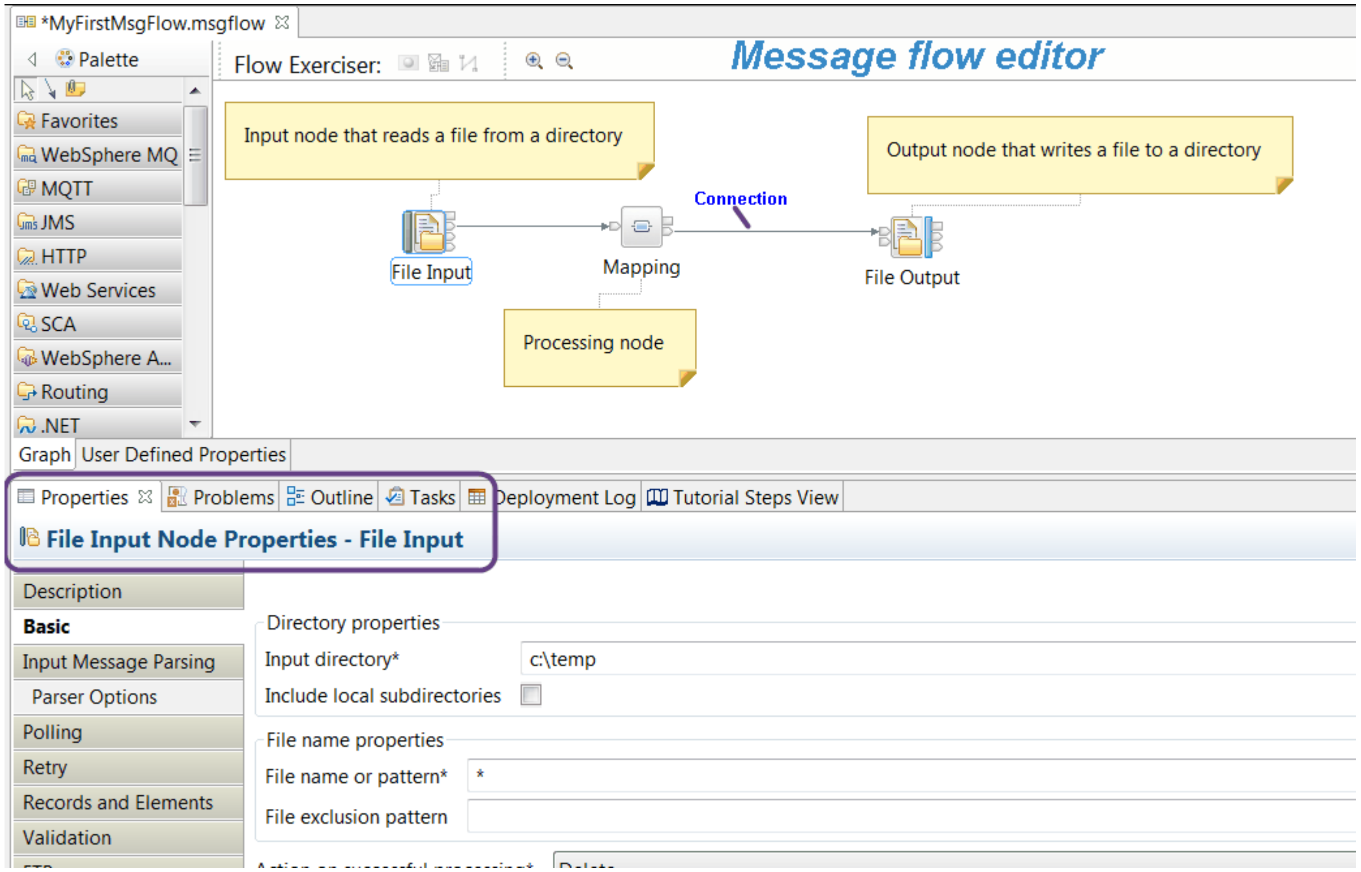
- Graphical Data Mapping – Get started with Graphical Data Mapping (GDM)
- Java
- ESQL
- .NET
- XSL

To send the data that is produced by the message flow to a target application, you can include one or more output nodes. The output node that you select depends on the transport across which the target application expects to receive those messages.

To make a request to an external system, and optionally use information from that system to enrich your message, you use a request node.

To reuse common application logic, use a subflow. A subflow is a group of processing nodes that perform a task.

Message flow nodes and subflows have a fixed number of input and output points known as terminals. In a message flow, you define connections (by using drag and drop) between the terminals of a node or subflow to define the routes that a message can take through a message flow. You can define multiple nodes and subflows within a message flow.



Directory properties

Input directory* c:\temp

Include local subdirectories ☐

File name properties

File name or pattern* *

File exclusion pattern

The message flow starts when a message is retrieved from an input device. The message flow ends when none or more output messages have been sent by one or more output nodes, and control returns back to the input node. The input node either commits or rolls back the transaction. Input and output nodes can be protocol-specific, to interact with particular systems such as web services.

There are different type of message flow
nodes:

Input nodes: Input nodes do not have input terminals. An input node checks when there is data for the message flow to process, reads that data from the transport or server, and presents that data to the rest of the flow for processing. For more information, see Input nodes

Output nodes: An output node sends a message to a target application. You can include one or more output nodes in a flow. For more information, see Output nodes

Processing nodes: These nodes typically transform a message from one format to another, or route a message along a particular path, or provide more complex options such as aggregation or filtering. For more information, see Processing nodes

Request nodes: These nodes are used to make a one-way request or a request-response call, in the middle of your flow, to an external system. For more information, see Request nodes

Nodes for making decisions: These nodes determine the order and flow of control in the message flow. You can use these nodes to decide how messages are processed by the flow. For more information, see Nodes for making decisions

Nodes for controlling time-sensitive operations: These nodes are used to run processes at specific times, or at fixed intervals, and take action if transactions are not completed within a defined time. For more information, see Nodes for controlling time-sensitive operations

WebSphere Adapter nodes: These nodes are used to communicate with Enterprise Information Systems (EIS), such as SAP, Siebel, JD Edwards, and PeopleSoft. For more information, see WebSphere Adapters nodes.

