Global Knowledge®

IBM Integration Bus V10 Application Development I

Student Exercises

WM666G, ERC: 1.0
3614, Version 001
wm6661xstud

Global Knowledge®

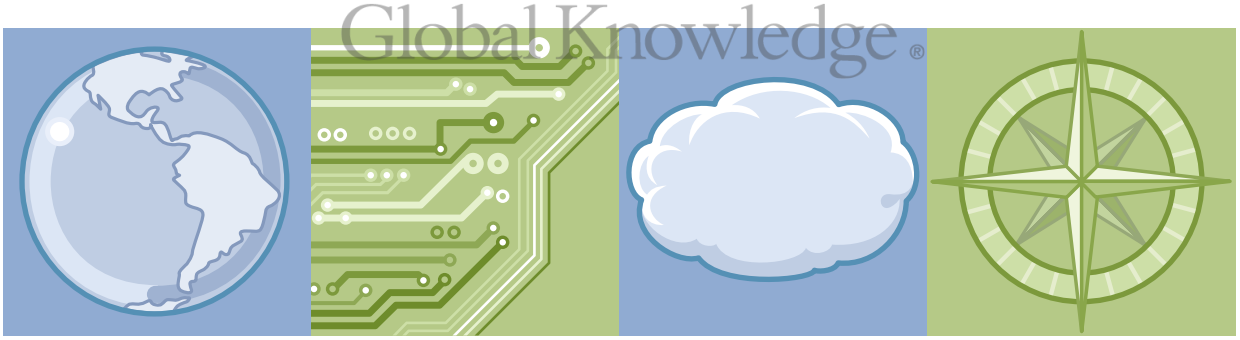
Global Knowledge®

IBM Training

Student Exercises

IBM Integration Bus V10 Application Development I

Course code WM666 / ZM666 ERC 1.0



IBM Systems Middleware

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

CICS®	DataPower®	DB™
DB2®	developerWorks®	Express®
IMS™	PartnerWorld®	Redbooks®
Tivoli®	Watson™	WebSphere®
z/OS®		

Intel and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

August 2015 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

CICS®	DataPower®	DB™
DB2®	developerWorks®	Express®
IMS™	PartnerWorld®	Redbooks®
Tivoli®	Watson™	WebSphere®
z/OS®		

Intel and Intel Core are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware and the VMware "boxes" logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the "Marks") of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

Exercises description

This course includes the following exercises:

- Exercise 1: Importing and testing a message flow
- Exercise 2: Creating a message flow application
- Exercise 3: Connecting to IBM MQ
- Exercise 4: Adding flow control to a message flow application
- Exercise 5: Creating a DFDL model
- Exercise 6: Processing file data
- Exercise 7: Using problem determination tools
- Exercise 8: Implementing explicit error handling
- Exercise 9: Referencing a database in a map
- Exercise 10: Transforming data by using the Compute and JavaCompute nodes
- Exercise 11: Creating a runtime-aware message flow

In the exercise instructions, you can check off the line before each step as you complete it to track your progress.

Most exercises include required sections that should always be completed. It might be necessary to complete these sections before you can start later exercises. Some exercises might also include optional sections that you might want to complete if you have sufficient time and want an extra challenge.

Important

The exercise **Introduction** contains important information for understanding the purpose and procedures in the exercise. Read the exercise **Introduction** before doing the exercise.

Logging on to the course image

Log on the course image with the following credentials:

User name: **`iibadmin`**

Password: **`web1sphere`**

How to use the course exercise instructions

Each exercise is divided into sections with a series of steps and substeps. The step represents an action. If required, the substeps provide guidance on completing the action.

Important

The steps that are listed here are just an example. Do not do them.

For example:

__ 1. Create a user account that named `testuser`.

__ a. Right-click **My Computer** and click **Manage** from the menu.

__ b. Expand **Local Users and Groups**.

In this example, the creation of a new user account is the action. The substeps provide specific guidance on how to create a user account in Windows. Words that are highlighted in bold represent menu items, button names, and field names.

An underscore precedes each step and substep. You are encouraged to use these markers to track your progress. As you complete a step, place an X or a check mark on the underscore to indicate that it is completed. Tracking your progress in this manner helps you to stay focused in case of interruptions during a lengthy exercise.

Exercise 1. Importing and testing a message flow

What this exercise is about

This exercise introduces you to the IBM Integration Bus development environment. To become familiar with the IBM Integration Toolkit views and navigator, you import a simple message flow project and examine the message flow components and properties. You also use the IBM Integration Toolkit Flow exerciser to test the message flow.

What you should be able to do

After completing this exercise, you should be able to:

- Import an IBM Integration Bus project interchange file
- Use the Message Flow editor to examine the message flow components and properties
- Test the message flow by using the IBM Integration Toolkit Flow exerciser

Introduction

In the first part of this exercise, you start the IBM Integration Toolkit and verify that the development integration node and integration server are active.

In the second part of this exercise, you import an IBM Integration Bus project interchange file that contains a message flow application.

Important

The exercises in this course use a set of lab files that might include scripts, applications, files, solution files, project interchange files, and others. The course lab files can be found in the `C:\labfiles` directory.

The exercises point you to the lab files as you need them.

In this exercise, you import an IBM Integration Bus project interchange (`.zip`) file that contains a simple application. The application contains a simple message flow that receives XML data over HTTP. The flow transforms

the input XML structure into a different output XML structure by using a Mapping node, and sends this back to the HTTP request.

The message flow contains three nodes.

- An HTTP Input that receives the message
- A Mapping node that is named Map that transforms the message.
- An HTTP Reply node the returns the message

You learn more about these nodes later in the course.

In the third part of this exercise, you examine the message flow and learn how to access message flow node properties, terminal information, and connection information. You also use the XML Schema editor to examine an XML schema and the Graphical Map editor to examine an Integration Bus map.

In the fourth part of this exercise, you use the Flow exerciser to deploy and test a message flow application. The project interchange file contains a sample XML that you use to test the message flow by using the Integration Toolkit Flow exerciser.

The <SalesEnvelope> portion of the message is shown here. It contains a header, sales list, and trailer.

```
<SaleEnvelope>
   <Header>
      <SaleListCount>1</SaleListCount>
      <TransformationType>xsl</TransformationType> <
   /Header>
   <SaleList>
      <Invoice>
         <Initial>K</Initial>
         <Initial>A</Initial>
         <Surname>Braithwaite</Surname>
         <Item>
            <Code>00</Code>
            <Code>01</Code>
            <Code>02</Code>
            <Description>Twister</Description>
            <Category>Games</Category>
            <Price>00.30</Price>
            <Quantity>01</Quantity>
         </Item>
         <Item>
            <Code>02</Code>
            <Code>03</Code>
            <Code>01</Code>
            <Description>The Times Newspaper</Description>
            <Category>Books and Media</Category>
            <Price>00.20</Price>
            <Quantity>01</Quantity>
         </Item>
         <Balance>00.50</Balance>
         <Currency>Sterling</Currency>
      </Invoice>
...Another invoice
   </SaleList>
   <Trailer>
      <CompletionTime>12.00.00</CompletionTime>
   </Trailer>
</SaleEnvelope>
```

The Mapping node in the message flow transforms the input message. The <SalesEnvelopeA> portion of the output message is shown here.

```
<SaleEnvelopeA>
   <SaleListA>
      <Statement>
         <Customer>
            <Initials>KA</Initials>
            <Name>Braithwaite</Name>
            <Balance>00.50</Balance>
         </Customer>
         <Purchases>
            <Article>
               <Desc>Twister</Desc>
               <Cost>0.48</Cost>
               <Qty>01</Qty>
            </Article>
            <Article>
               <Desc>The Times Newspaper</Desc>
               <Cost>0.32000000000000006</Cost>
               <Qty>01</Qty>
            </Article>
         </Purchases>
         <Amount> 0.8
            <Currency>Sterling</Currency>
         </Amount>
         <Style>Full</Style>
         <Type>Monthly</Type>
      </Statement>
...Another Statement
   </SaleListA>
</SaleEnvelopeA>
```


Requirements

This lab requires the following elements:

- A lab environment with the IBM Integration Bus V10 Integration Toolkit
- The lab files in the C:\labfiles\Lab01-TestSimpleFlow directory

Exercise instructions

Part 1: Start the IBM Integration Toolkit

In this part of the exercise, you start the IBM Integration Toolkit so that you can develop and test a simple message flow.

__ 1. Log in to the course image by using the following credentials.

User name: `ibadmin`

Password: `web1sphere`

__ 2. Start the IBM Integration Toolkit by double-clicking the **IBM Integration Toolkit** shortcut on the desktop.

You can also start the IBM Integration Toolkit by clicking **Start > All Programs > IBM Integration Bus 10.0.0.0 > IBM Integration Toolkit 10.0.0.0** from the Windows desktop.

After several moments, the Integration Toolkit starts and the **Welcome** page displays.

__ 3. In the upper-right corner of the page, click **Go to the Integration Toolkit** to close the **Welcome** page and open the Integration Development perspective.

__ 4. Verify that the IBM Integration Toolkit automatically created an integration node and integration server for testing Integration Bus applications.

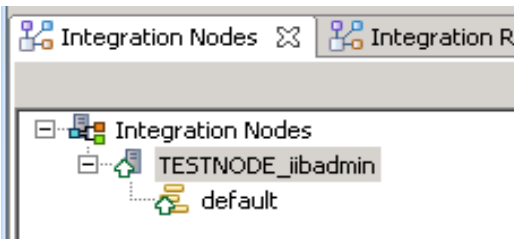
__ a. In the lower left of the Application Development perspective, click the **Integration Nodes** tab.

__ b. Verify that the integration node that is named **TESTNODE_iibadmin** is running as indicated by the green up arrow. If the integration node is not running, start it by right-clicking **TESTNODE_iibadmin** and then clicking **Start**.

__ c. Verify that the integration server that is named **default** is running as indicated by the green up arrow.

If you do not see the integration server, right-click the integration node and then click **Refresh**.

If the integration server is not running, start it by right-clicking **default** and then clicking **Start**.



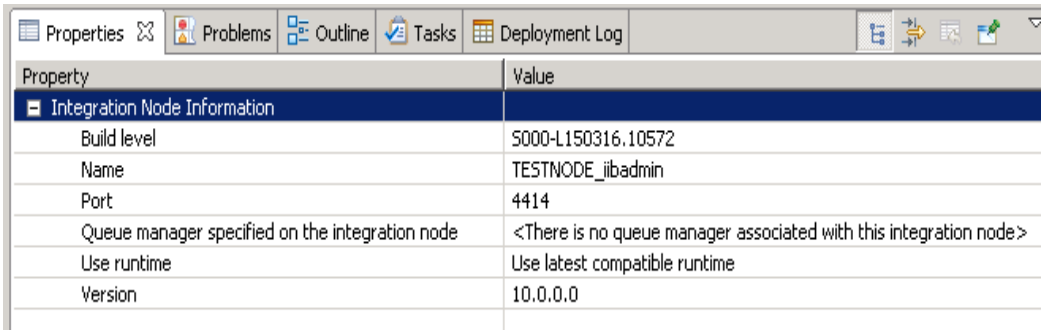
__ 5. Examine the properties of the integration node **TESTNODE_iibadmin**.

__ a. Click the integration node **TESTNODE_iibadmin** in the **Integration Nodes** view.

__ b. Click the **Properties** tab to show the integration node properties.

The development node **TESTNODE_iibadmin** is listening on port 4414, which is the default port for a new integration node.

You should also see that no IBM MQ queue manager is associated with this integration node.

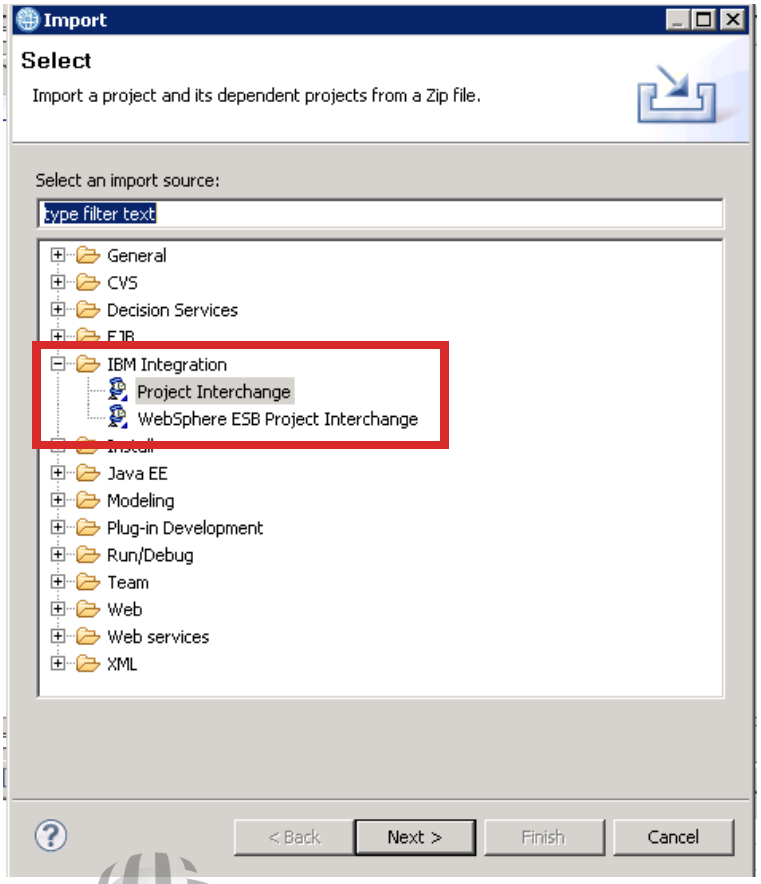


Part 2: Import a project interchange file

In this part of the exercise, you import an IBM Integration Bus project interchange file into the IBM Integration Toolkit. The project interchange file contains an application with the simple message flow that is described in the exercise introduction.

__ 1. Click **File > Import** from the Integration Development perspective menu bar.

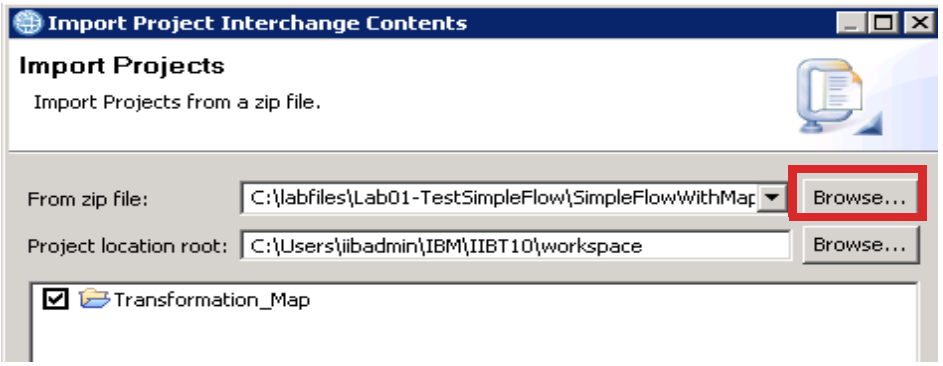
__ 2. Click **IBM Integration > Project Interchange**, and then click **Next**.



__ 3. To the right of **From zip files**, click **Browse** to locate the project interchange file.

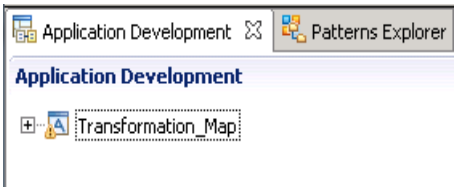
__ 4. Browse to the `C:\labfiles\Lab01-TestSimpleFlow` directory, select the `SimpleFlowWithMap_PI.zip` file, and then click **Open**.

__ 5. Ensure that the **Project location root** is set to the current workspace of `C:\Users\iibadmin\IBM\IIBT10\workspace`.



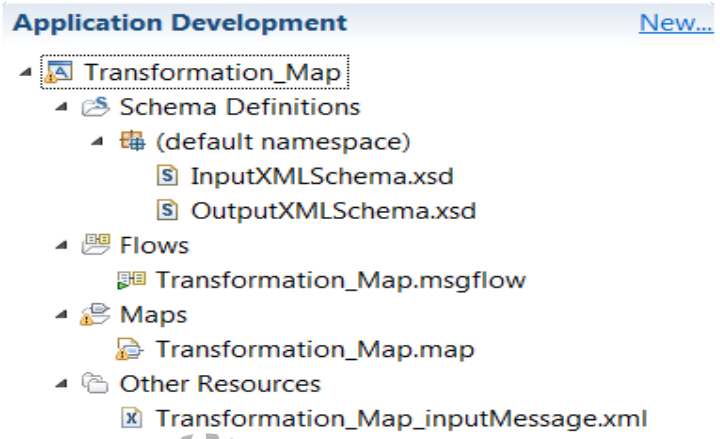
__ 6. This project interchange file contains one application that is named **Transformation_Map.** Ensure that it is selected and then click **Finish**.

__ 7. Verify that the **Transformation_Map** application is imported into the Application Development navigator.



Part 3: Examine the application objects and message flow components

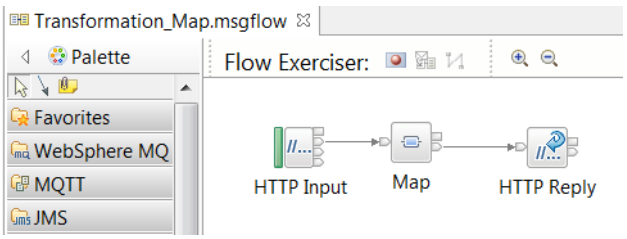
__ 1. In the **Application Development** view, expand the application folders to show the contents of the application.



The figure shows the objects that are included in the **Transformation_Map** application. The application includes the following files:

- The XML schema definition files that define the input message (`InputXMLSchema.xsd`) and the output message (`OutputXMLSchema.xsd`)
- The message flow (`Transformation_Map.msgflow`)
- The transformation map (`Transformation_Map.map`) that the Mapping node references
- A sample input file (`Transformation_Map_inputMessage.xml`)

__ 2. Double-click the message flow file `Transformation_map.msgflow` in the **Application Development** view to open it in the Message Flow editor.

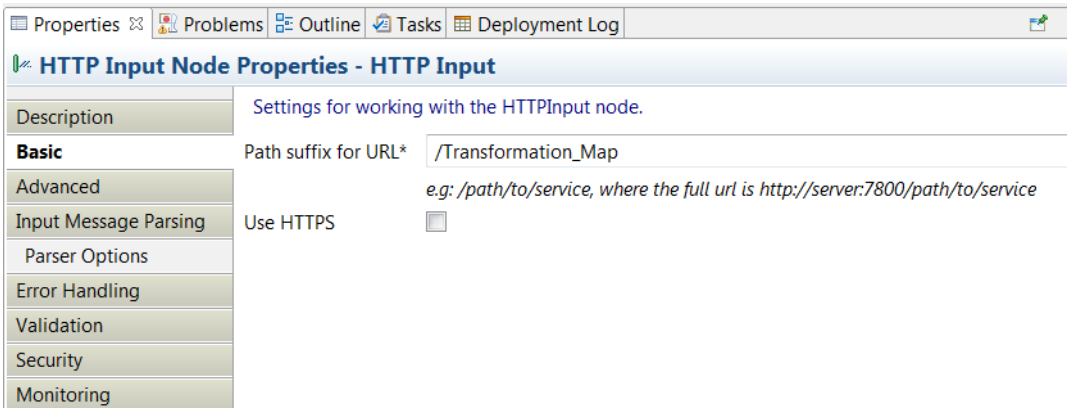


The message flow contains three nodes:

- An HTTP Input node
- A Mapping node that is named **Map**
- An HTTP Reply node

__ 3. In the Message Flow editor, double-click the HTTP Input node to display the node properties. The node properties are shown in the **Properties** view at the bottom of the Integration Toolkit.

You can also click the node to highlight it, and then click the **Properties** view.



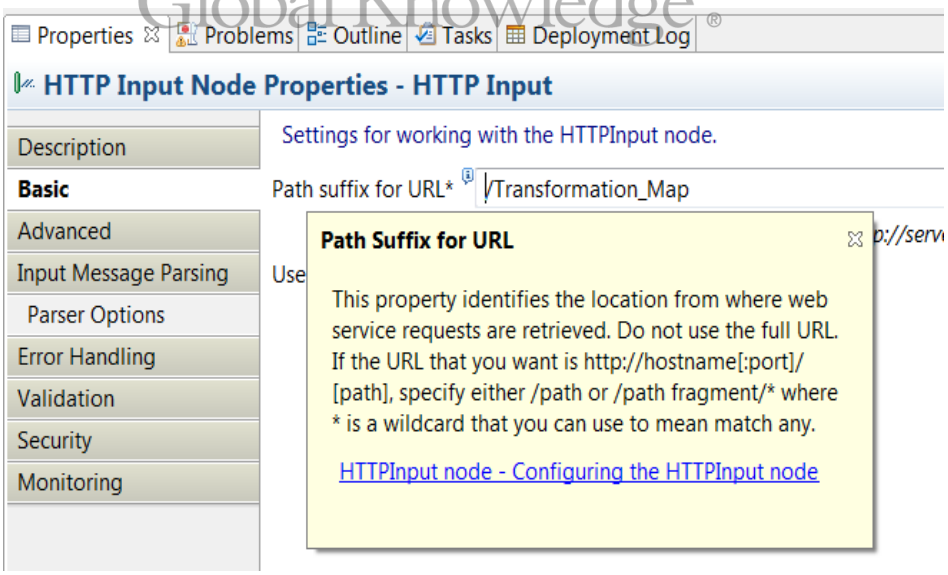
Several properties tabs are shown. By default, the **Basic** tab is selected.

__ 4. The Integration Toolkit provides context-sensitive help for node properties. To display the context-sensitive help for the **Path suffix for URL** field:

__ a. Click in the **Path suffix for URL** field on the **Basic** tab.

__ b. Click the “Information” icon that appears to the left of the **Path suffix for URL** field.

__ c. Close the context-sensitive help window.



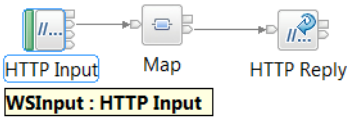
__ 5. Click each of the **Properties** tabs for the HTTP Input node to examine the node properties and answer the following questions.

__ a. What is the message domain (parser) that the HTTP Input node uses in this flow?

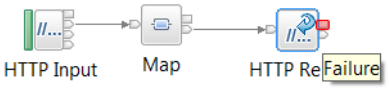
__ b. What is the parse timing?

__ 6. Examine the node properties for the other nodes in the message flow.

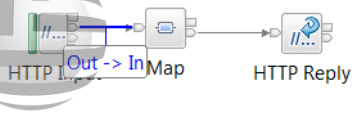
__ 7. In the Message Flow editor, hover the mouse pointer over each of the nodes. The node type and node name are displayed in the format `NodeType : NodeName`.



__ 8. In the Message Flow editor, hover the mouse pointer over the node terminals to display the terminal names.



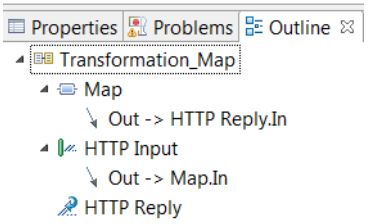
__ 9. In the Message Flow editor, hover the mouse pointer over the wires that connect the nodes to display the connection information.



The names of the terminals to which the wire is connected are displayed in the format `SourceTerminal -> TargetTerminal`.

__ 10. Click the **Outline** tab (in the same view group as the **Properties** view) to display the connection information.

__ 11. Expand the **Map** and **HTTP Input** node entries in the **Outline** view.



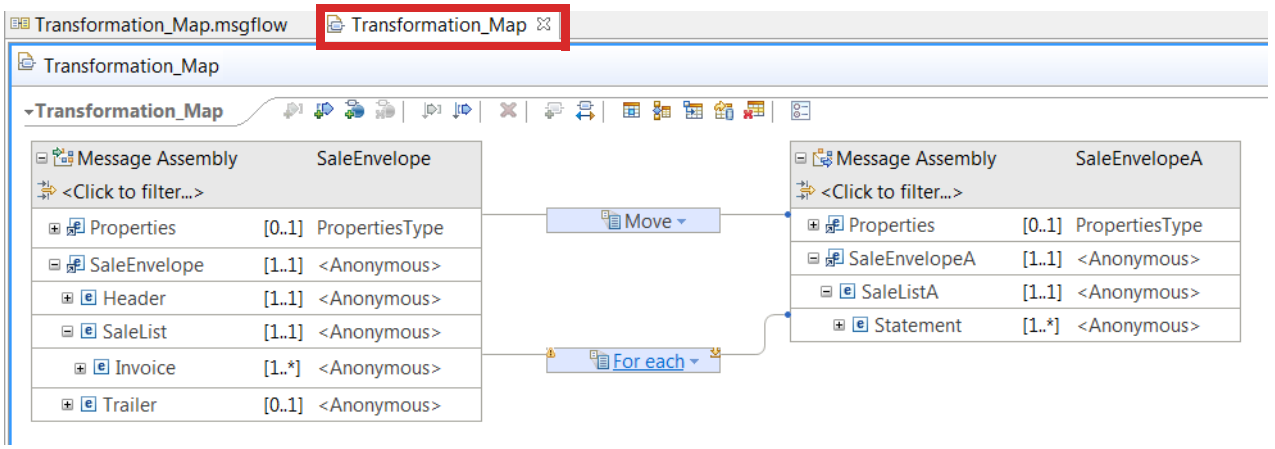
The **Outline** view shows that the **Out** terminal of the Map node is wired to the **In** terminal of the HTTP Reply node as denoted by **HTTP Reply.In**.

The **Out** terminal of the HTTP Input node is wired to the **In** terminal of the Map node as denoted by **Map.In**.

__ 12. Examine the transformation map.

__ a. Double-click the Map node in the Message Flow editor to display the Graphical Map editor view of the transformation map.

__ b. Double-click the **Transformation Map** tab to expand the Graphical Map editor view so that you can see the entire map.



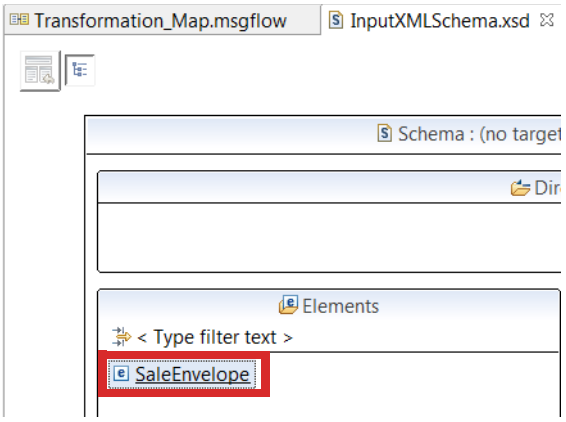
__ c. Double-click the **Transformation Map** tab again so that it resizes the view back to its original size.

__ d. You learn more about message transformation and the Mapping node later in this course. Close the **Transformation Map** tab by clicking the **X** on the tab.

__ 13. Examine the XML schema that defines the message flow input.

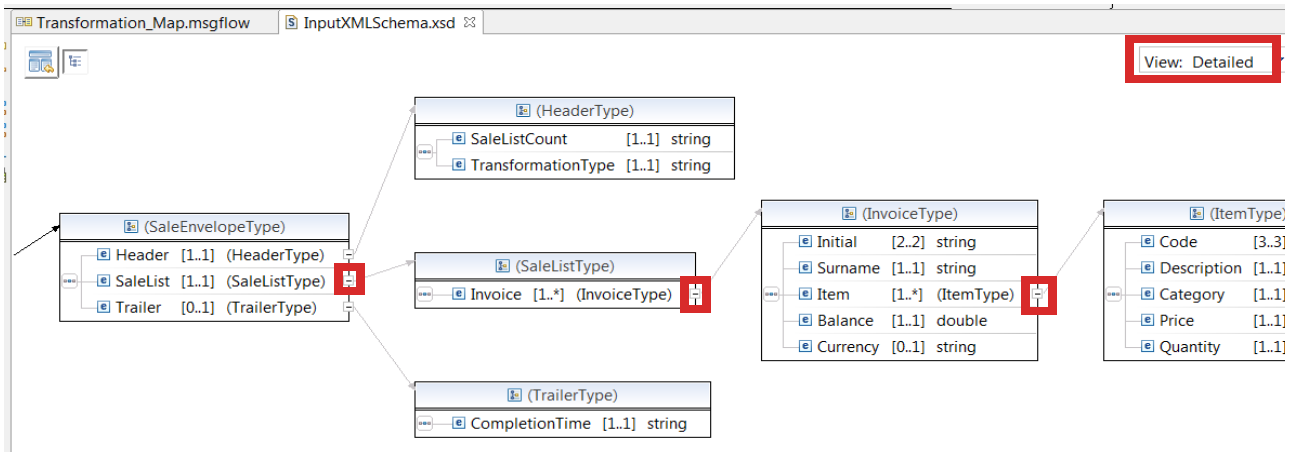
__ a. In the **Application Development** view, double-click **InputXMLSchema.xsd** file under the **Schema Definitions > (default namespace)** folder to open it in the XML Schema editor.

__ b. In the XML Schema editor, double-click **SalesEnvelope** to display the schema details.



__ c. Double-click the **InputXMLSchema.xsd** tab to expand the XML Schema editor view.

__ d. Ensure that **View** is set to **Detailed** and then click the plus signs to expand the XML schema.



__ e. After you finish examining the XML schema, double-click the **InputXMLSchema.xsd** tab to resize the view.

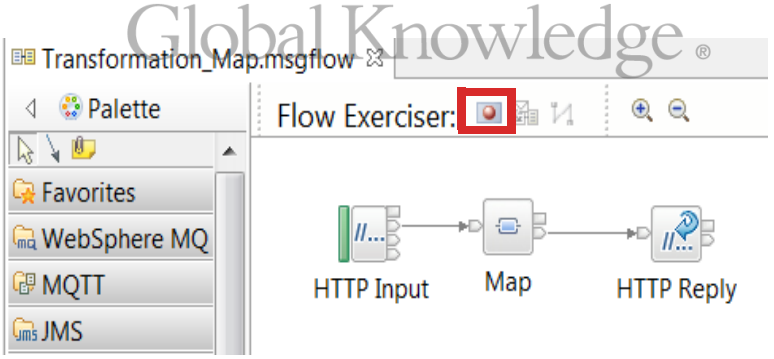
__ f. Close the **InputXMLSchema.xsd** tab.

Part 4: Test with the Flow exerciser

In this part of the exercise, you test the message flow by using the IBM Integration Toolkit Flow exerciser and a test message.

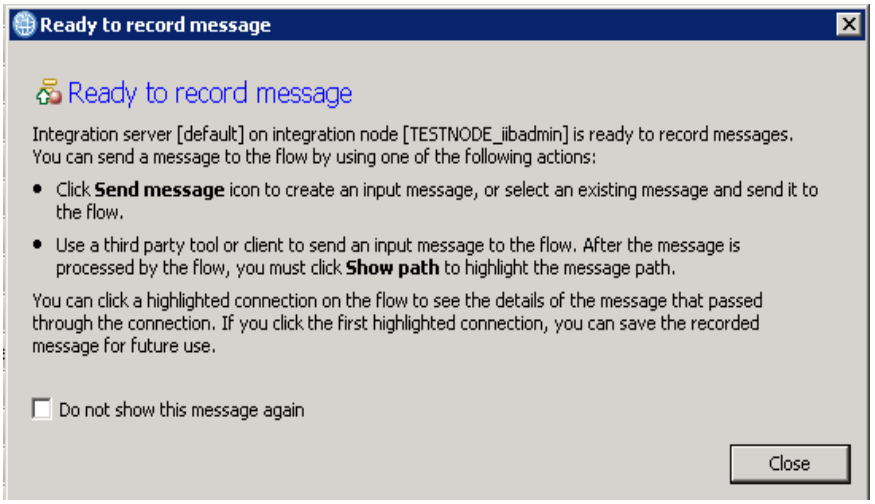
__ 1. Start the Flow exerciser by clicking the **Start flow exerciser** icon at the top of the Message Flow editor.

The **Start flow exerciser** icon is a toggle with which you can switch between Flow exerciser mode and Message flow editor mode.



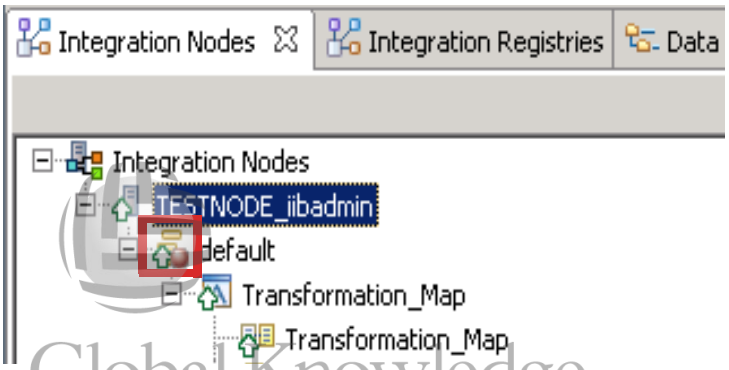
The Flow exerciser creates the BAR file and deploys it to the integration server. If any messages are displayed while the Flow exerciser is running, ignore them; they are information messages and disappear on their own.

__ 2. After a few seconds, you might see a window that indicates that the Flow exerciser is ready to record a message. If you see this information window, click **Close**.



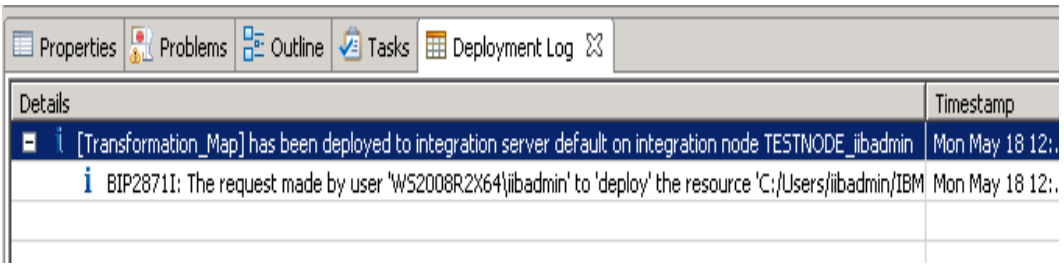
__ 3. Verify that the Flow exerciser **Record** mode is running on the **default** integration server.

In the **Integration Nodes** view, verify that **default** integration server shows the **Recording** icon.



__ 4. Examine the **Deployment Log** view near the bottom of the Application Development perspective.

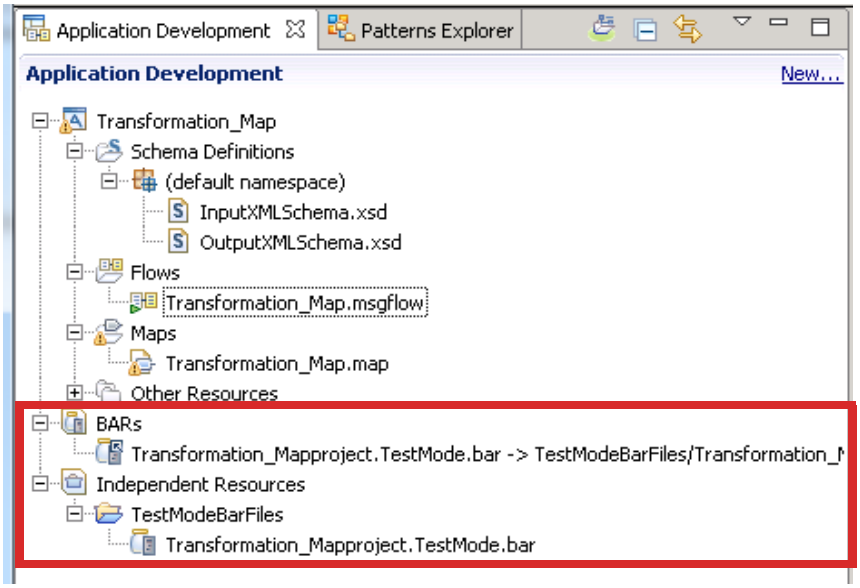
You should see an entry in the log that indicates that the **Transformation_Map** application was deployed to the **default** integration server on the **TESTNODE_iibadmin** integration node.



__ 5. Verify that the message flow application is running on the **default** integration server.

In the **Integration Nodes** view, verify that the **Transformation_Map** application is displayed under the **default** integration server.

__ 6. Examine the Application Development navigator. You should now see a project that is named **BARs** and another project that is named **TestModeBarFiles** under the **Independent Resources** folder.

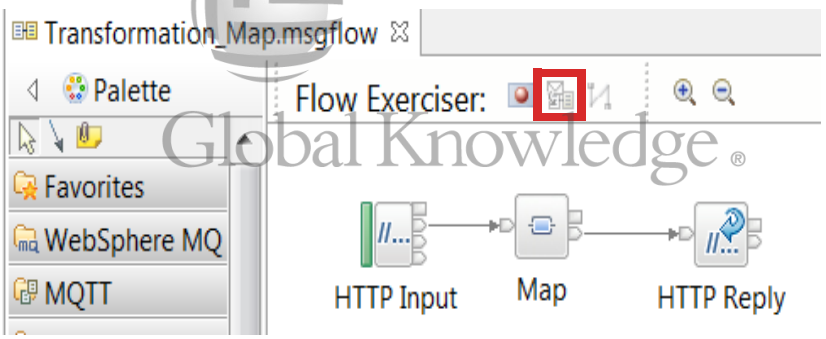


These projects contain the BAR file (with a `.bar` extension) that the Flow exerciser built and deployed to the integration server.

You use the BAR File editor to examine and a BAR file in the next exercise.

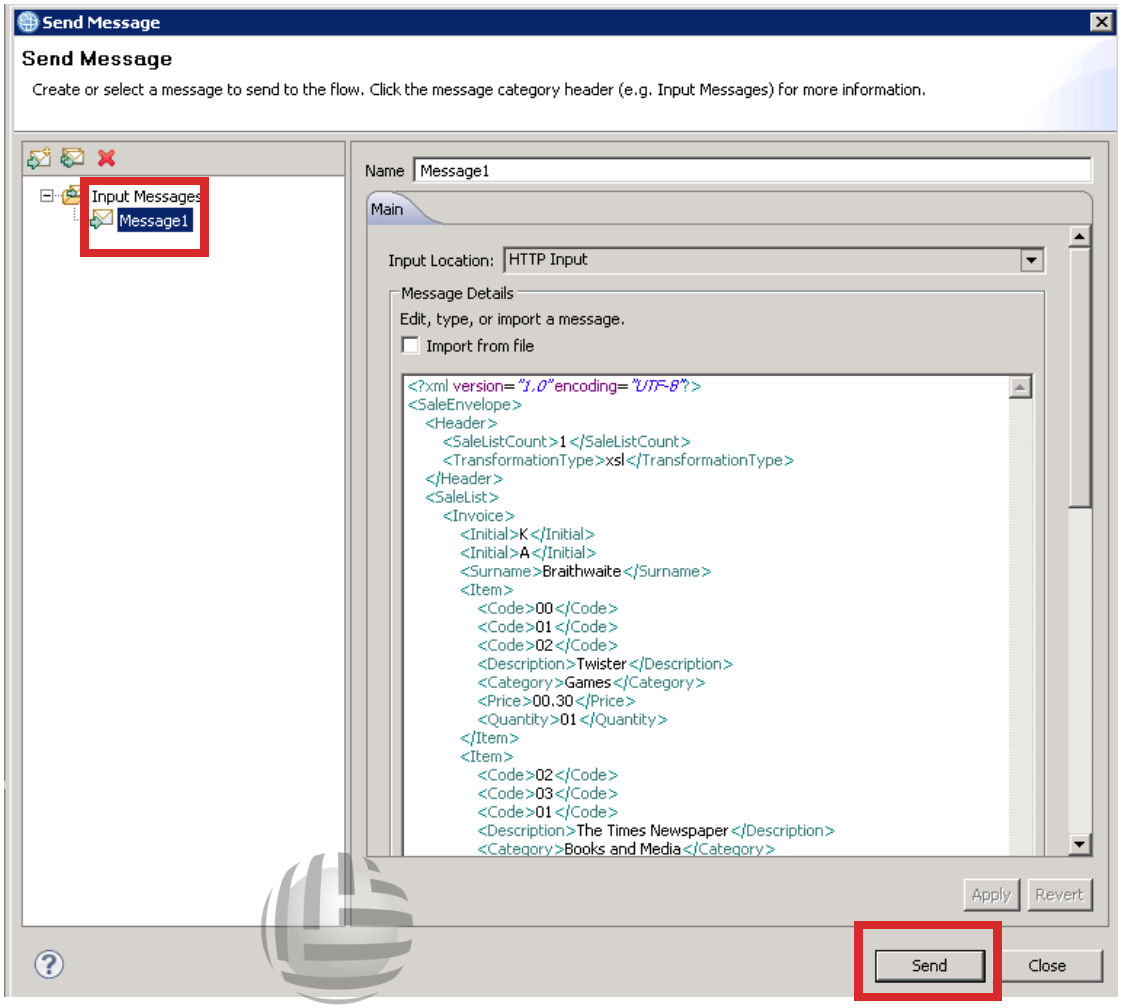
__ 7. Send a test message to the **Transformation_Map** message flow.

__ a. In the Message Flow editor, click the Flow exerciser **Send a message to the flow** icon.



__ b. In the **Send Message** window, click **Message1**. Take a moment to examine the message.

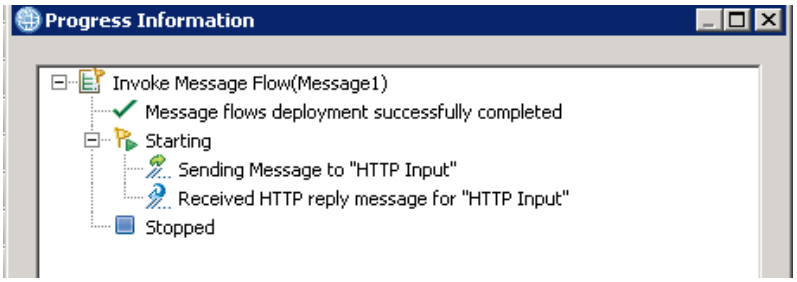
__ c. Click **Send**.



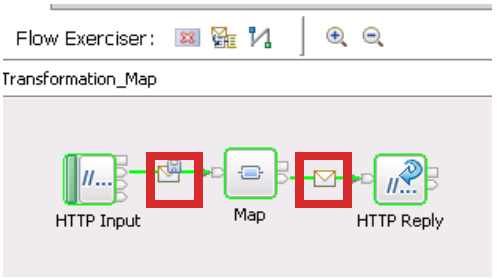
__ 8. After some moments, the test runs. The **Progress Information** window shows you the status of message flow test.

In this exercise, the **Progress Information** shows that the message was sent to the HTTP Input node and that the reply message was received.

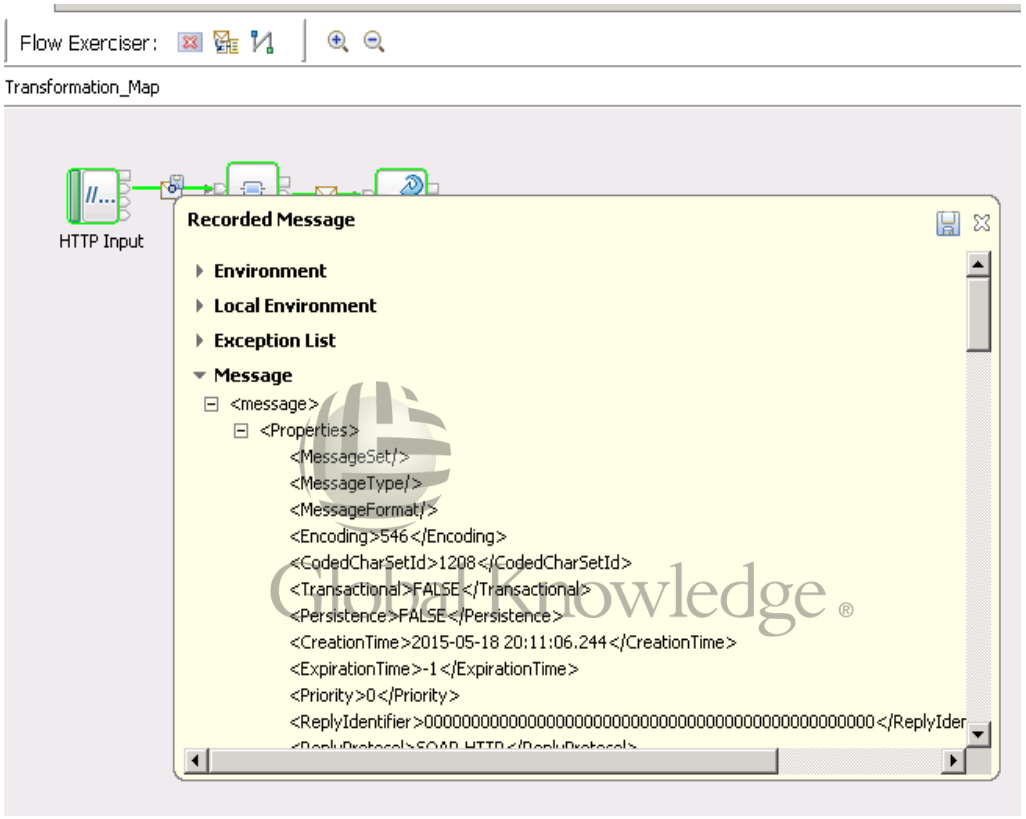
Click **Close**.



__ 9. In the Message Flow editor, the message flow updates to show the message path through the flow. The message icons on the connections indicate the points where you can view the message.



__ a. Click the message icon that is between the HTTP Input node and the Map node to view the input message.

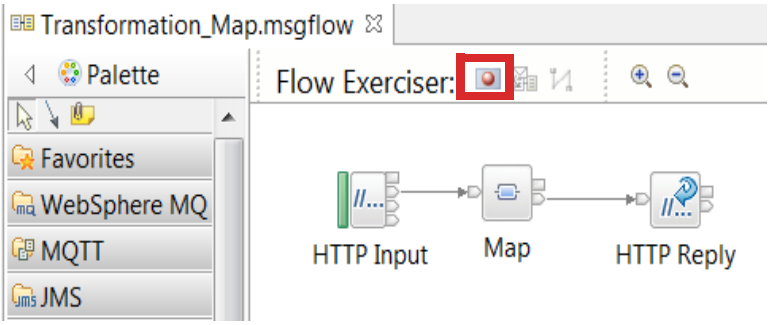


__ b. After you review the input message, close the message window.

__ c. Click the message icon that is between the Map node and the HTTP Reply to view the message after the Map node transforms it.