User-defined nodes: User-defined nodes are the main mechanism for extending the functions of IBM Integration Bus. A user-defined node must be written using the user-defined node API provided by IBM Integration Bus for both C and Java™ languages. The most common uses for a user-defined node are:

- Calling an external system for which IBM Integration Bus does not provide nodes
- Calling already defined program libraries that perform a transformation or calculation that is required in the design of a message flow
- Packaging a subflow

Before you consider constructing a user-defined node, make sure that no built-in node is available to perform the required actions. For more information, see User-defined nodes

For a complete list of built-in nodes, see Message flow nodes

You can configure a message flow node by setting or changing the values for its properties. Some nodes have mandatory properties, for which you must set a value. Other properties must have a value, but are assigned a default value that you can leave unchanged. The remaining properties are optional properties and no value is required. How you set the properties of the message flow nodes influences how the messages are processed by the message flow.

Back to top

Step 6: Plan the error handling strategy for your integration solution
In IBM Integration Bus, an integration node (run time where a message flow runs) provides basic error handling for all your flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows by using message flow node terminals to provide your own error handling.
All built-in nodes include error handling as part of their processing. If an error is detected within the node, the message is propagated to the failure terminal. What happens then depends on the structure of your message flow. You can use the basic error handling that is provided by the integration node, or you can enhance your flow by adding error processing nodes and flows to provide more comprehensive failure processing.

An integration service might encounter errors such as message failure or timeout. An error handler in an integration service is implemented as a subflow.
For more information, see Implementing an integration service error handler subflow.

An error handler in a REST API is implemented as a subflow. If you do not implement an error handler, default error handling behavior is used.
For more information, see Implementing an error handler in a REST API.

For more information, see Handling errors in message flows

Back to top

Other aspects to consider when you develop integration solutions

Message validation: Message flows are designed to transform and route messages that conform to certain rules. By default, parsers perform some validity checking on a message, but only to ensure the integrity of the parsing operation. However, you can validate a message more stringently against the message model contained in the message set by specifying validation options on certain nodes in your message flow.
You can use validation options to validate the following messages:

- Input messages that are received by an input node
- Output messages that are created, for example, by a Compute, Mapping, or JavaCompute node

These validation options can ensure the validity of data entering and leaving the message flow. The options provide you with some degree of control over the validation performed to:

- Maintain a balance between performance requirements and security requirements
- Validate at different stages of message flow completion; for example, on input of a message, before a message is propagated, or at any point in between
- Cope with messages that your message model does not fully describe

For more information, see Validating messages

Transactionality
IBM Integration Bus supports transactions, and every piece of data processed by a message flow has an associated transaction. A message flow transaction is started by the integration node when input data is received at an input node in the flow. It is committed when the flow has finished with that message, or rolled back if an error occurs.

A transaction describes a set of updates that are made by an application program, which must be managed together. The updates might be made to one or more systems. The updates made by the program are controlled by the environment in which the program executes, and either all are completed, or none. This property of a transaction is known as consistency: transactions might have other properties of atomicity, isolation, and durability.

If the flow contains more than one input node, one transaction is started for each input node when it receives input data. A transaction is started for every type of input node, including user-defined input nodes.

The nodes that you include in your message flow provide specific processing of a message, according to the defined function of each node. The processing that they do includes internal work, some of which you can influence by configuring the node properties. Some nodes perform additional tasks that might affect systems that are external to the message flow or the integration node.

If an external system, such as a database, supports the concept of commit and rollback, and it can take part in an integration node transaction, you can configure the node so that the work it does is included in the flow transaction. Depending on the node, you can also specify if the work done in an external system that supports transactions is committed immediately, or when the message flow transaction completes.

Many of the resources with which your message flows can interact are controlled by resource managers that can participate in coordinated transactions; for example, databases, WebSphere® MQ messages and queues, and JMS messages. Other resource managers do not provide transactional support; for example, the HTTP protocol and file systems.

For more information, see Message flow transactions

Security
An important aspect of securing an enterprise system is the ability to protect the information that is passed from third parties. In addition to securing the transport, you can secure individual messages based on their identity. IBM Integration Bus provides a security manager to control access to individual messages in a message flow, by using the identity of the message.
You can configure an integration node for processing end-to-end an identity that is carried in a message through a message flow by using

a security profile. By creating a security profile, you can configure security for a message flow to control access based on the identity that is associated with the message and provides a security mechanism that is independent of both the transport type and message format.
For more information, see Message flow security overview

Operational policies: You can use operational policies to control the behavior of message flows, and the nodes within message flows, at run time without the need to redeploy your resources.
An operational policy enables you to define a common approach to controlling particular node properties, such as connection credentials, as well as certain aspects of message flow behavior, including flow rate. Operational policies provide a shared and managed definition that you can reuse, and they enable the following user capabilities:

- For developers, operational policies offer the ability to reuse configuration information in multiple places.
- For administrators, operational policies offer the ability to define key configuration data for each environment.
- For operators, operational policies offer the ability to monitor and dynamically modify configuration data.

For more information, see Operational policies

Back to top