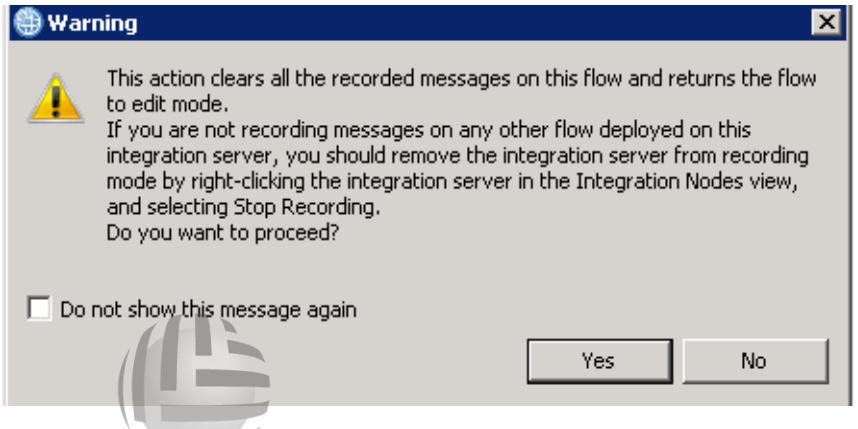
__ d. After you review the output message, close the message window.

__ 10. Return to the edit mode by clicking the **Return to edit mode** icon at the top of the Message Flow editor.



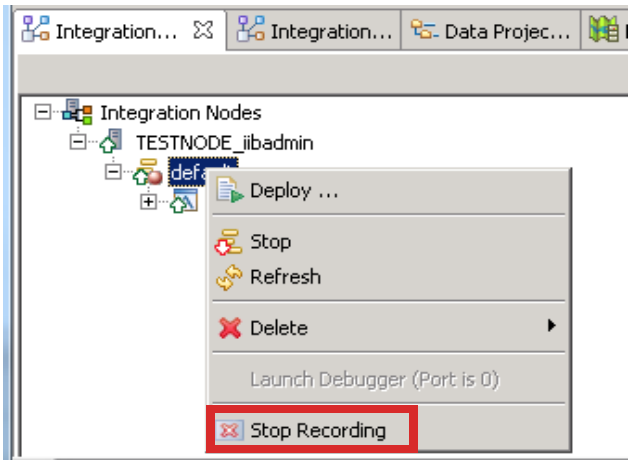
__ 11. A warning message reminds you that changing back to edit mode clears all the recorded messages on this flow. It also reminds you to stop recording mode on the integration server if you are finished with testing,

Click **Yes**.



__ 12. In the **Integration Nodes** view, stop the Flow exerciser recording mode on the **default** integration server.

Right-click the **default** integration server and then click **Stop Recording**.

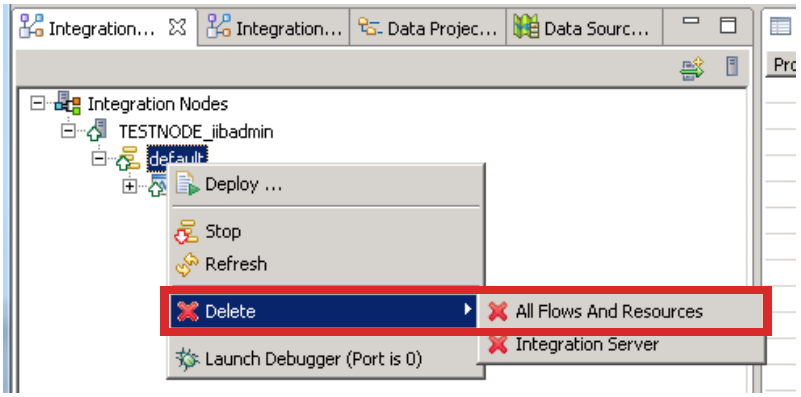


__ 13. Close the **Transformation_Map** message flow in the Message flow editor.

__ 14. To avoid any potential conflicts with the next exercises, remove the **Transformation_Map** application from the **default** integration server.

__ a. Right-click the default integration server and then click **Delete > All Flows and Resources**.

__ b. Click **OK** on the confirmation window.



__ 15. Close all open editor tabs.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you started the IBM Integration Toolkit and verified that the development integration node and integration server are active.

In the second part of this exercise, you imported an IBM Integration Bus project interchange file that contained a message flow application.

In the third part of this exercise, you examined the message flow and saw how to access message flow node properties, terminal information, and connection information. You also used the XML Schema editor to examine an XML schema and the Graphical Map editor to examine an Integration Bus map.

In the fourth part of this exercise, you used the Flow exerciser to deploy and test a message flow application.

Exercise 2. Creating a message flow application

What this exercise is about

In this exercise, you create a simple message flow application, and use the IBM Integration Flow exerciser to test it. You also use the IBM Integration web user interface to check the status of the integration node, integration server, and message flow application at run time.

What you should be able to do

After completing this exercise, you should be able to:

- Create a message flow application
- Use the IBM Integration Flow exerciser to test the message flow application
- Use the IBM Integration web user interface to check the status of the integration node, integration server, and message flow application at run time

Introduction

In the first part of this lab exercise, you create a message flow that uses HTTP nodes and acts as a simple web service. The message flow also writes the HTTP request message to a file. This simple web service is called from a web browser.

The message flow contains:

- An HTTPInput node that receives the HTTP request message from the web browser
- A FileOutput node that writes the request message to a file
- An HTTPReply node that sends the HTTP reply message

The focus in this part of the exercise is on the steps that are required to completely define a message flow, not on the function of the nodes. Each of these nodes is described in detail later in this course.

In the second part of this exercise, you test the message flow by using a web browser and the Integration Toolkit Flow exerciser.

In the third part of this exercise, you use the IBM Integration web user interface to check the status of the integration node, integration server, and message flow application.

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit
- A web browser such as Internet Explorer

Exercise instructions

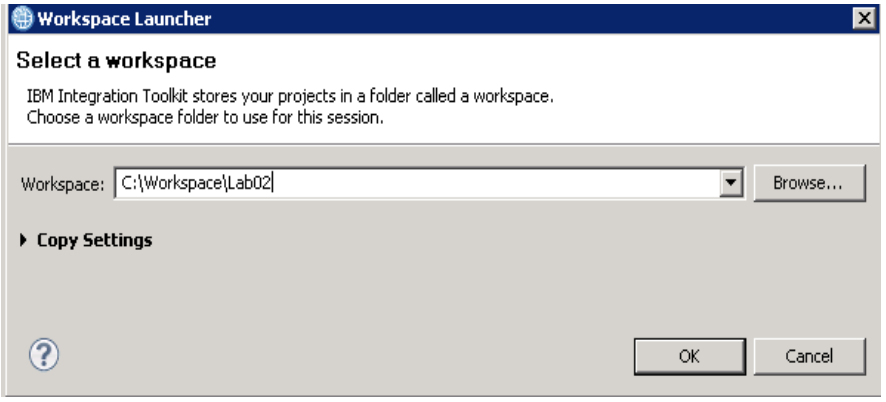
Part 1: Create the message flow application

In this part of the exercise, you create a simple HTTP flow that acts as a simple web service that writes to a file.

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab02`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab02`**

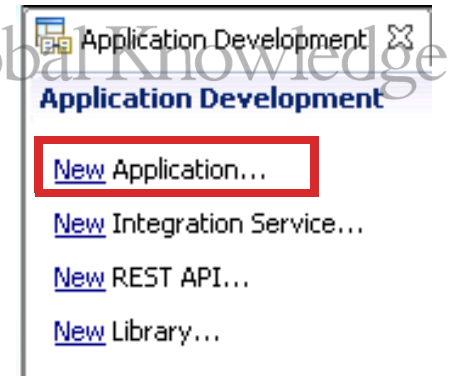


__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

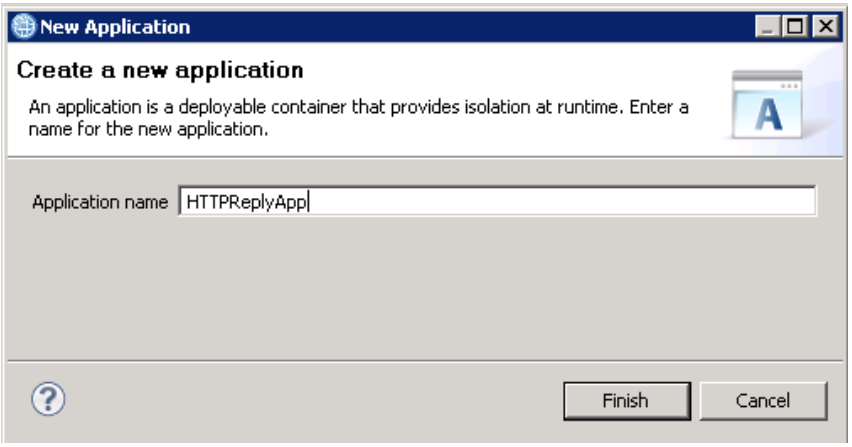
__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 2. Create an application that is named **HTTPReplyApp**.

__ a. In the **Application Development** view, click **New Application**.



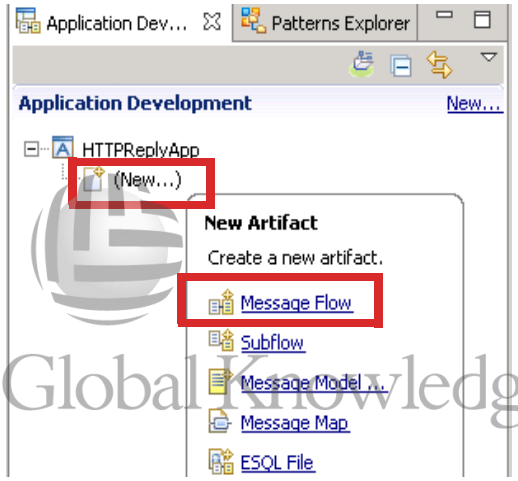
__ b. For the application name type: **HTTPReplyApp**



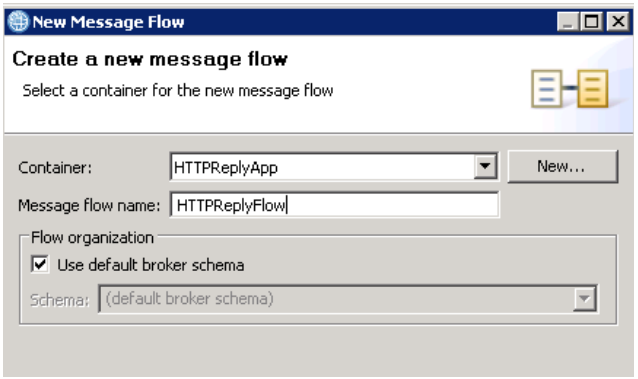
__ c. Click **Finish**.

__ 3. Create a message flow that is named **HTTPReplyFlow**.

__ a. Click **(New…)** under the **HTTPReplyApp** folder in the **Application Development** view, and then click **Message Flow**.

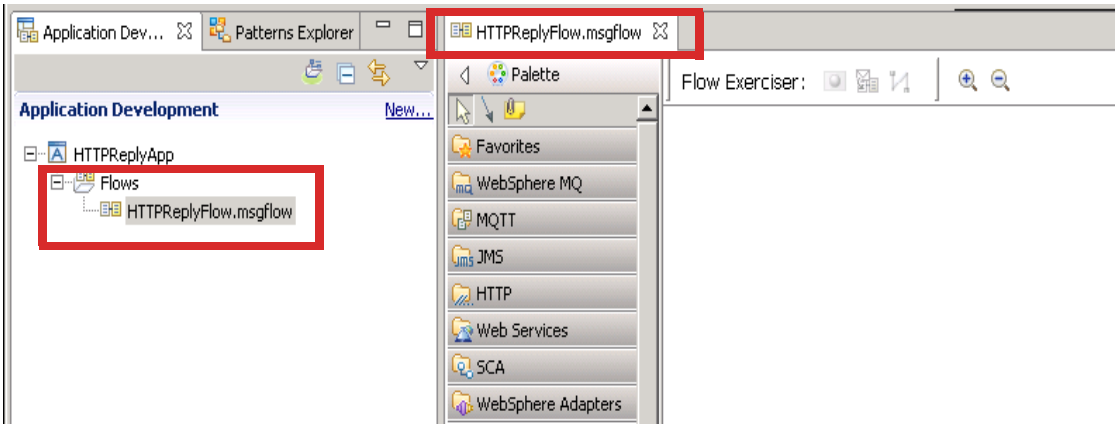


__ b. For the **Message flow name** type: **HTTPReplyFlow**



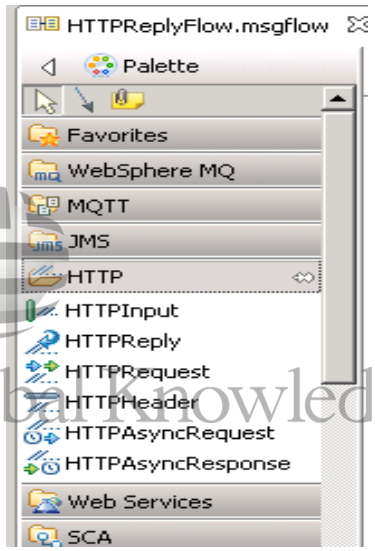
__ c. Click **Finish**.

You should see the new message flow in the **Application Development** view under the **Flows** folder in the application. The **HTTPReplyFlow.msgflow** file is also open in the Message Flow editor.



__ 4. In the Message Flow editor, create a message flow that contains an HTTPInput node, FileOutput node, and HTTPReply node.

__ a. Expand the **HTTP** drawer in the Message Flow editor **Palette**.

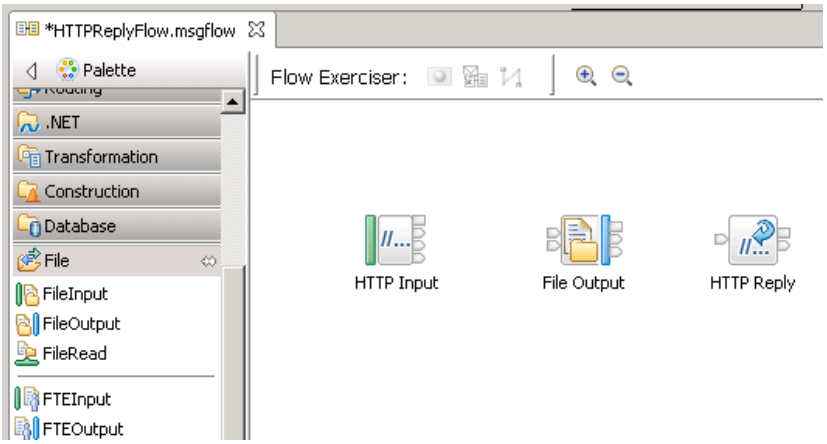


__ b. Click **HTTPInput** and then click the Message Flow editor view to add the node on the canvas.

__ c. Click **HTTPReply** and then click the Message Flow editor view to add the node on the canvas.

__ d. Expand the **File** drawer in the **Palette**.

__ e. Click **FileOutput** and then click the Message Flow view between the HTTP node to add the node between the HTTP nodes.

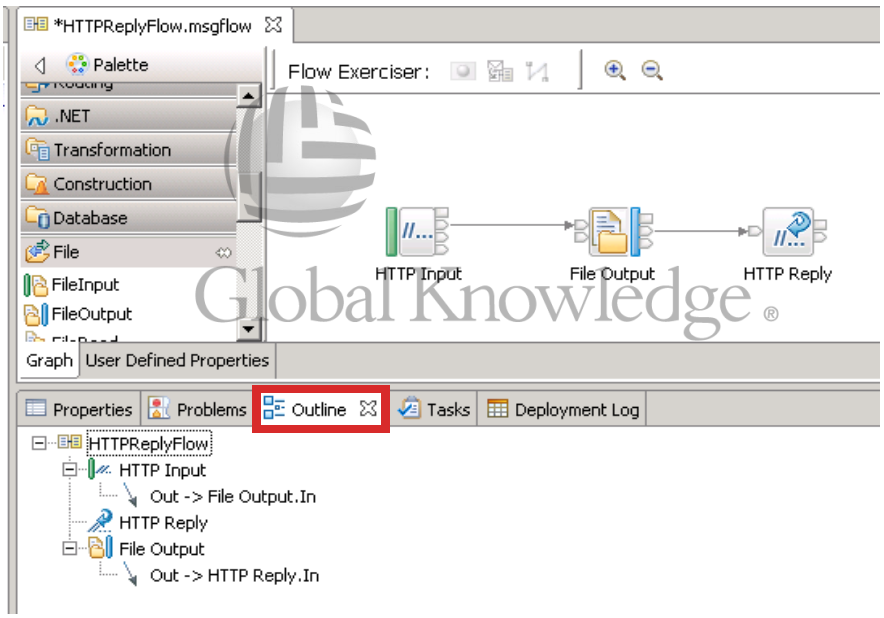


__ 5. Wire the nodes.

__ a. Click the **Out** terminal on the HTTPInput node and then click the **In** terminal of File Output node to connect the nodes.

Hover your cursor over the terminal or use the **Outline** view to ensure that you wired the terminals correctly.

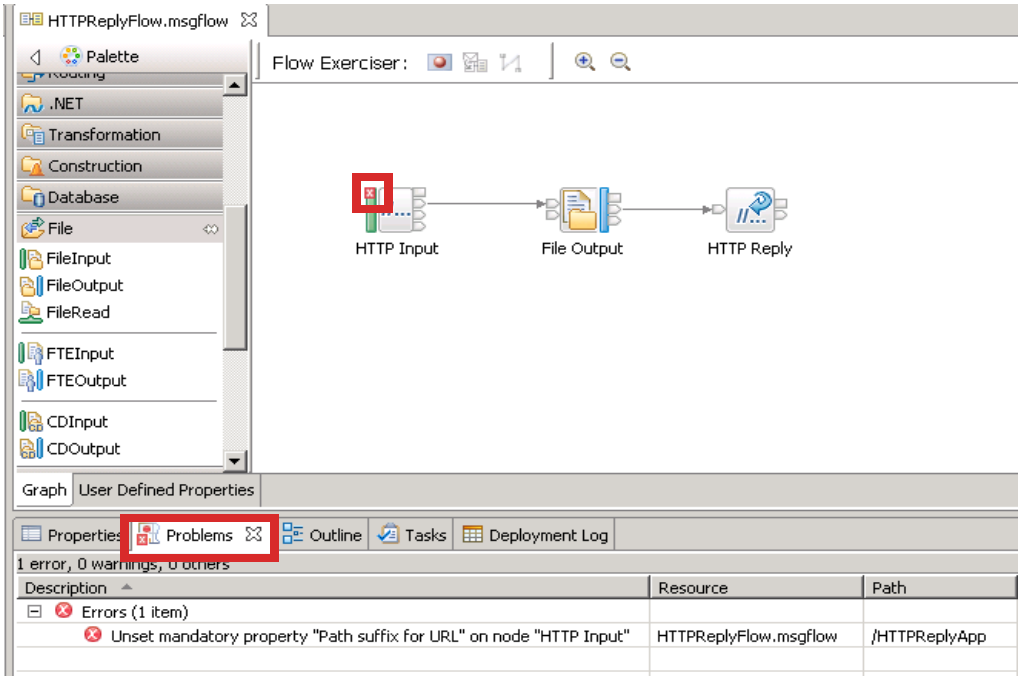
__ b. Wire the **Out** terminal of the FileOutput node to the **In** terminal of the HTTPReply node.



__ 6. Save the message flow by clicking the **Save** icon or by pressing Ctrl+S

__ 7. You should see that the HTTP Input node has an error as indicated by the red “X” on the node.

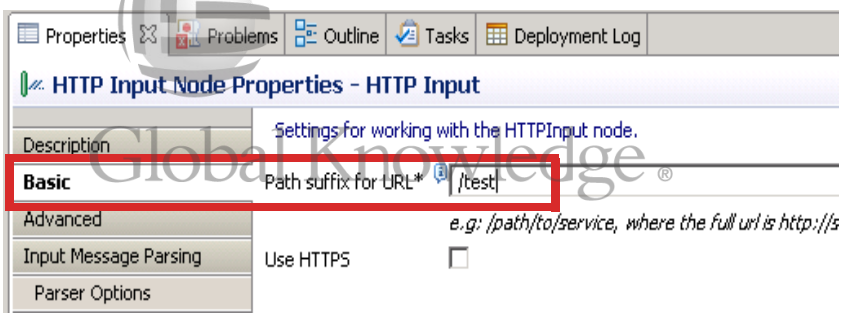
Click the **Problems** view tab to display information about the error. You should see that a missing mandatory property caused the error.



__ 8. Define the node properties for the HTTP Input node.

__ a. Click the HTTPInput node in the message flow canvas and then select the **Properties** view.

__ b. On the **Basic** tab, change the **Path suffix for URL*** property to: `/test`



This property identifies the location from where web service requests are retrieved. Setting the path suffix to `/test` defines a fully resolved URL of `http://server:7800/test`

__ c. Save the message flow.

You should see that the error indicator on the HTTPInput node is gone and that the **Problems** view is now empty.

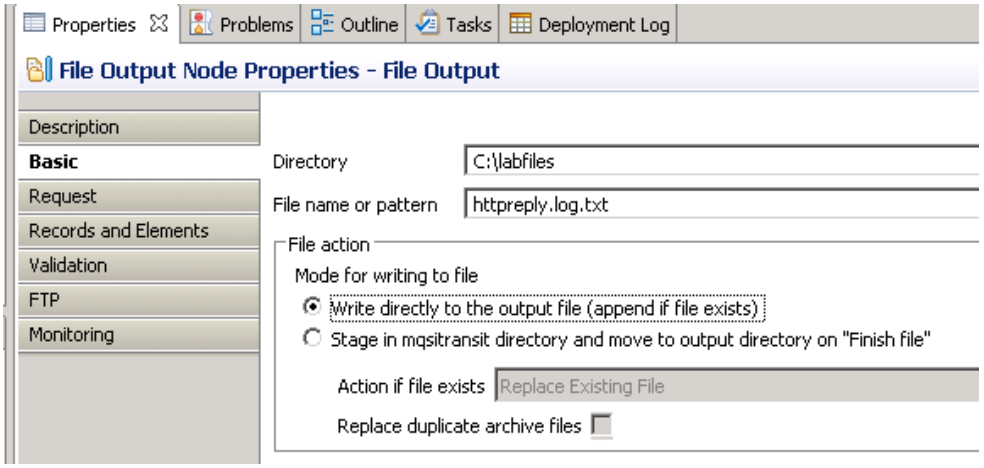
__ 9. Edit the properties of the FileOutput node **Basic** properties, so that the file is written to the **`httpreply.log.txt`** file in the **`C:\labfiles`** directory.

__ a. Click the FileOutput node in the message flow, and then click the **Properties** view tab.

__ b. For the **Directory** property, type: **`C:\labfiles`**

__ c. For the **File name or pattern property**, type: **`httpreply.log.txt`**

__ d. For the **Mode for writing to file**, select **Write directly to the output file (append if file exists)**.



__ e. Save the message flow.

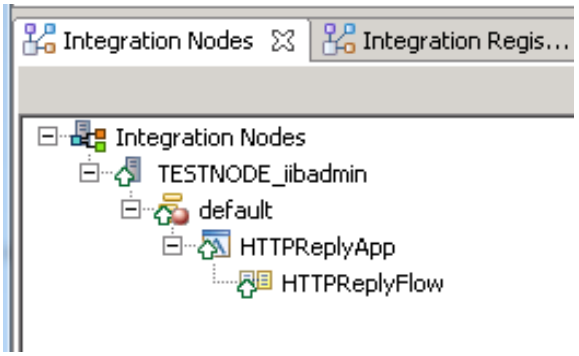
Part 2: Test the message flow

In this part of this exercise, you test the message flow by using a web browser and the Integration Toolkit Flow exerciser.

__ 1. In the Message Flow editor, click the **Start Flow exerciser** icon to deploy the message flow.

The **HTTPReplyApp** application is automatically deployed to the **default** integration server on **TESTNODE_iibadmin**. You should see a window that indicates that the message flow is being deployed.

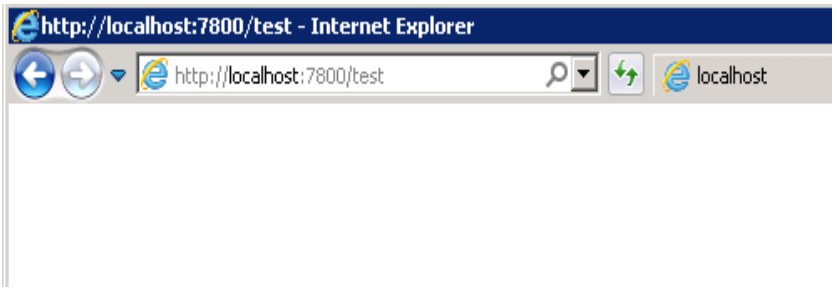
__ 2. In the **Integration Nodes** view, verify that the **HTTPReplyApp** application and the **HTTPReplyFlow** message flow are deployed to the **default** integration server on the **TESTNODE_iibadmin** integration node.



__ 3. Start Internet Explorer by double-clicking the icon on the desktop.

__ 4. Test the flow by entering the following URL in the address bar: **`http://localhost:7800/test`**

In the URL, **localhost** is the server location of the integration node. Port 7800 is the port on which the integration node receives HTTP connections. When you defined the HTTPInput node properties, you specified that the message flow would use the /test URL.



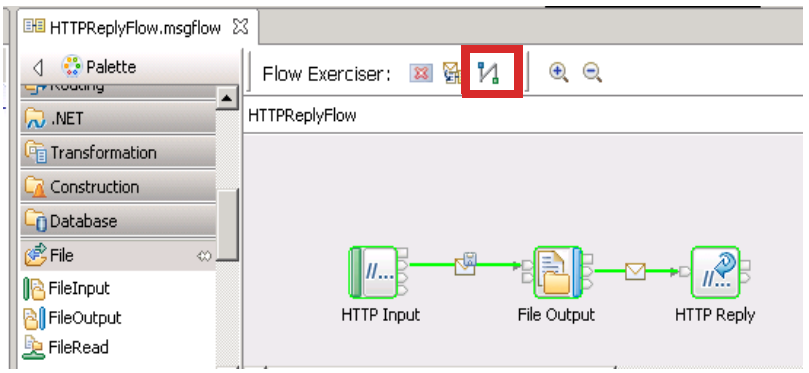
A response is not expected on the browser window from the message flow, but notice there is no error, so the request was successful.

__ 5. Check the output file httpreply.log.txt in the C:\labfiles directory.

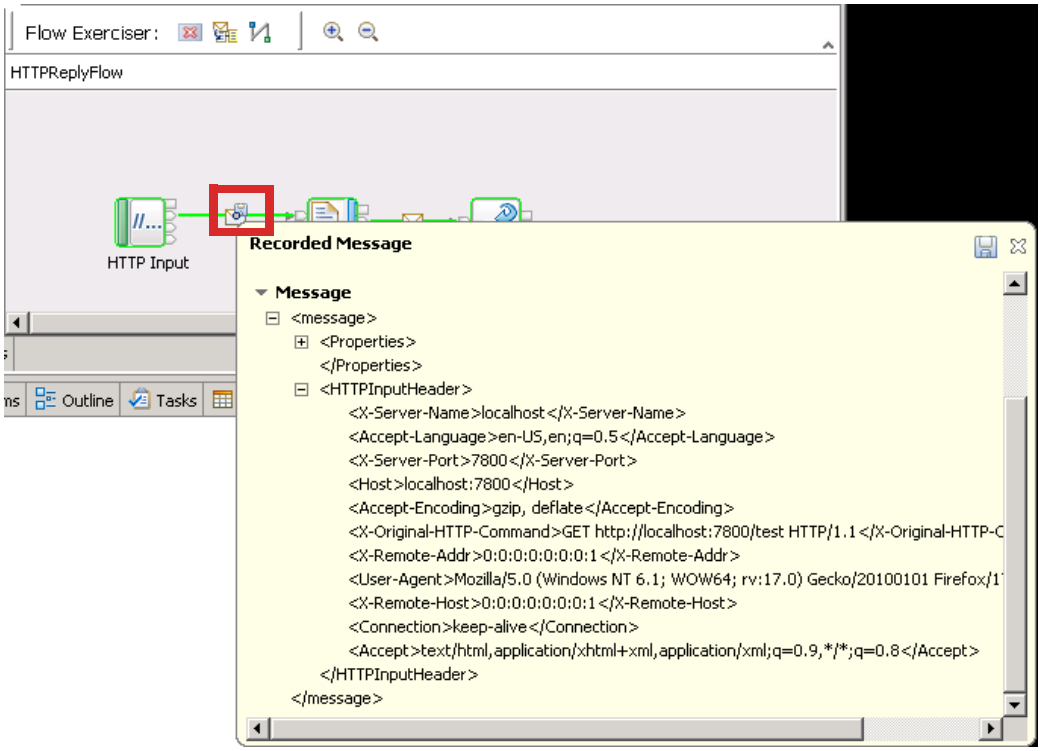
__ a. Start Windows Explorer and go to the C:\labfiles directory.

__ b. Double-click the httpreply.log.txt file in Windows Explorer to view the file in Notepad.

__ 6. In the Message Flow editor, click the Flow exerciser **View path** icon to see the path that the message took through the flow and the data that Flow exerciser captured.



__ 7. Click the message icons to examine the message before and after the FileOutput node processes it.

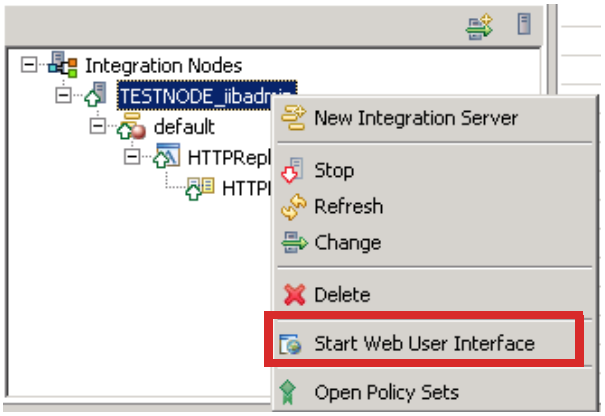


__ 8. Click the Flow exerciser **Return flow to edit mode** icon to stop the Flow exerciser.

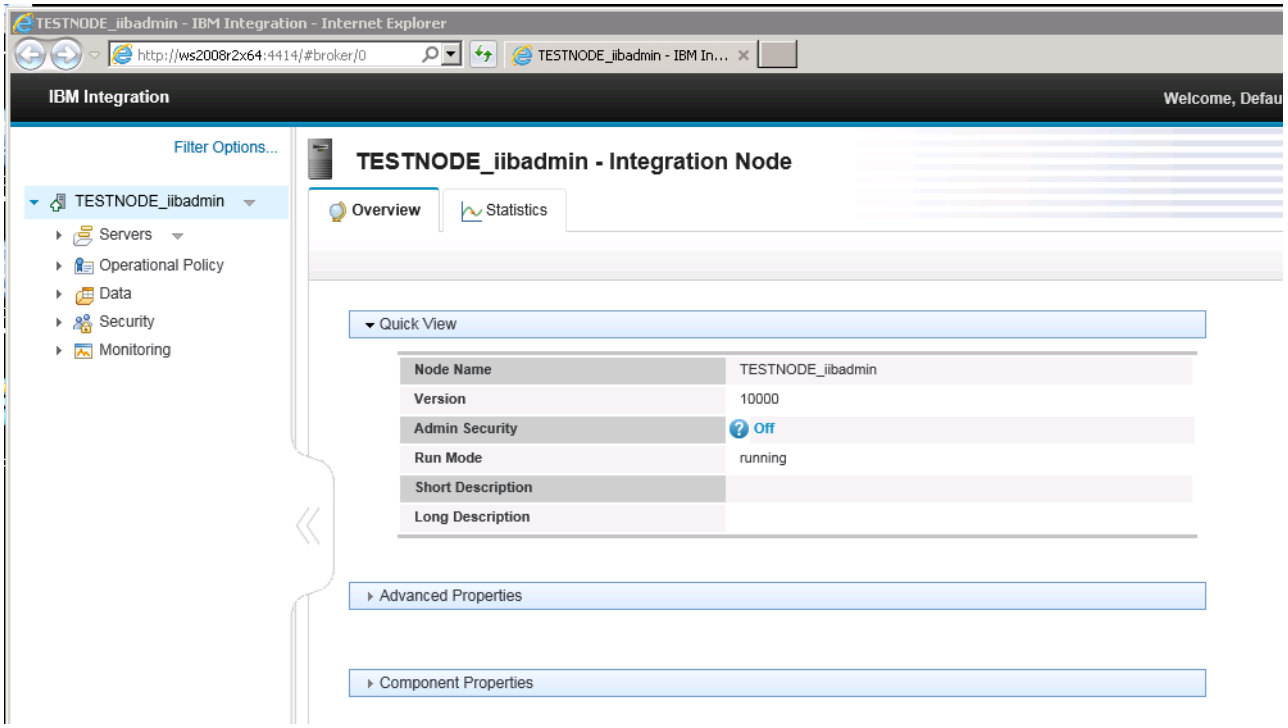
Part 3: Use the IBM Integration web user interface to view status

In this part of the exercise, you use the IBM Integration web console to check the status of the integration node, integration server, and message flow application.

__ 1. In Integration Nodes view, right-click the **TESTNODE_iibadmin** and then click **Start Web User Interface** to start the IBM Integration web user interface in a web browser.



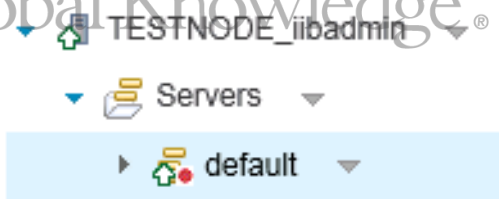
After a few seconds, the IBM Integration web user interface opens in Internet Explorer.



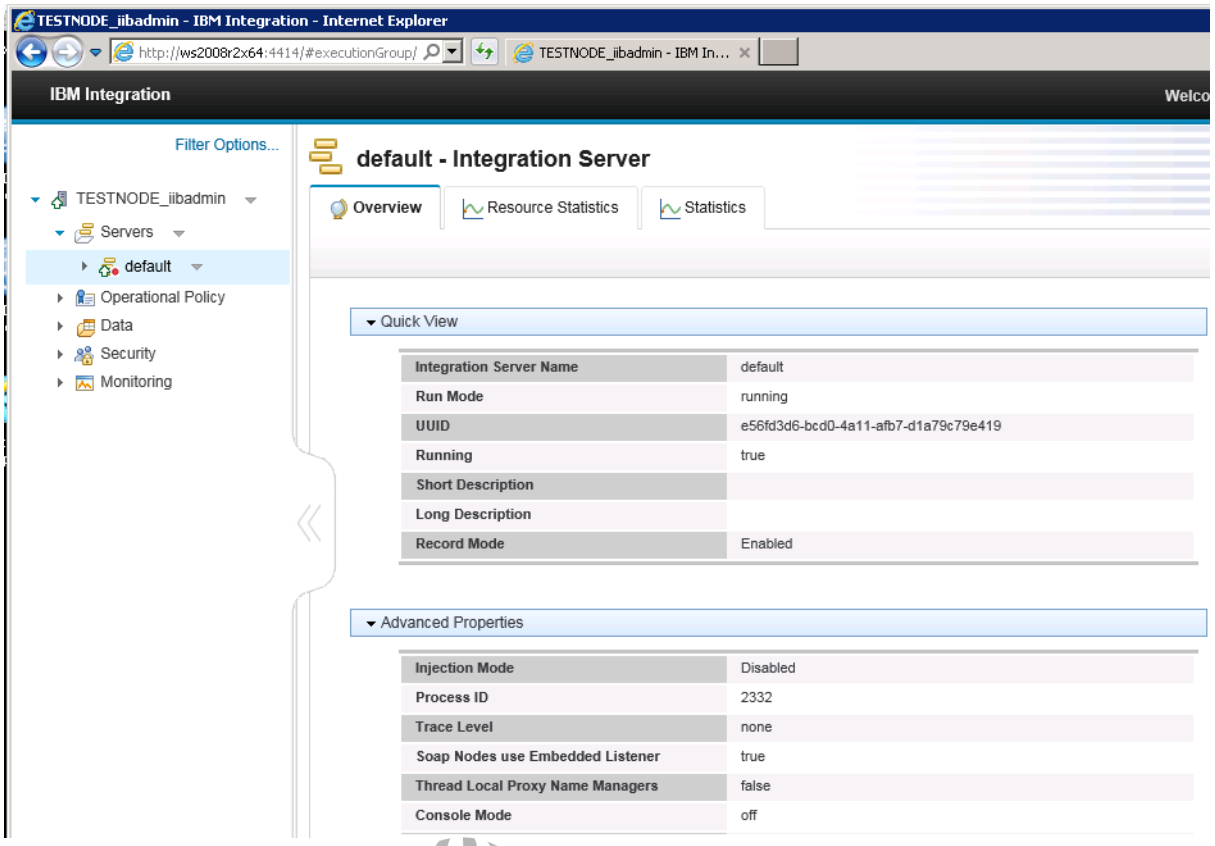
__ 2. Expand **Servers** to display the **default** integration server.

__ 3. This integration node currently has one integration server that is named **default**.

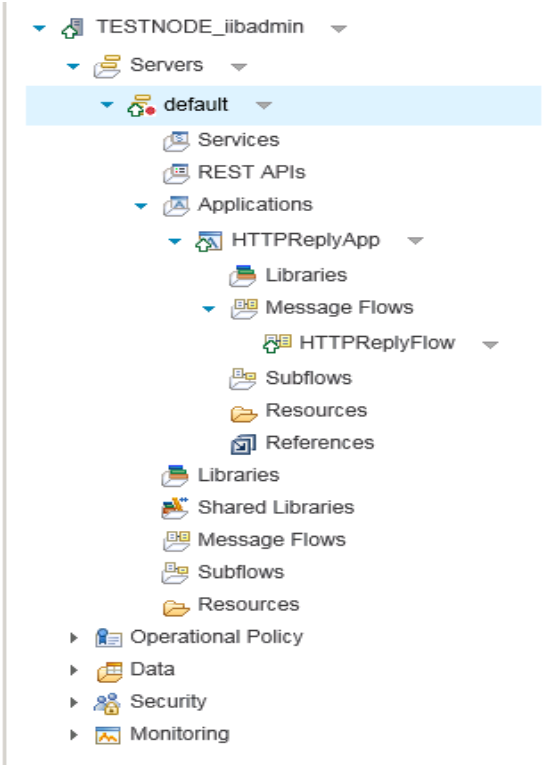
The icons next to the integration server indicate that it is running (green up arrow) and that the Flow exerciser "Record" mode is enabled (red circle). "Record" mode records and saves the test messages so that you can use them to test the message flow again, or modify them to create a test message.



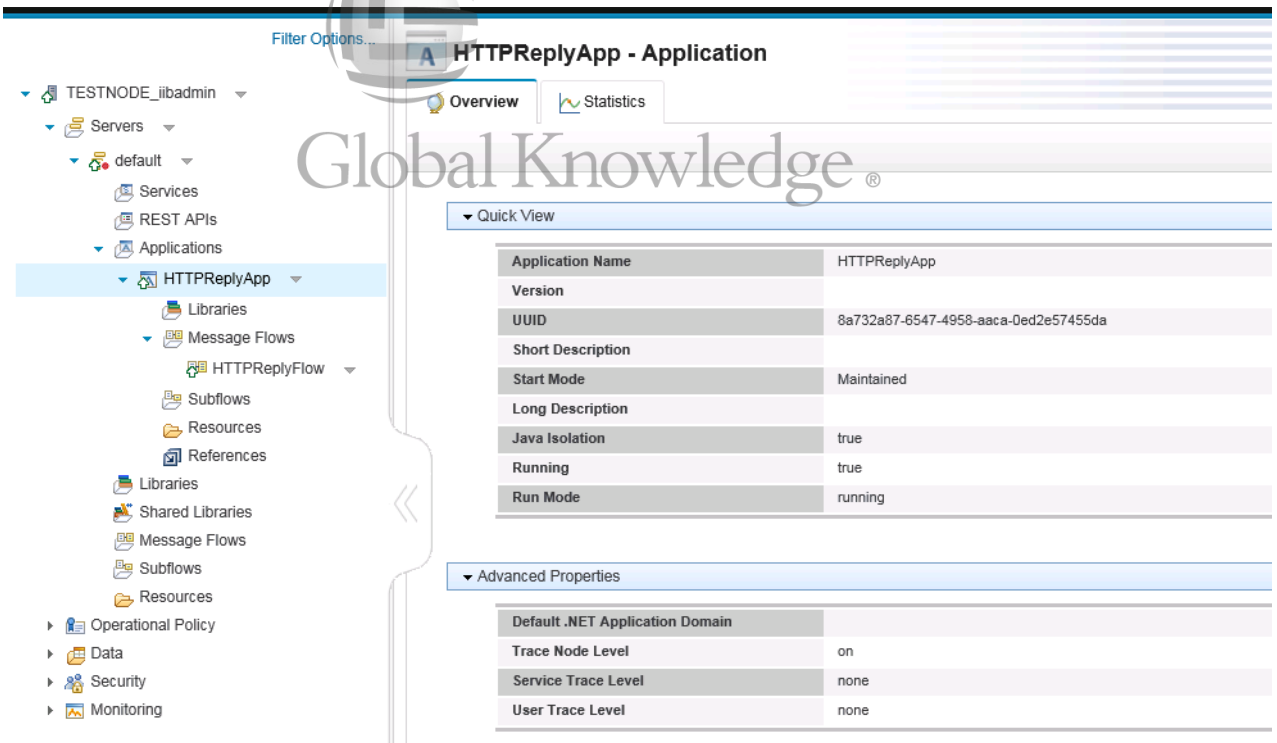
__ 4. Click the **default** integration server to display the integration server status.



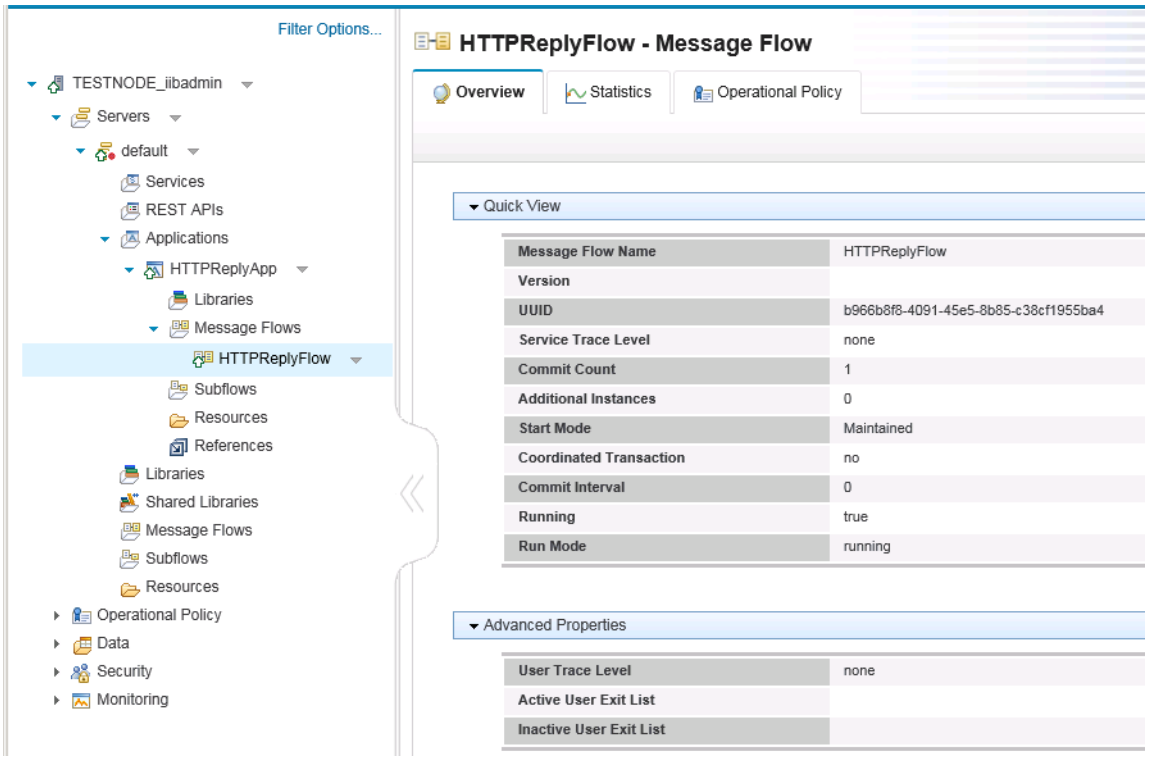
__ 5. Expand all the folders under the **default** integration server to show the message flows and other resources that are running on the integration server.



__ 6. Under **Applications**, click **HTTPReplyApp** to show the application status.



__ 7. Under **Message Flows**, click **HTTPReplyFlow** to show the message flow status.



Part 4: Exercise clean-up

__ 1. In the Integration Toolkit, stop the Flow exerciser recording mode on the **default** integration server.

In the **Integration Nodes** view, right-click the **default** integration server and then click **Stop Recording**.

__ 2. In the Integration Toolkit, close the Message flow editor.

__ 3. To avoid any potential conflicts with the next exercise, remove the **HTTPReplyApp** application from the **default** integration server.

In the **Integration Nodes** view in the Integration Toolkit, right-click the **default** integration server and then click **Delete > All Flows and Resources**.

__ 4. Close the IBM Integration web user interface.

__ 5. Close Windows Notepad.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you created a simple HTTP flow that acted as a simple web service that writes to a file.

In the second part of this exercise, you tested the message flow by using a web browser and the Integration Toolkit Flow exerciser.

In the third part of this exercise, you used the IBM Integration web user interface to check the status of the integration node, integration server, and message flow application.

Having completed this exercise, you should be able to:

- Create a message flow application
- Use the IBM Integration Flow exerciser to test the message flow application
- Use the IBM Integration web user interface to check the status of integration node, integration server, and message flow application at run time

Exercise 3. Connecting to IBM MQ

What this exercise is about

In this exercise, you create an IBM MQ queue manager and IBM Integration Bus integration nodes that use the same queue manager as a default queue manager. Next, you create and test a simple flow that gets a message from an input queue and puts a message to an output queue. Finally, you test that the workload is evenly distributed between the integration nodes by sharing a queue manager and putting multiple messages to the input queue.

What you should be able to do

After completing this exercise, you should be able to:

- Create an integration node that uses a default IBM MQ queue manager
- Share a default IBM MQ queue manager with multiple integration nodes
- Create a message flow that gets a message with an MQInput node and puts a message with an MQOutput node
- Edit a BAR file
- Manually deploy a BAR file
- Verify that the integration nodes that share a queue manager also share the workload

Introduction

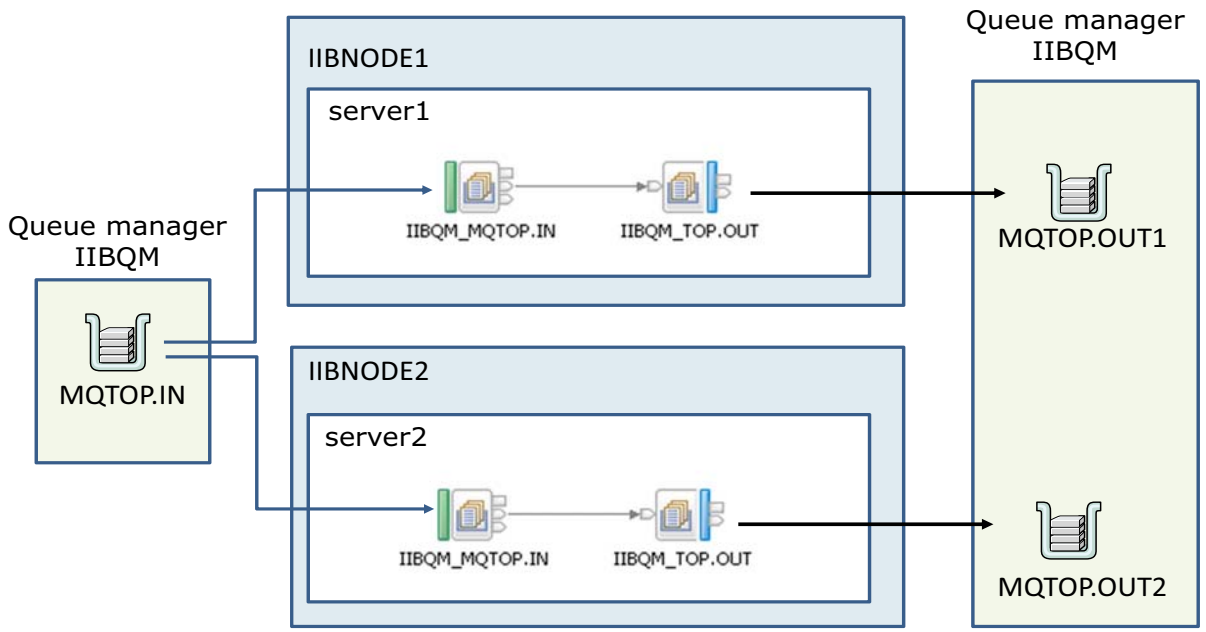
IBM MQ is no longer a prerequisite for using IBM Integration Bus V10. You can develop and deploy many message flow applications with IBM Integration Bus independently of IBM MQ; however, some IBM Integration Bus capabilities still require access to an IBM MQ queue manager. For example, the MQInput and MQOutput nodes in a message flow require access to an IBM MQ queue manager.

In IBM Integration Bus V10, multiple integration nodes can share the queue manager. Flows that are deployed to integration servers in integration nodes, which are sharing a queue manager can put and get from different queues or the same queues.

Using a shared queue manager can easily provide high availability for an application. If one of the flows, integration server, or integration node fails, the other flow on the other integration node can assume the workload.

In this exercise, you deploy message flows on two different integration nodes so that they share an input queue on the same queue manager. This

configuration provides workload distribution capabilities between integration nodes.



In the first part of this exercise, you create a queue manager that is named IIBQM.and the queues that are used in this exercise: MQTOP.IN, MQTOP.OUT1, and MQTOP.OUT2.

In the second part of this exercise, you create two integration nodes (IIBNODE1 and IIBNODE2) that use the same queue manager (IIBQM) as a default queue manager. You also create an integration server on each integration node.

In the third part of this exercise, you create the message flow that gets messages from the MQTOP.IN queue by using an MQInput node and puts them to the MQTOP.OUT1 queue by using an MQOutput node.

Information

The MQInput and MQOutput nodes require that an IBM MQ Client or Server is installed on the same computer as the integration node. They do not require a queue manager to be specified on the integration node unless you want to use this queue manager by default for the local IBM MQ connection. The integration nodes that you create in this exercise use a default queue manager.

In the fourth part of the exercise, you test the message flow with IIBNODE1 by putting some messages on the input queue by using IBM MQ Explorer and the IBM MQ `amqsput` sample program. You use IBM MQ Explorer to verify the messages on the output queue MQTOP.OUT1.

There are 15 messages in the `data.txt` input file. The messages in the file are numbered so that you can verify that all 15 messages were processed successfully.

```
01 Messages to generate queue data to use with amqsput
02 Messages to generate queue data to use with amqsput
03 Messages to generate queue data to use with amqsput
04 Messages to generate queue data to use with amqsput
05 Messages to generate queue data to use with amqsput
06 Messages to generate queue data to use with amqsput
07 Messages to generate queue data to use with amqsput
08 Messages to generate queue data to use with amqsput
09 Messages to generate queue data to use with amqsput
10 Messages to generate queue data to use with amqsput
11 Messages to generate queue data to use with amqsput
12 Messages to generate queue data to use with amqsput
13 Messages to generate queue data to use with amqsput
14 Messages to generate queue data to use with amqsput
15 Messages to generate queue data to use with amqsput
```

In the fifth part of the exercise, you modify the BAR file to put messages to MQTOP.OUT2 and deploy it to IIBNODE2. You then use the IBM MQ `amqsput` sample program to put multiple messages to the input queue so that you can verify workload distribution. You see that IBNODE1 and IBNODE2 each get messages from the MQTOP.IN queue.

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit and IBM MQ V8
- Lab files in the `C:\labfiles\Lab03-MQ` directory
- The user `iibadmin` is a member of the “mqm” group

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts between any existing message flow applications and the application that you create in this exercise, switch to a new workspace that is named `C:\Workspace\Lab03`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab03`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. To avoid excessive memory use, stop any integration nodes that are running.

In the IBM Integration Toolkit, right-click the integration node in the **Integration Nodes** view and then click **Stop**.

Part 1: Use IBM MQ Explorer to create the queue manager and queues

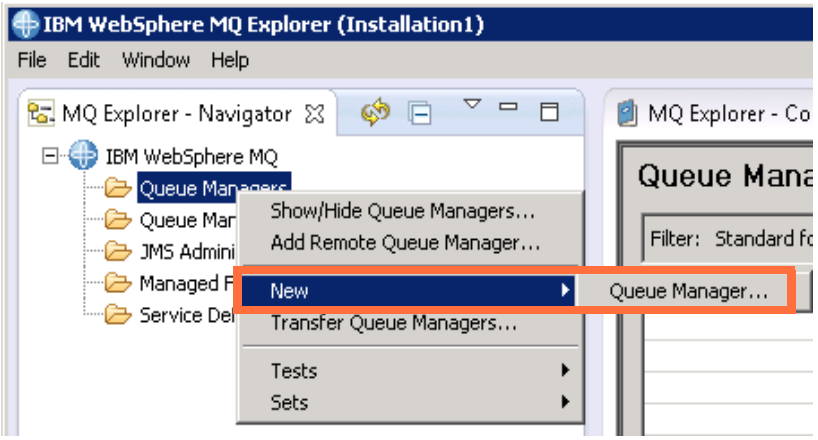
In this part of the exercise, you use IBM MQ Explorer to create a queue manager that is named IIBQM and the queues that are used in this exercise: DLQ, MQTOP.IN, MQTOP.OUT1, and MQTOP.OUT2.

__ 1. Start IBM MQ Explorer.

From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

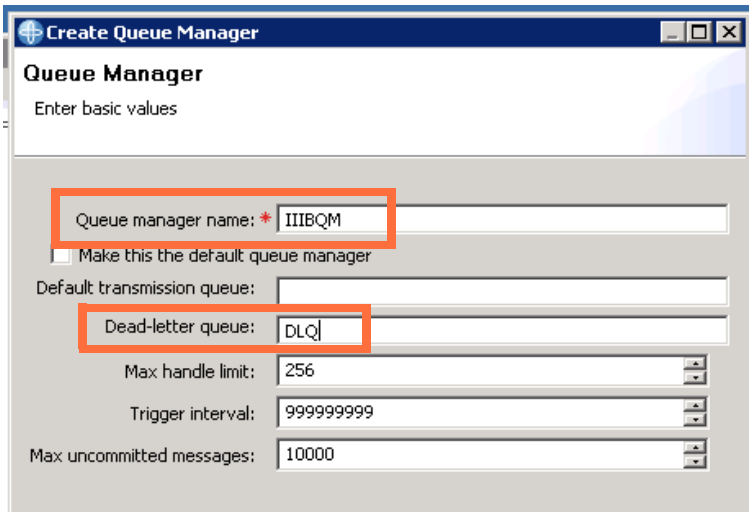
__ 2. Create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ`.

__ a. In the **MQ Explorer Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ b. For the **Queue Manager** name, type: **`IIBQM`**

__ c. For the **Dead-letter queue**, type: **DLQ**



__ d. Click **Finish**.

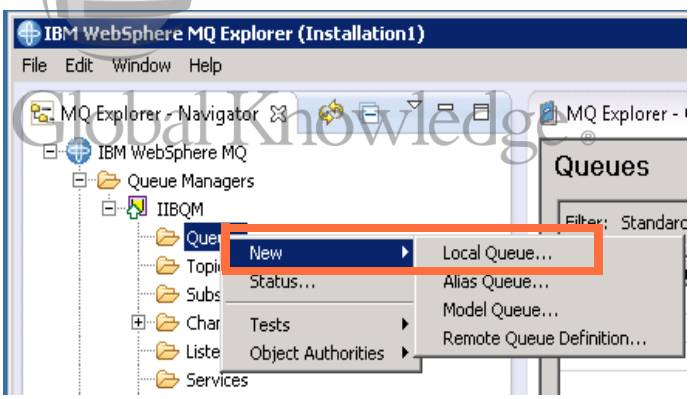
After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

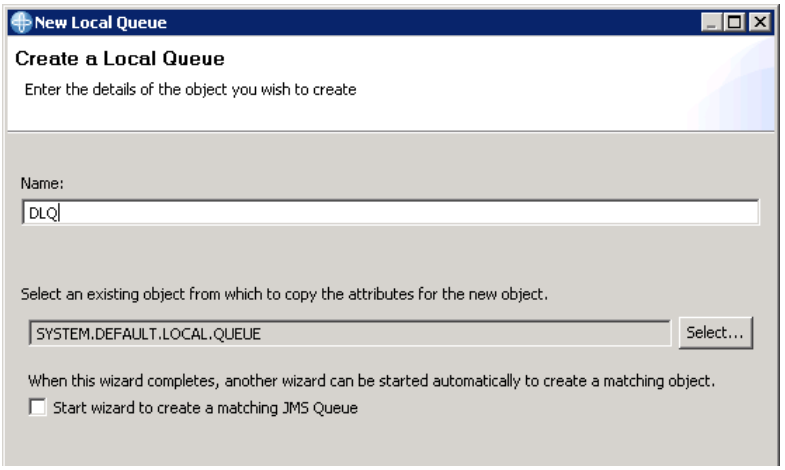
__ 3. Create a queue that is named DLQ for the dead-letter queue.

__ a. In the **MQ Explorer - Navigator** view, expand the **IIBQM** queue manager folder.

__ b. Right-click **Queues** and then click **New > Local Queue**.



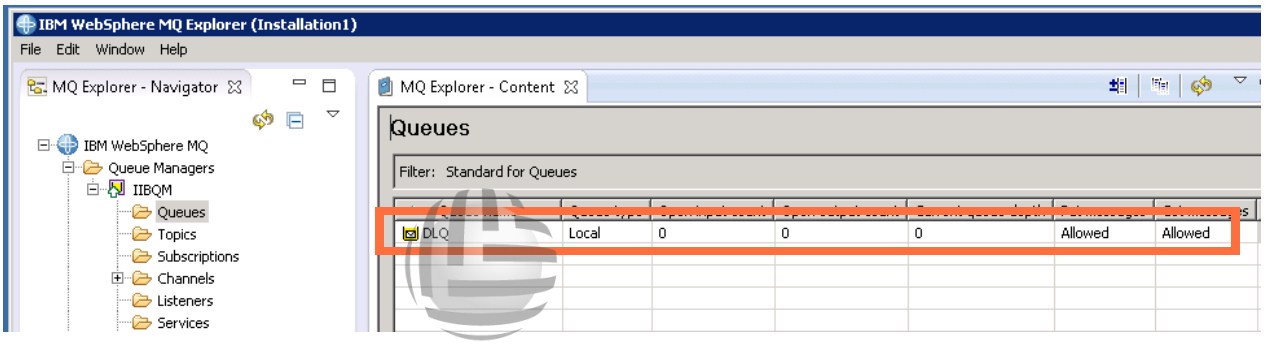
__ c. For the queue **Name**, type: **DLQ**



__ d. Click **Finish**.

__ e. Click **OK** on the Confirmation window.

The queue **DLQ** is listed in the **Queues** content view.

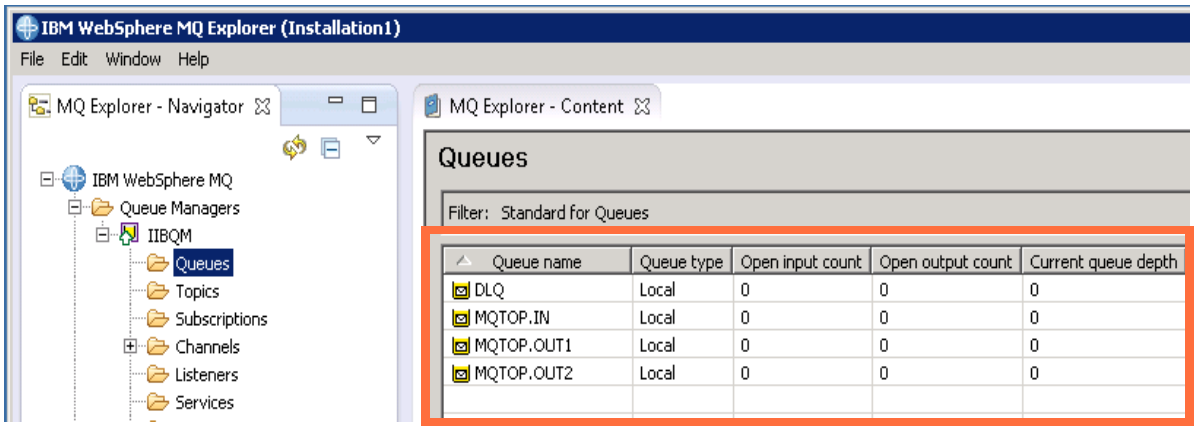


__ 4. Following the same procedure that you used to create the queue in Step 3, create the IBM MQ input queue and output queues that are used in this exercise.

__ a. Create a local queue that is named: `MQTOP.IN`

__ b. Create a local queue that is named: `MQTOP.OUT1`

__ c. Create a local queue that is named: `MQTOP.OUT2`



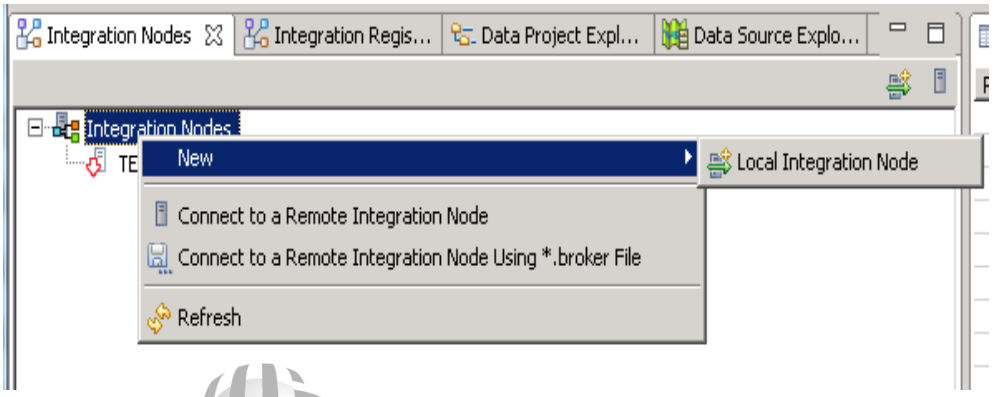
__ 5. Minimize IBM MQ Explorer. You use it later to view test the message and view messages on the queues.

Part 2: Use the IBM Integration Toolkit to create the integration nodes and integration servers

In this part of the exercise, you use the IBM Integration Toolkit to create two integration nodes that use the same queue manager as the default queue manager. You also create an integration server on each integration node.

__ 1. Create an integration node that is named `IIBNODE1` with a default integration server that is named `server1`.

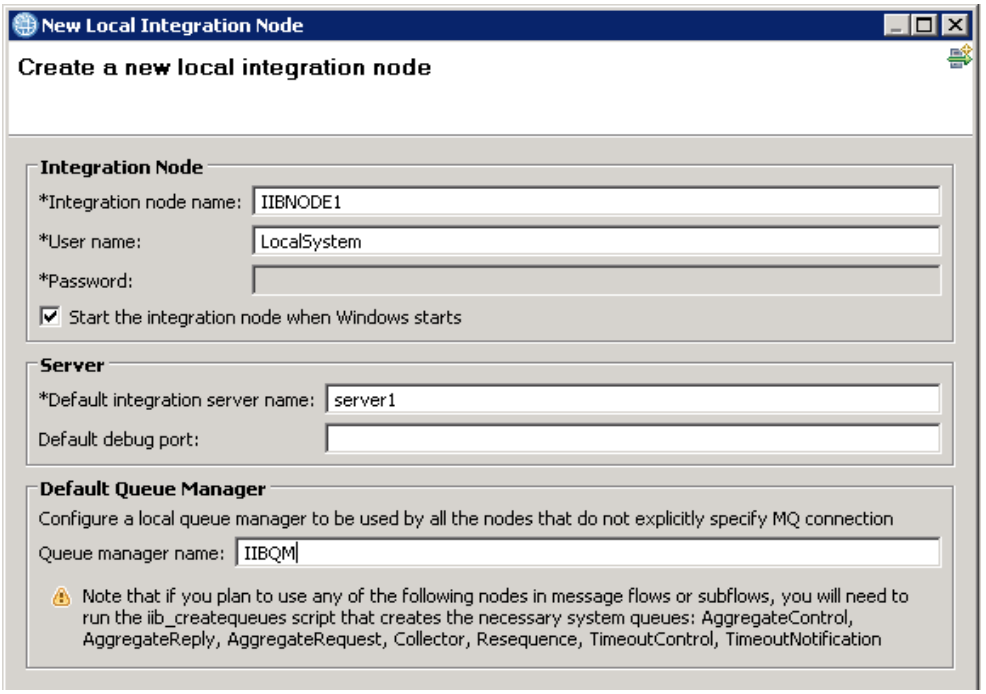
__ a. In the **Integration Nodes** view in the IBM Integration Toolkit, right-click **Integration Nodes** and then click **New > Local Integration Node**.



__ b. For the **Integration node name**, type: **`IIBNODE1`**

__ c. For the **Default integration server name**, type: **server1**

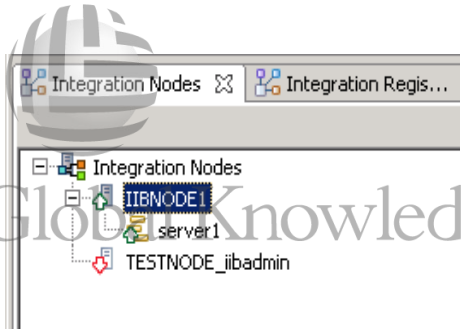
__ d. For the **Queue manager name**, type: **IIBQM**



__ e. Click **Finish**.

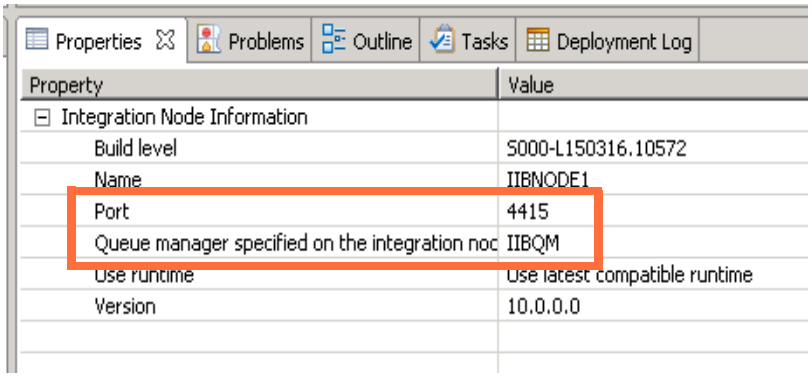
__ f. Click **Close** on the confirmation window.

The new integration node and integration server should be shown in the **Integration Nodes** view.



__ 2. Verify the integration node properties.

With the IIBNODE1 integration node selected in the **Integration Nodes** view, click the **Properties** view tab.



Verify that the integration node properties indicate that the queue manager IIBQM is specified on the integration node.

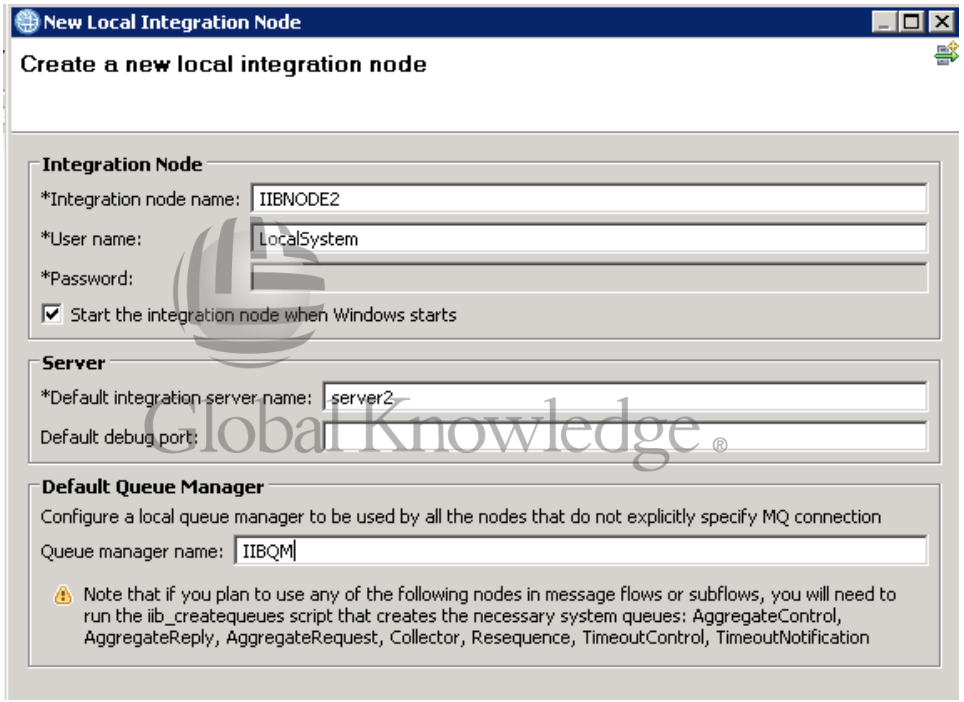
Also, note the port number for accessing the IBM Integration web user interface. For IIBNODE1, the port for accessing the IBM Integration web user interface is 4415.

Information

IBM Integration Bus assigns a web user administration port number dynamically. When the first integration node is started, it is assigned the default port of 4414. For subsequent integration nodes, the port number increments and the node is assigned the next port number in sequence.

__ 3. Following the same procedure that you used for creating the integration node **IIBNODE1** and the integration server **server1**, create another integration node that is named **IIBNODE2** and an integration server of **server2**.

Be sure to specify the same queue manager, **IIBQM**, as the default queue manager.



__ 4. Verify that the integration node properties for **IIBNODE2** indicate that queue manager **IIBQM** is specified on the integration node.

Also, that the port number for accessing the IBM Integration web user interface is 4416.

Part 3: Create the message flow

In this part of the exercise, you create the message flow that gets messages from the MQTOP.IN queue by using an MQInput node and puts them to the MQTOP.OUT1 queue by using an MQOutput node.

__ 1. In the IBM Integration Toolkit, create an application that is named `MQTOP`.

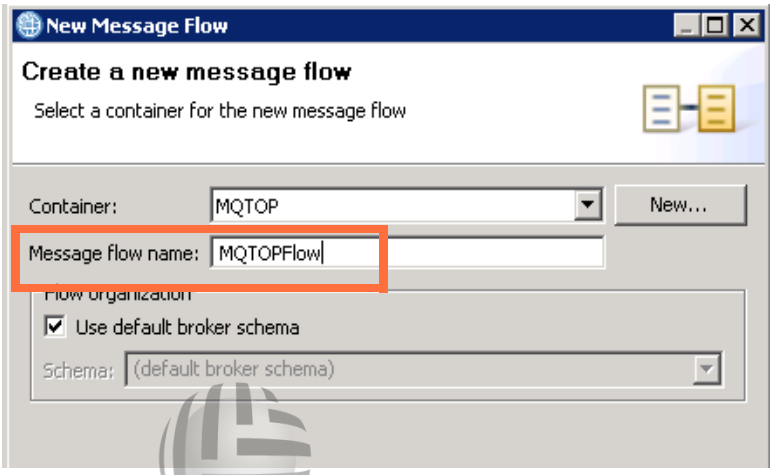
__ a. In the Application Development view, click **New Application**.

__ b. For the application name, type: `MQTOP`

__ 2. In the MQTOP application, create a message flow that is named `MQTOPFlow`.

__ a. In the Application Development view, click **New** and then click **MessageFlow**.

__ b. For the Message Flow name, type: **`MQTOPFlow`**

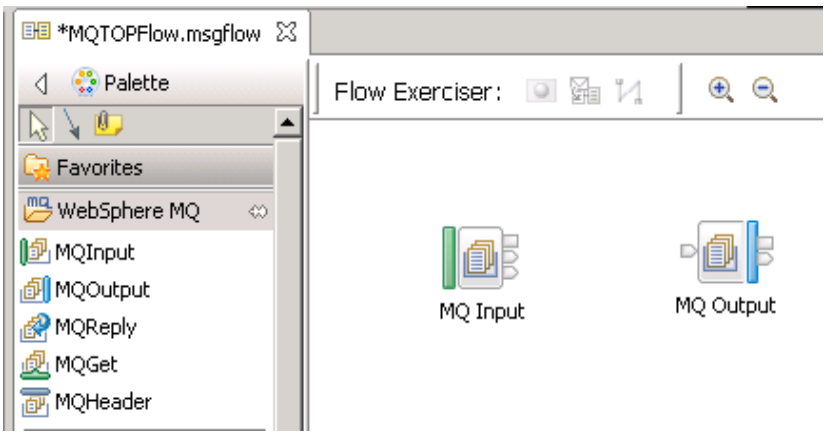


__ 3. Add the nodes to the Message Flow editor canvas.

__ a. In the Message Flow editor Palette, expand the **WebSphere MQ** drawer.

__ b. Drag an MQInput node to the Message Flow editor canvas.

__ c. Drag an MQOutput node to the Message Flow editor canvas.



__ 4. Wire the **Out** terminal on the MQ Input node to the **In** terminal on the MQ Output node.

__ 5. Configure the node properties for MQ Input node.

__ a. Modify the display name of the MQ Input node. On the **Description** properties tab, type: **IIBQM_MQTOP.IN**

__ b. Identify the input queue. On the **Basic** properties tab, for the **Queue name**, type: **MQTOP.IN**

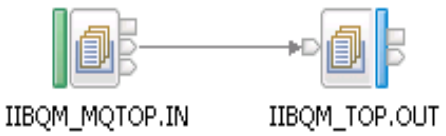
__ c. On **MQ Connection** properties tab, verify that **Connection** is set to **Local queue manager**.

__ 6. Configure the node properties for the MQ Output node.

__ a. Modify the display name of the MQ Output node. On the **Description** properties tab, type: **IIBQM_MQTOP.OUT**

__ b. Identify the output queue. On the **Basic** properties tab, for the **Queue name**, type: **MQTOP.OUT1**

__ c. On **MQ Connection** properties tab, verify that **Connection** is set to **Local queue manager**.



__ 7. Save the flow.

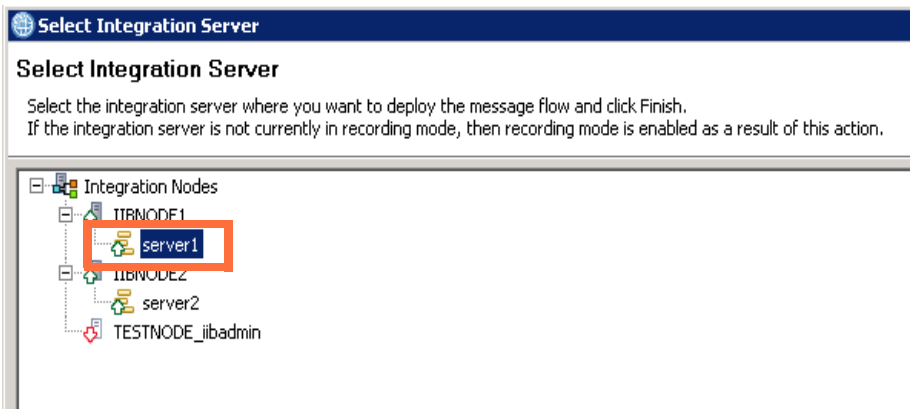
__ 8. Check the **Problems** view and verify that there are no errors in the message flow.

Part 4: Test the message flow

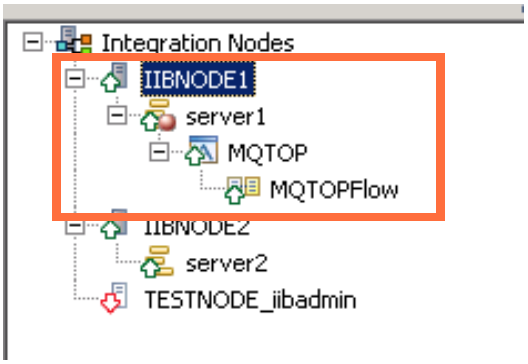
In this part of the exercise, you test the message flow with IIBNODE1. You put messages on the queue by using the IBM MQ Explorer and then by using the IBM MQ `amqsput` sample program. You use IBM MQ Explorer to verify the messages on the output queue.

__ 1. In the Message Flow editor, click the **Start Flow exerciser** icon to deploy the message flow.

__ 2. In the **Select Integration Server** window, click **server1** under IIBNODE1.

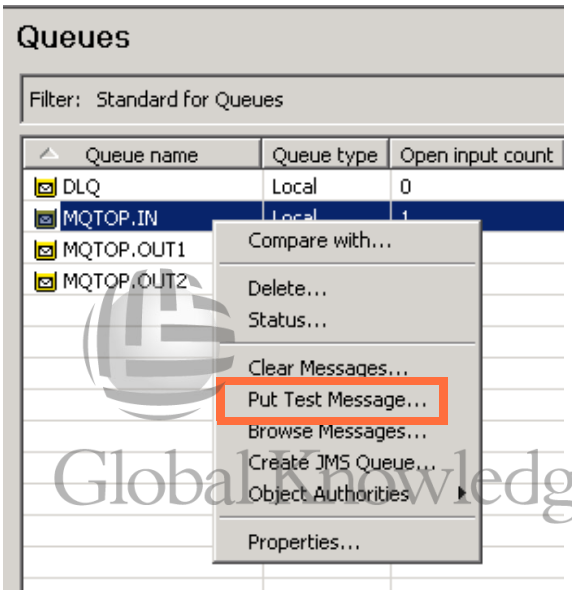


__ 3. Verify that the message flow is deployed to **server1** on IIBNODE1 in the **Integration Nodes** view.

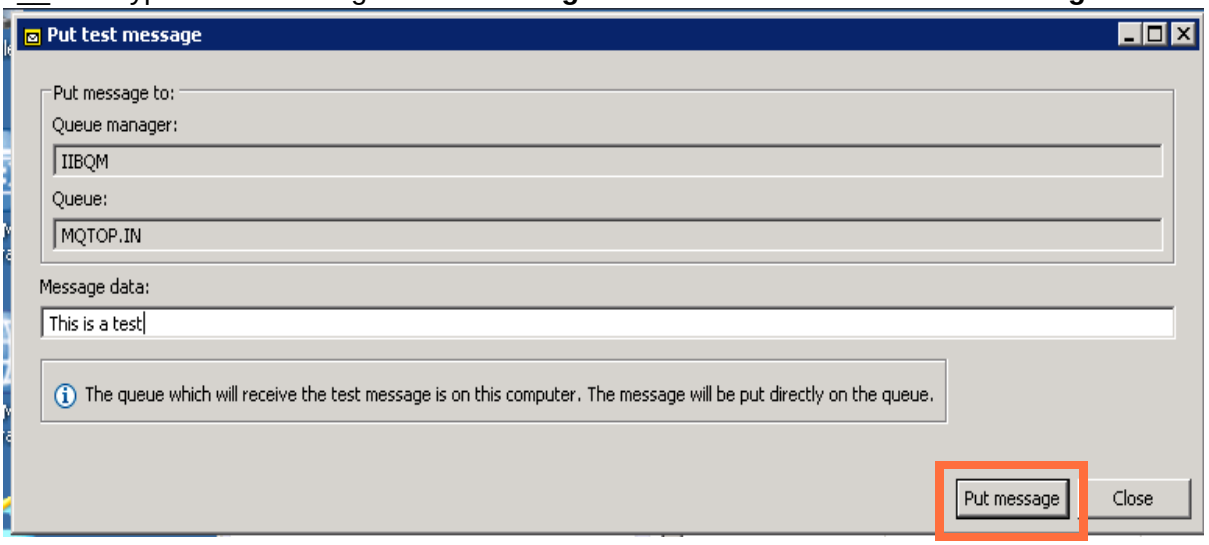


__ 4. Put a message on the input queue by using IBM MQ Explorer.

__ a. In **Queues** content view, right-click the **MQTOP.IN** queue and then click **Put Test Message**.

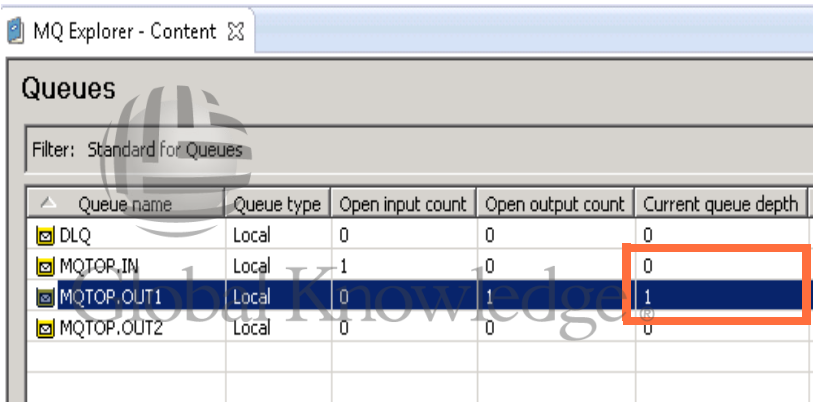


__ b. Type a test message in the **Message data** field and then click **Put message**.

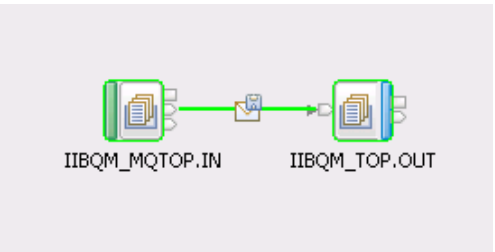


__ 5. In the IBM MQ Explorer **Queues** view, you should see the **Current queue depth** for the input queue MQTOP.IN increment to 1.

__ 6. After a brief pause, you should see **Current queue depth** on MQTOP.IN decrement and return to zero. You should also see the **Current queue depth** on the output queue MQTOP.OUT1 increment to 1.



__ 7. To verify that the message flow processed the message, return to the IBM Integration Toolkit and click the Flow exerciser **View Path** icon.



__ 8. Stop the Flow exerciser **Recording** mode on the server1 integration server on IIBNODE1.

Right-click the **server1** integration server in the **Integration Nodes** view and then click **Stop Recording**.

__ 9. Put multiple messages to the MQTOP.IN queue by using the IBM MQ `amqsput` sample program and the `data.txt` file in the `C:\labfiles\Lab03-MQ` directory.

__ a. Open a Command Prompt window.

__ b. Type: **`amqsput MQTOP.IN IIBQM < C:\labfiles\Lab03-MQ\data.txt`**

__ 10. The messages in the file are numbered so that you can verify that all 15 messages were processed successfully.

By using IBM MQ Explorer, verify that you now have 16 messages on the MQTOP.OUT1 queue (one message from Step 4 and the 15 messages from Step 9).

__ a. In the IBM MQ Explorer **Queues** view, right-click the MQTOP.OUT1 and then click **Browse Messages**.

__ b. In the **Message browser** window, scroll to the right to view the **Message data** column.

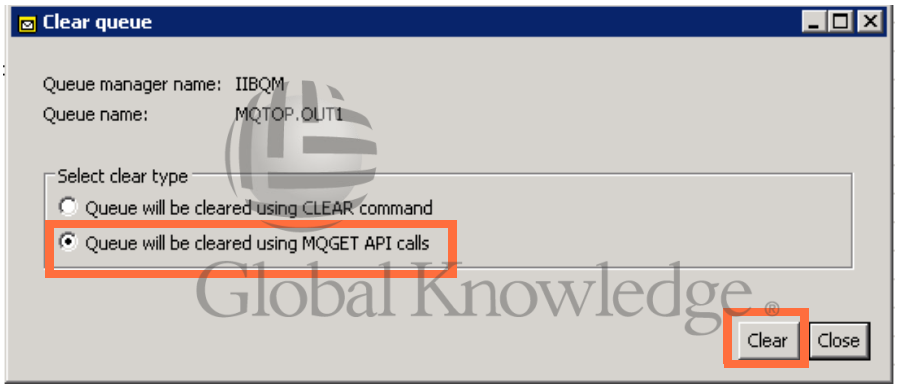
__ c. Click **Close** to close the **Message browser** window.

__ 11. Clear the messages on the MQTOP.OUT1 queue.

__ a. In the IBM MQ Explorer **Queues** view, right-click the MQTOP.OUT1 and then click **Clear Messages**.

__ b. Click the **Queue will be cleared using MQGET API calls** option.

__ c. Click **Clear**.



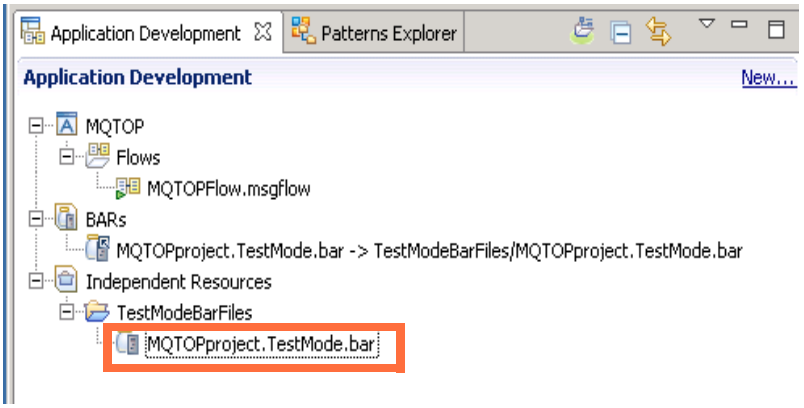
__ d. Click **OK** on the confirmation window.

Part 5: Modify the BAR file and manually deploy the message flow

In Part 4 of this exercise, the Flow exerciser created a BAR file under the **TestModeBarFiles** folder in the **Independent Resources** project and a reference to that BAR file in the **BARs** folder in the application.

In this part of the exercise, you modify the BAR file that the Flow exerciser created to put messages to MQTOP.OUT2 and deploy it to **server2** on IIBNODE2. You then use the IBM MQ `amqsput` sample program to put multiple messages to the input queue.

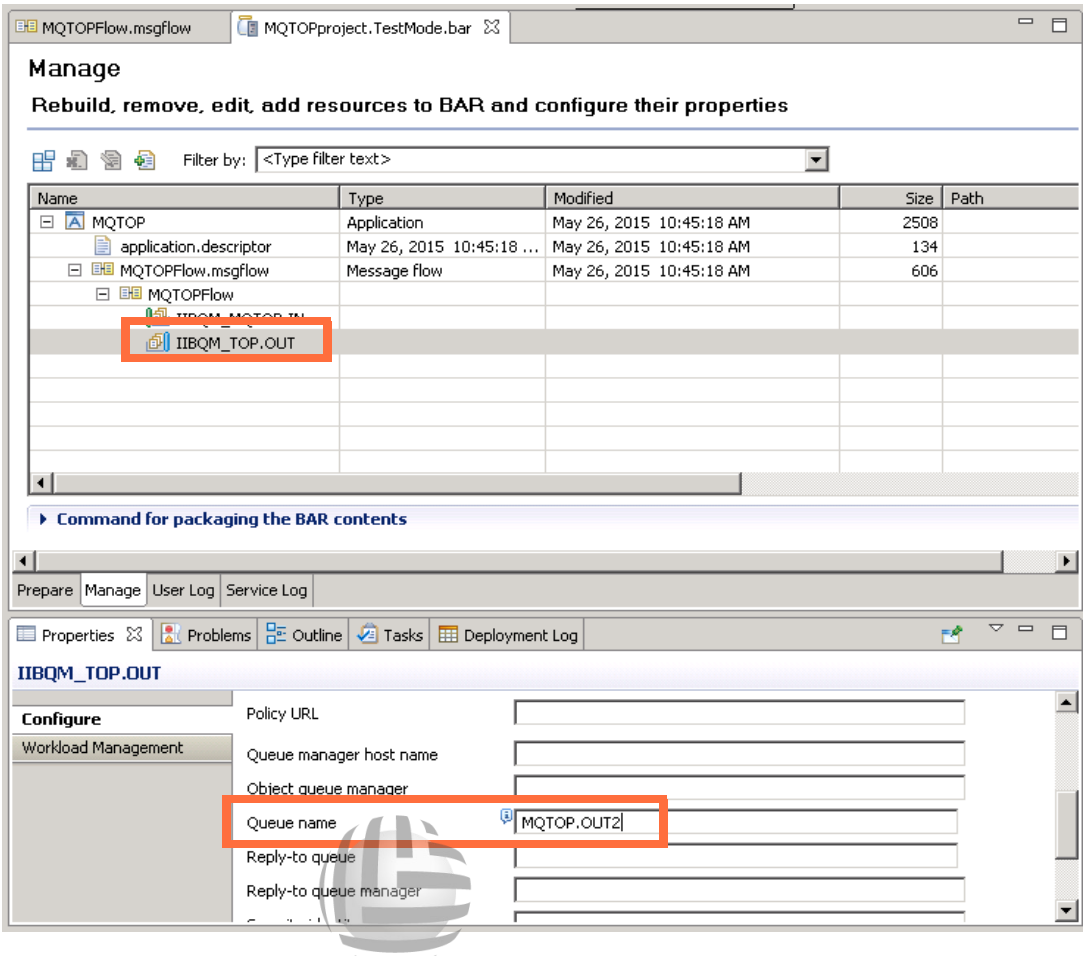
__ 1. In the IBM Integration Toolkit Application Development view, double-click the **MQTOPProject.TestMode.bar** file under **Independent Resources > TestModeBarFiles** to open it in the BAR File editor.



__ 2. In the BAR File editor, expand the **MQTOP** application and the **MQTOPFlow.msgflow** until you see the message flow nodes.

__ 3. Select the **IIBQM_TOP.OUT** node in the BAR File editor to show the node **Properties** view.

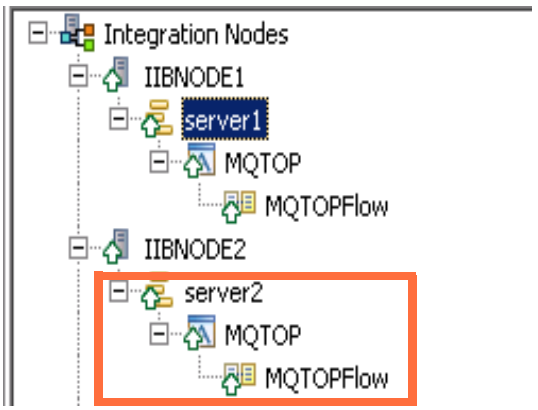
__ 4. In the **Properties** view for the IIBQM_TOP.OUT node, change the **Queue name** property to **MQTOP.OUT2**.



__ 5. Save the BAR file.

__ 6. Deploy the updated BAR file to **server2** on IIBNODE2 by dragging it from the Application Development view and dropping it onto **server2** in the **Integration Nodes** view.

__ 7. Verify that the BAR file was deployed successfully by checking the **Deployment Log** view and the **Integration Nodes** view.



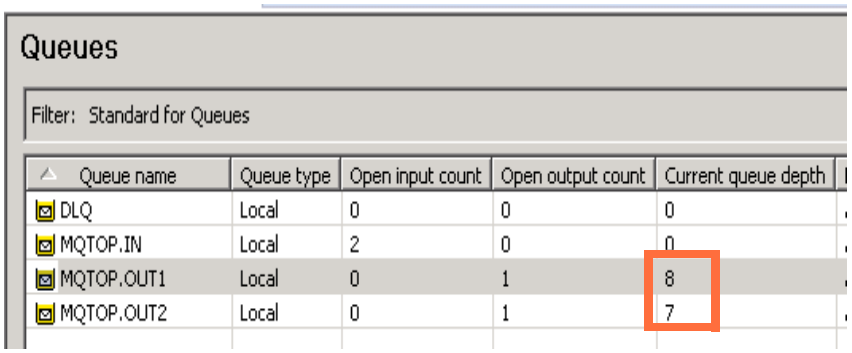
__ 8. Put multiple messages to the MQTOP.IN queue by using the IBM MQ `amqsput` sample program and the `data.txt` file in the `C:\labfiles\Lab03-MQ` directory.

In a Command Prompt window, type:

```
amqsput MQTOP.IN IIBQM < C:\labfiles\Lab03-MQ\data.txt
```

__ 9. Verify that the workload is shared between the integration nodes by checking the queue depths for the MQTOP.OUT1 queue and MQTOP.OUT2 queue.

In the IBM MQ Explorer **Queues** view, you should see that messages appear on both queues.



Note

The distribution of the messages on the queues might not exactly match the screen capture that is shown here as an example.

__ 10. The messages in the file are numbered so that you can verify that all 15 messages were processed successfully.

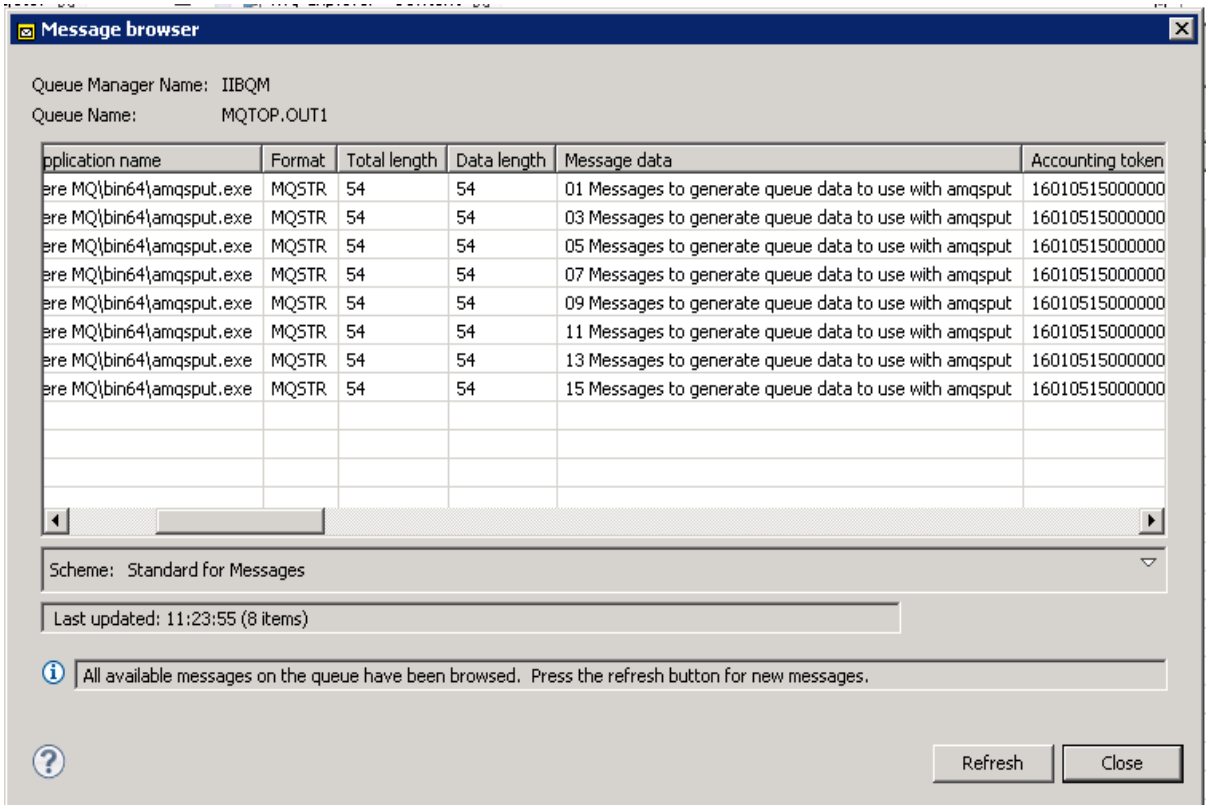
__ a. In the IBM MQ Explorer **Queues** view, right-click the MQTOP.OUT1 and then click **Browse Messages**.

__ b. In the “Message browser” window, scroll to the right to view the **Message data** column.

You should see that both queue managers got messages from the same input queue and that no messages are duplicated on the output queues. Look at the first two characters of the **Message data** column in the IBM MQ Explorer **Message browser** view for each output queue to verify that the messages on the input queue were shared between the two queue managers. The screen capture provided here is an example of the **Message browser** window for MQTOP.OUT1.

Note

The distribution of messages that you see in your lab environment might not exactly match the screen capture.



Information

You should observe that multiple message flows that share a queue manager provide workload distribution. If one of the flows, integration server, or integration node fails, the other flow on the other integration node can assume the workload.

Exercise clean-up

__ 1. In the IBM Integration Toolkit **Integration Nodes** view, delete all flows and resources from **server1** integration server on IIBNODE1.

__ 2. In the **Integration Nodes** view, stop IIBNODE1 by right-clicking it and then clicking **Stop**.

__ 3. In the **Integration Nodes** view, delete all flows and resources from **server2** integration server on IIBNODE2.

__ 4. In the **Integration Nodes** view, stop IIBNODE2 by right-clicking it and then clicking **Stop**.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Create an integration node that uses a default IBM MQ queue manager
- Share a default IBM MQ queue manager with multiple integration nodes
- Create a message flow that gets a message with an MQInput node and puts a message with an MQOutput node
- Edit a BAR file
- Manually deploy a BAR file
- Verify that the integration nodes that share a queue manager also share the workload

Exercise 4. Adding flow control to a message flow application

What this exercise is about

In this exercise, you add a Route node to a message flow to route the message based on the message content. You also wire the Failure terminal to handle exceptions.

What you should be able to do

After completing this exercise, you should be able to:

- Use the Route node to control message processing
- Use the XPath Expression Builder to define a filter pattern
- Create custom output terminals on the Route node
- Connect a Failure terminal to an output node to capture exceptions
- Test the message flow by importing messages into the IBM Integration Toolkit Flow exerciser

Introduction

In this exercise, you create a message flow that processes XML files that contain information about a store. The XML file contains a store number, stock ID for a product, and the last date that the product is available. A sample XML file is shown here.


```
<?xml version="1.0" encoding="UTF-8"?>
   <inputrec>
      <store-number>201</store-number>
      <stock-id>shower gell</stock-id>
      <weekend-date>24/02/2015</weekend-date>
   </inputrec>
```

The XML schema that is named StoreProducts.xsd defines the XML file.

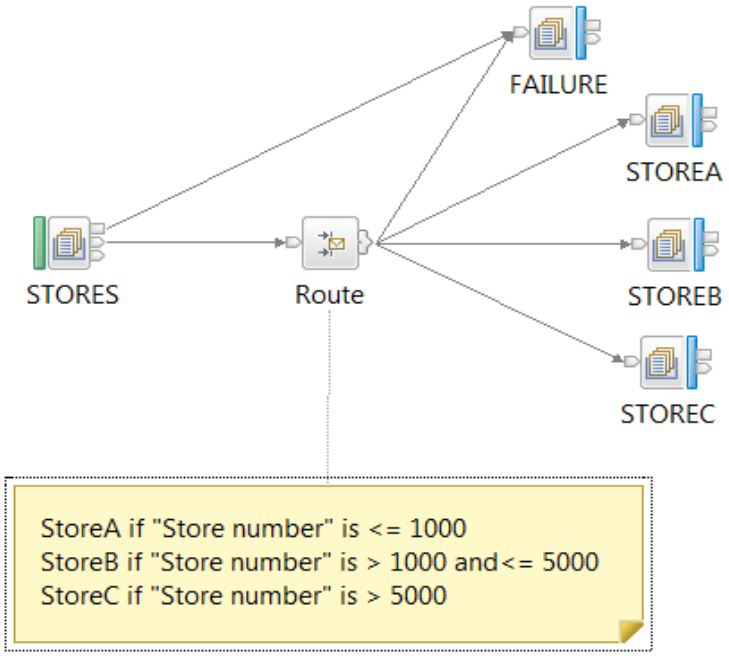
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="stock-id" type="xsd:string"/>
  <xsd:element name="store-number" type="xsd:string"/>
  <xsd:element name="inputrec">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="store-number"/>
        <xsd:element ref="stock-id"/>
        <xsd:element ref="weekend-date"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="weekend-date" type="xsd:string"/>
</xsd:schema>
```

In the message flow, you add a Route node to a message flow so that messages are routed to a specific IBM MQ queue based on the store number in the message.

By default, the Route node contains a single output terminal for any matches. In this exercise, you define explicit output terminals on the Route node for more control of the message flow.

- If the store-number value is less than or equal to 1000, the message should be routed out an output terminal that is named **StoresA** to a queue that is named STOREA.
- If the store-number value is greater than 1000 and less than or equal to 5000, the message should be routed out an output terminal that is named **StoresB** to a queue that is named STOREB.
- If the store-number value is greater than 5000, the message should be routed out an output terminal that is named **StoresC** to a queue that is named STOREC.
- If the message is invalid, the message should be routed to a queue that is named FAILURE.

When you reference the schema object in the Route node filter table, you must provide the objects full name by using an XPath expression. For example, the full name for store-number is $Root/XMLNSC/inputrec/store-number. As you see in this exercise, the XPath Expression Builder helps you define the name correctly.



In the first part of this exercise, you import an IBM Integration Bus project interchange file that contains the XML schema that defines the message and a partially completed message flow. You complete the message by:

- Adding the Route node to the message flow
- Defining the route filter patterns and output terminals
- Wiring the Route node output terminals

In the second part of this exercise, you deploy and test the message flow application by sending messages with different store numbers and verifying that the message is routed to the correct output queue.

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit and IBM MQ V8
- A local IBM MQ queue manager that is named IIBQM with the following local queues: DLQ, FAILURE, STOREA, STOREB, STOREC, and STORES.IN
- Lab files in the `C:\labfiles\Lab04-Routing` directory
- The user `iibadmin` is a member of the “mqm” group

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab04`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab04`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, verify that the integration node **TESTNODE_iibadmin** is started.

Start it if it is stopped.

__ 4. This exercise requires an IBM MQ queue manager.

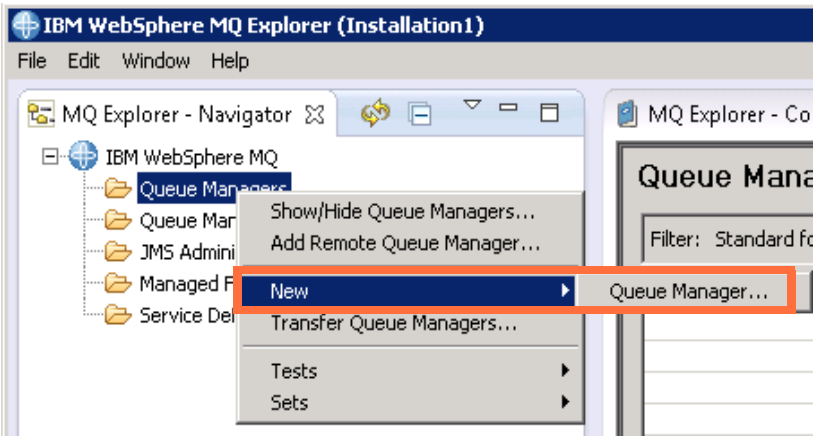
If you completed Exercise 3, you created a queue manager that is named IIBQM. You can use that queue manager in this exercise. Proceed to Step 5.

If you did not complete Exercise 3, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ` by following these instructions.

__ a. Start IBM MQ Explorer.

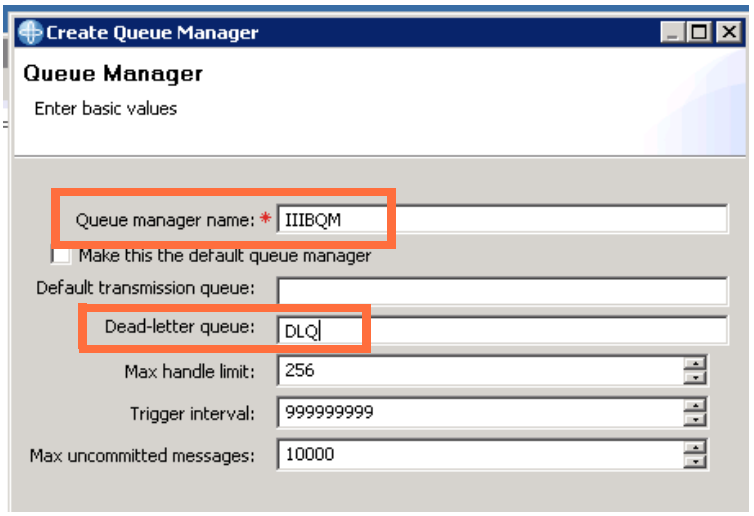
From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

__ b. In the **MQ Explorer Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ c. For the **Queue Manager** name, type: **`IIBQM`**

__ d. For the **Dead-letter queue**, type: **DLQ**



__ e. Click **Finish**.

After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

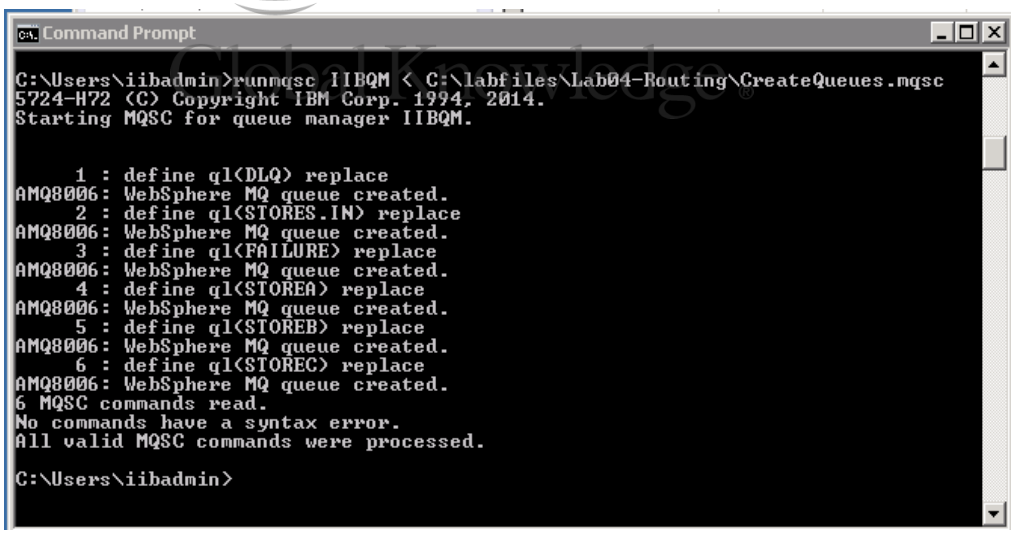
The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 5. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

__ a. Open a command prompt window.

__ b. In the command prompt window, type:

```
runmqsc IIBQM < C:\labfiles\Lab04-Routing\CreateQueues.mqsc
```

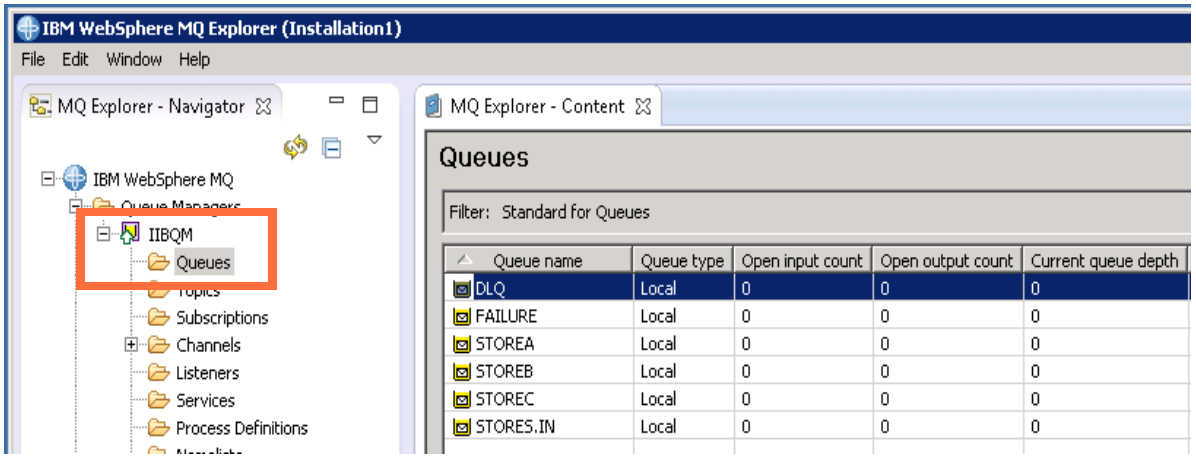


__ 6. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the IBM MQ Explorer Navigator view, expand the **IIBQM** folder.

__ b. Click **Queues** under the queue manager in the **MQ Explorer - Navigator** view to display the **Queues** view.

__ c. Verify that the following queues are listed in the **Queues** view: DLQ, FAILURE, STOREA, STOREB, STOREC, and STORES.IN



Note

You might have other queues for this queue manager from a previous exercise. You do not need to delete the unused queues.

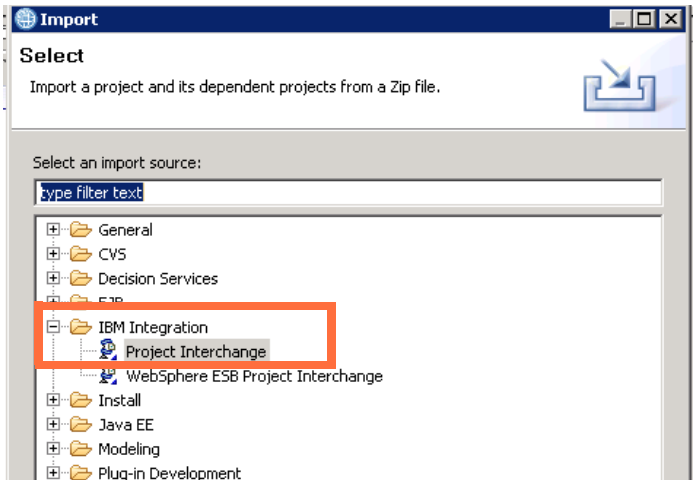
Part 1: Add routing to the message flow

In this part of this exercise, you import an IBM Integration Bus project interchange file that contains the XML schema that defines the message and a partially completed message flow.

__ 1. Import the IBM Integration Bus project interchange file that is named `RoutingApp_PI.zip` from the `C:\labfiles\Lab04-Routing` directory into the IBM Integration Toolkit

__ a. From the IBM Integration Toolkit, click **File > Import**.

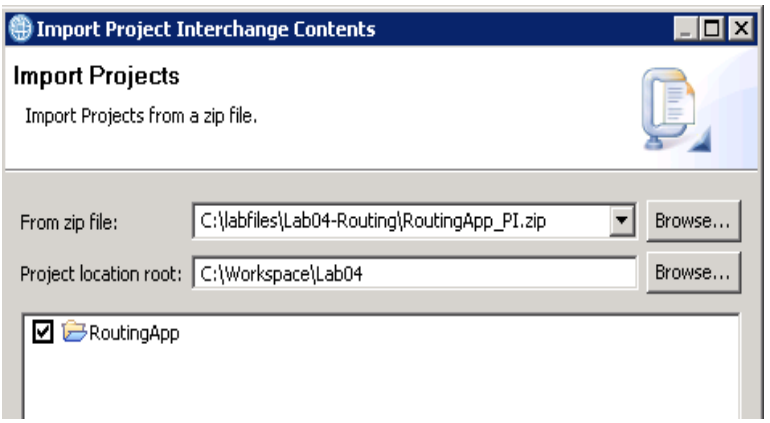
__ b. Click **Project Interchange** under **IBM Integration**, and then click **Next**.



__ c. To the right of **From zip files**, click **Browse** to locate the project interchange file.

__ d. Browse to the `C:\labfiles\Lab04-Routing` directory, select the `RoutingApp_PI` file, and then click **Open**.

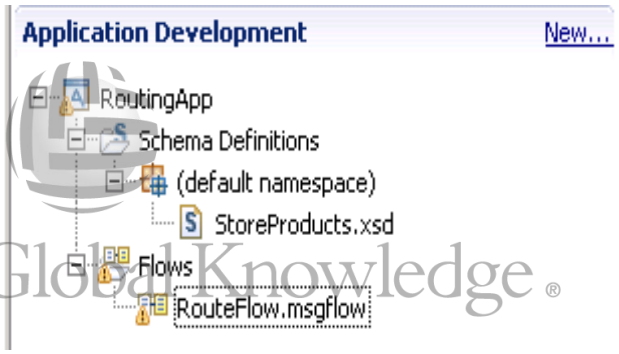
__ e. Ensure that the **Project location root** field is set to the current workspace of `C:\Workspace\Lab04`.



__ f. This project interchange file contains one application that is named **RoutingApp.** Ensure that it is selected and then click **Finish**.

__ g. Verify that the **RoutingApp** application is imported into the IBM Integration Toolkit. The application should be listed in the Application Development view.

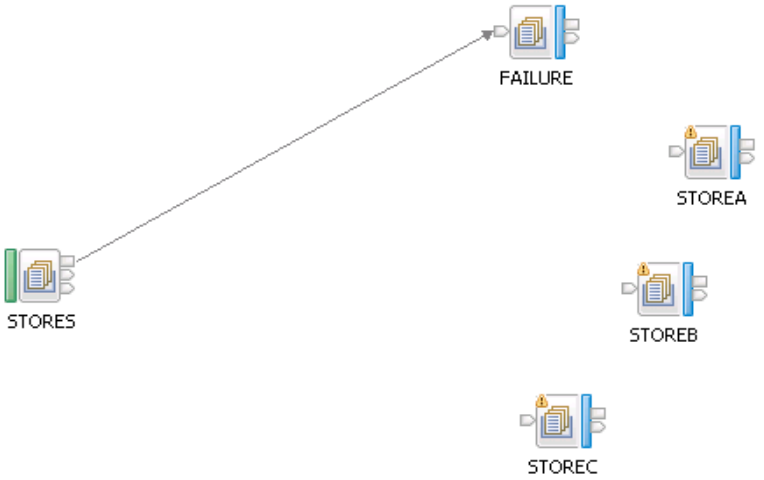
__ 2. Expand the application to view the components in the **RoutingApp** application.



The application contains the XML schema file `StoreProducts.xsd` that describes the message and a partially completed message flow that is named `RouteFlow.msgflow`.

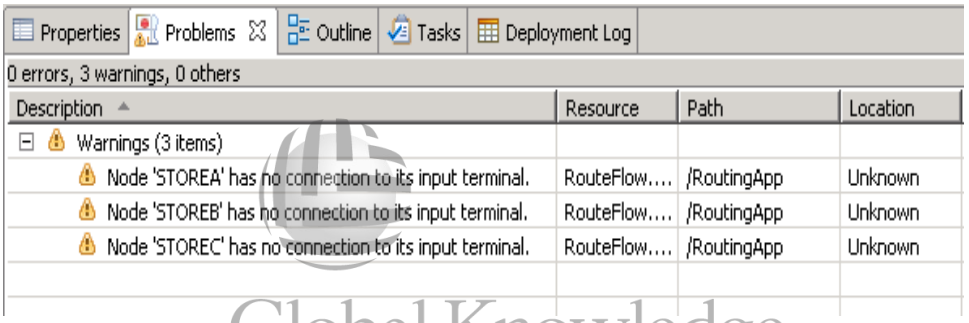
__ 3. Double-click the message flow **RouteFlow.msgflow** in the Application Development view to open it in the Message Flow editor.

The message flow contains one MQ Input node that is named STORES and four MQ Output nodes that are named FAILURE, STOREA, STOREB, and STOREC.



__ 4. As indicated by the “warning” icons on the STOREA, STOREB, and STOREC MQ Output nodes, the flow contains warnings.

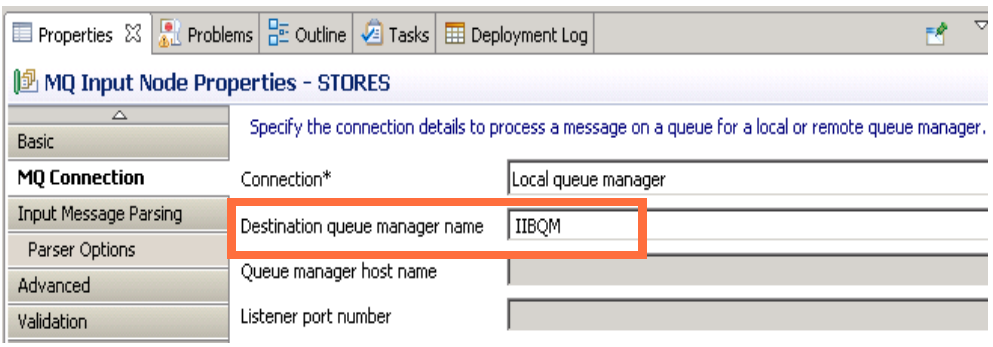
Examine the **Problems** view so that you know that why these nodes are flagged in the message flow.



__ 5. Examine the **Basic** tab and **MQ Connection** tab in the **Properties** view for each node in the flow.

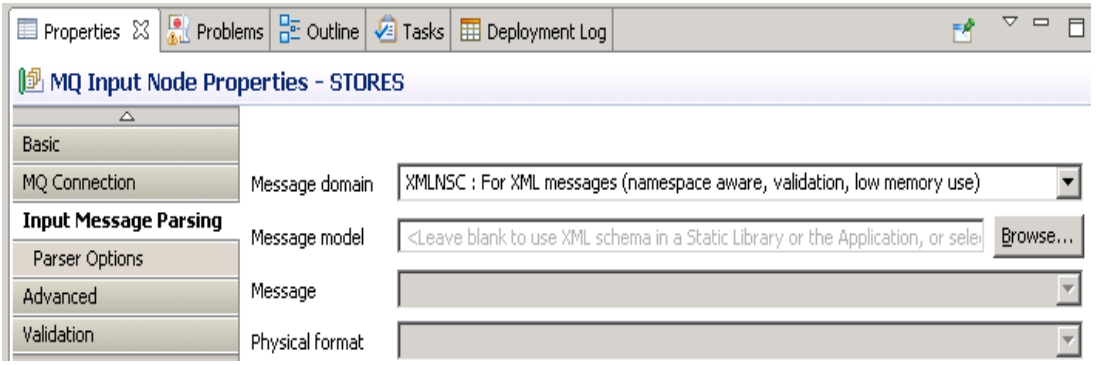
The **Basic** tab identifies the queue name that each node references.

The **MQ Connection** tab identifies the queue manager. The integration node that you use in this exercise, **TESTNODE_iibadmin**, does not have a specified default queue manager so the **MQ Connection** tab must specify IIBQM as the **Destination queue manager name**.



__ 6. Examine the **Input Message Parsing** tab in the **Properties** view for the MQ Input node that is named STORES.

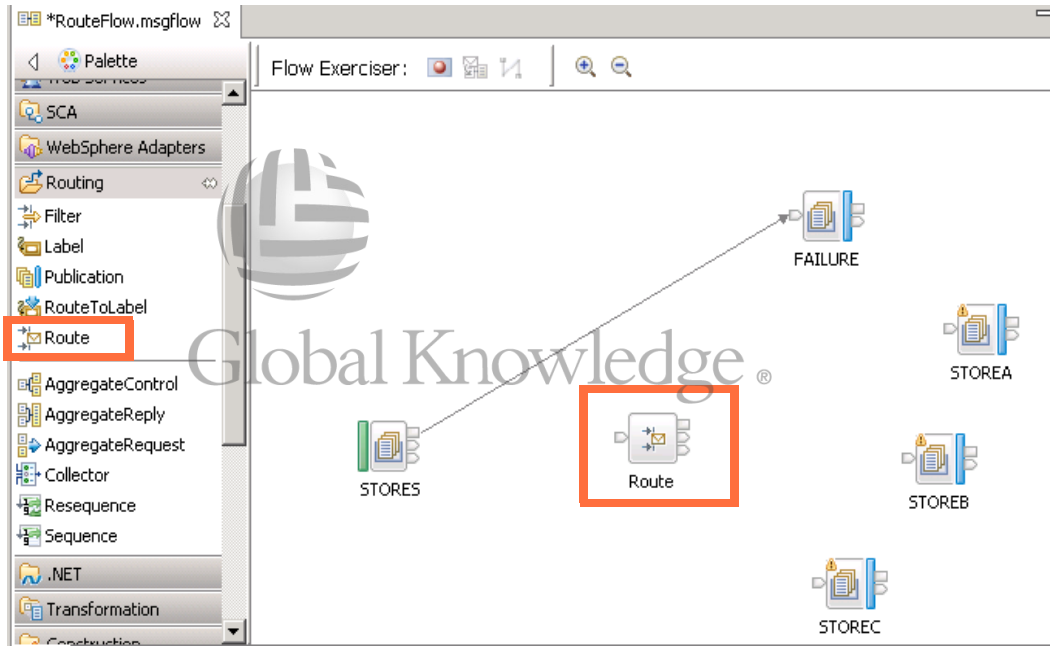
The **Message domain** property identifies the parser. In this exercise, the input message is an XML message so the **Message domain** is set to **XMLNSC**.



__ 7. Add the Route node to the message flow.

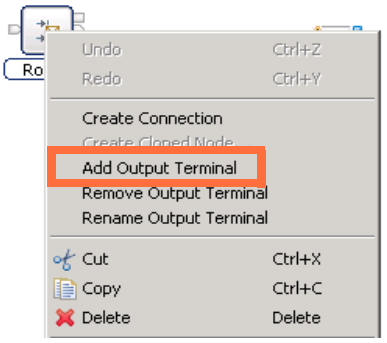
__ a. Expand the **Routing** drawer in the Message Flow editor Palette.

__ b. Drag the Route node from the Palette to the Message Flow editor canvas.



__ 8. Add output terminals to the Route node for each route.

__ a. Right-click the Route node and click **Add Output Terminal**.



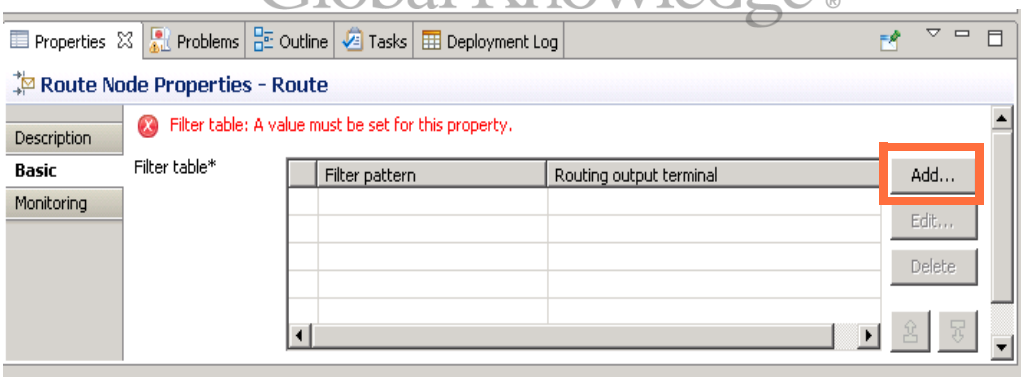
__ b. For the terminal name, type: **StoresA**

__ c. Click **OK**.

__ d. Using the same procedure, add two more output terminals that are named **StoresB** and **StoresC**.

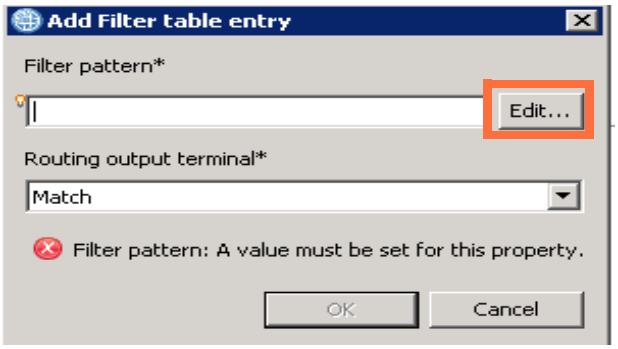
__ 9. Populate the Route node **Filter table** with the filter patterns and identify the routing output terminal.

__ a. On the **Basic** tab of the Route node **Properties** view, click **Add**.



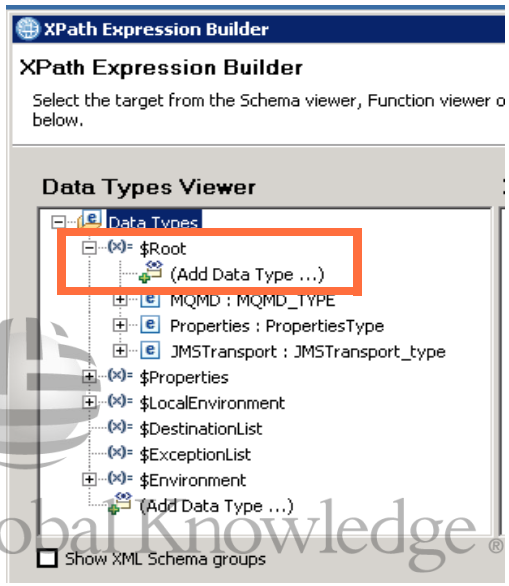
__ b. In this exercise, you use the XPath Expression Builder to create the filter pattern.

On the **Add Filter table entry** window, click **Edit** to open the XPath Expression Builder.

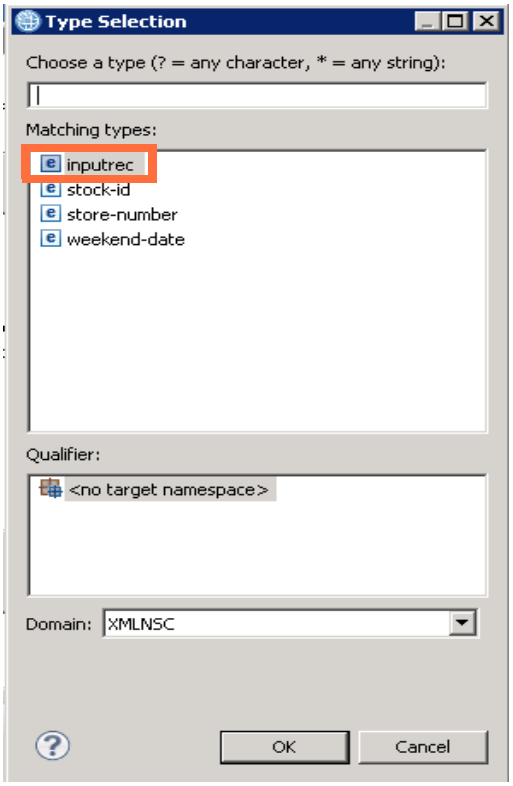


__ c. You need to add the XML schema structure and data types that define the XML message to the XPath Expression Builder.

In the **Data Types Viewer** pane of the XPath Expression Builder, expand **$Root** and then click **Add Data Type**.

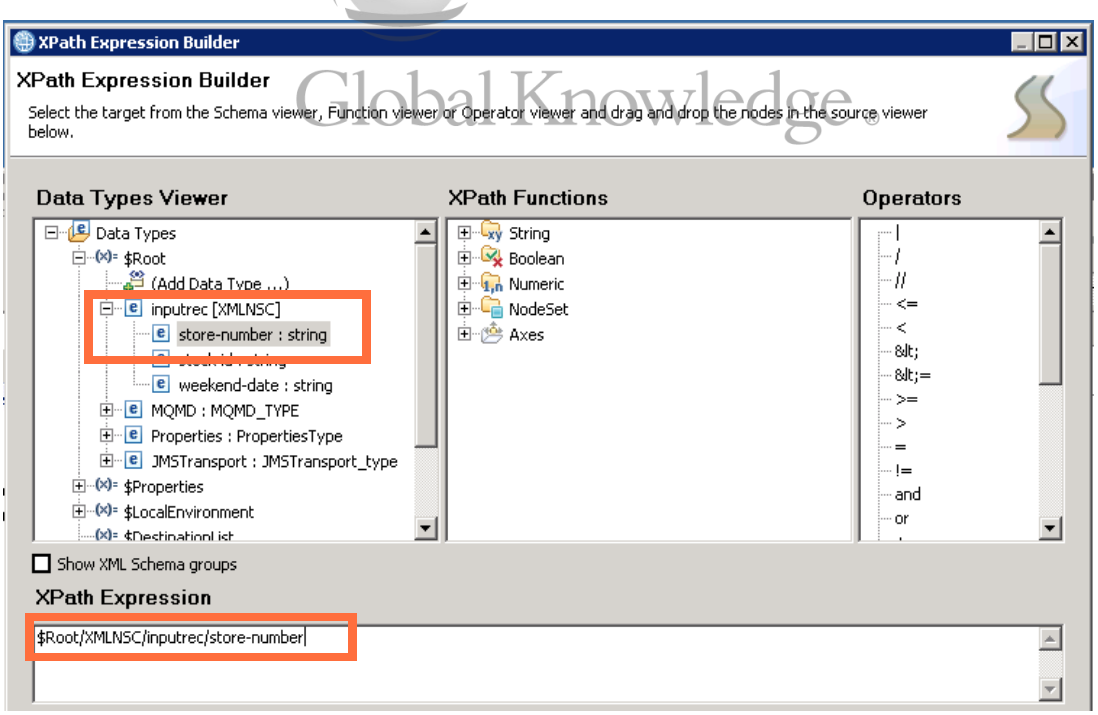


__ d. Select the top-level object in the `StoreProducts.xsd` schema, which is **inputrec.** Click **OK** to add it to the Data Types hierarchy.



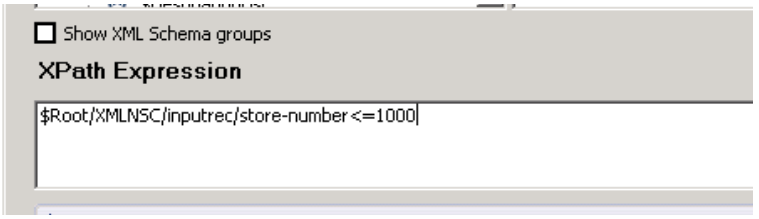
__ e. In the **Data Types Viewer** pane, expand **inputrec** to show its subtypes.

__ f. Double-click **store-number** to populate **XPath Expression** pane with the XPath to the **store-number** element.



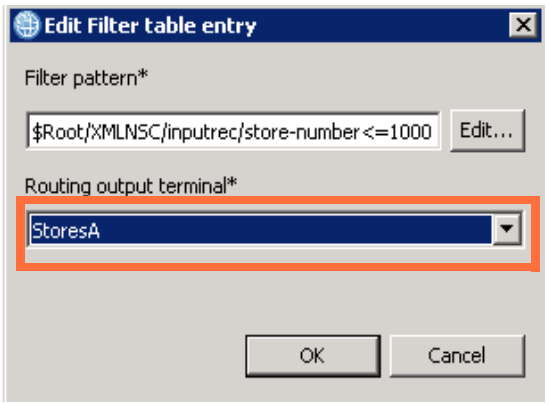
__ g. You can either type the remaining portion of the expression or select the operators (< and =) from the **Operators** column to complete the XPath Expression:

$Root/XMLNSC/inputrec/store-number<=1000



__ h. When the XPath Expression is complete, click **Finish**.

__ i. Select **StoresA** for the **Routing output terminal** and then click **OK**.



The filter pattern (XPath expression) and the routing output terminal should be listed in the Route note **Filter table**.

__ j. Using the same procedure, add the second filter pattern:

`$Root/XMLNSC/inputrec/store-number>1000` and
`$Root/XMLNSC/inputrec/store-number<=5000`

Select **StoresB** for the **Routing output terminal**.

Note

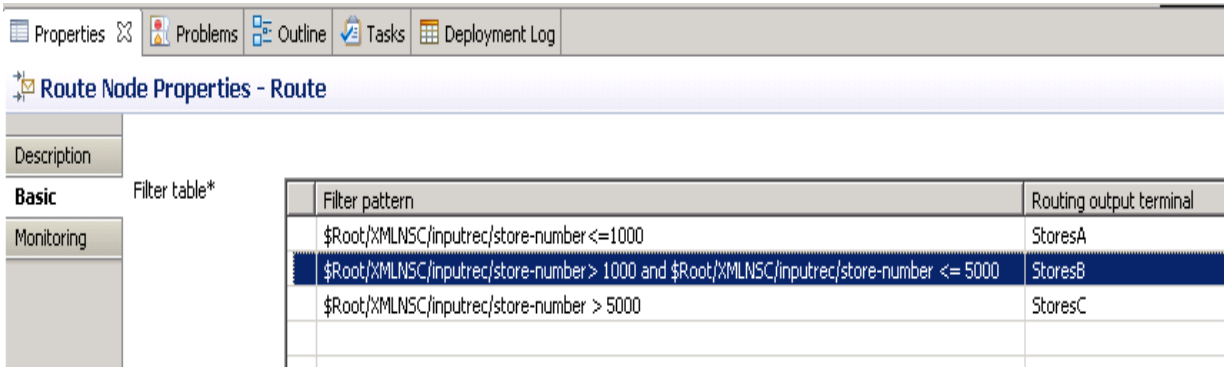
You do not need to repeat Step 9c to add the **inputrec** data type.

__ k. Using the same procedure, add the third filter expression.

`$Root/XMLNSC/inputrec/store-number>5000`

Select **StoresC** for the **Routing output terminal**.

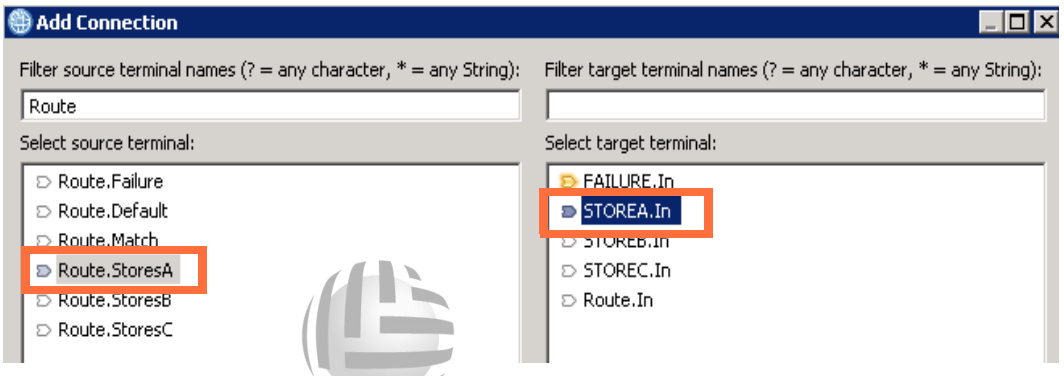
__ l. Verify that the **Filter table** on the Route node has three filter expressions that route the message to three different output terminals.



__ 10. Connect the Route node output terminals to the MQ Output nodes.

__ a. Right-click the Route node and then click **Create connection**.

__ b. Select **Route.StoresA** as the source terminal and **StoresA.In** as the target terminal.



__ c. Click **OK**.

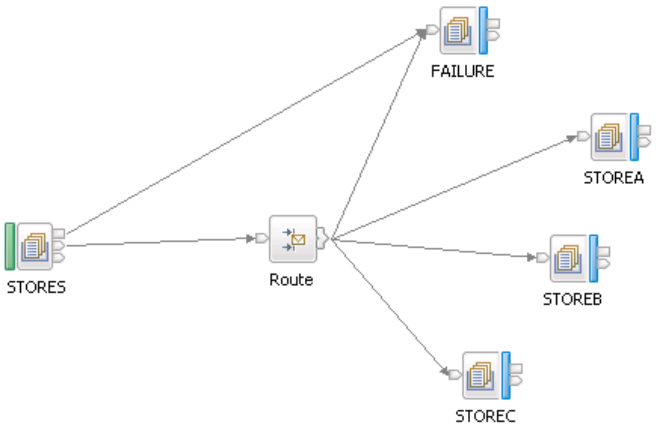
__ d. Using same procedure, add a connection from **Route.StoresB** to **STOREB.In**.

__ e. Add a connection from **Route.StoresC** to **STORESC.In**.

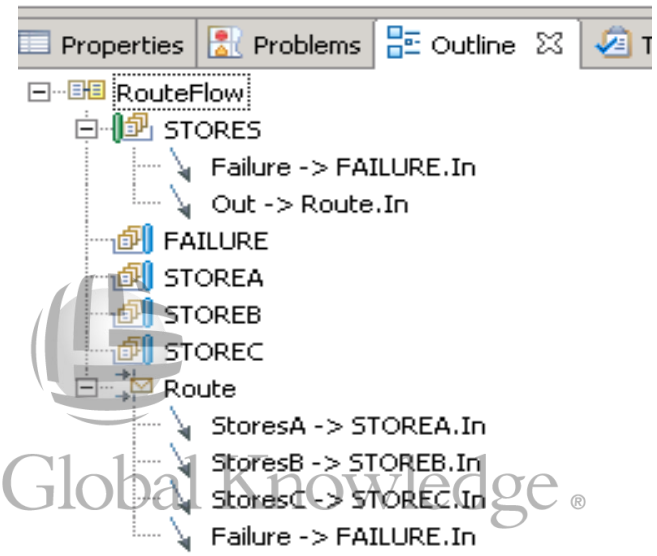
__ f. Add a connection from **Route.Failure** to **Failure.In**.

__ 11. In the Message Flow editor, connect the STORES MQInput node **Out** terminal to the Route node **In** terminal.

All the nodes in the message flow should now be connected.



__ 12. Verify the message flow connections in the **Outline** view.



__ 13. Save the flow.

__ 14. Verify that there are no warnings or errors in the **Problems** view.

Part 2: Test the message flow

In this part of the exercise, you use the IBM Integration Toolkit Flow exerciser to test the message flow.

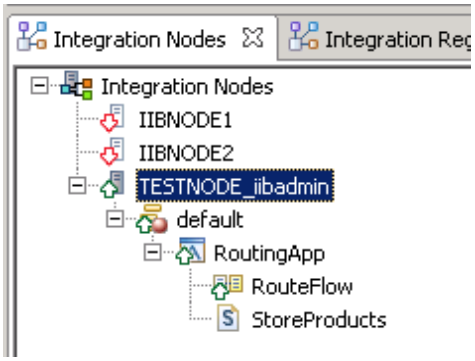
The `C:\labfiles\Lab04-Routing\data` directory contains three test files:

- When you test the flow with file `Stores201.xml`, the message should go to the STOREA node because `store-number` is <= 1000.
- When you test the flow with `Stores4444.xml`, the message should go to the STOREB node because the `store-number` is greater than 1000 and less than 5000.

- When you test the flow with `Stores7777.xml`, the message should go to the STOREC node because the `store-number` is greater than 5000.

__ 1. In the IBM Integration Toolkit Message Flow editor, start the Flow exerciser to create the BAR file, deploy the application to the **default** integration server on TESTNODE_iibadmin, and start recording.

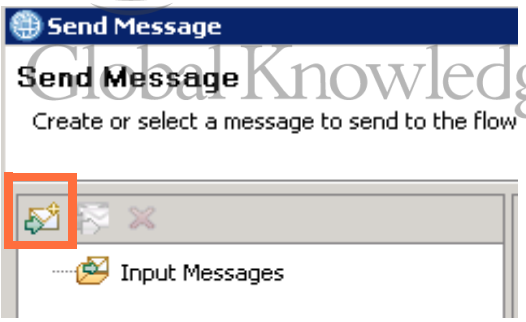
__ 2. Using the **Deployment Log** view and the **Integration Nodes** view, verify that the message flow application deployed successfully.



__ 3. Test the message flow with the `Stores201.xml` file by importing the file from the file system and sending the message with the Flow exerciser.

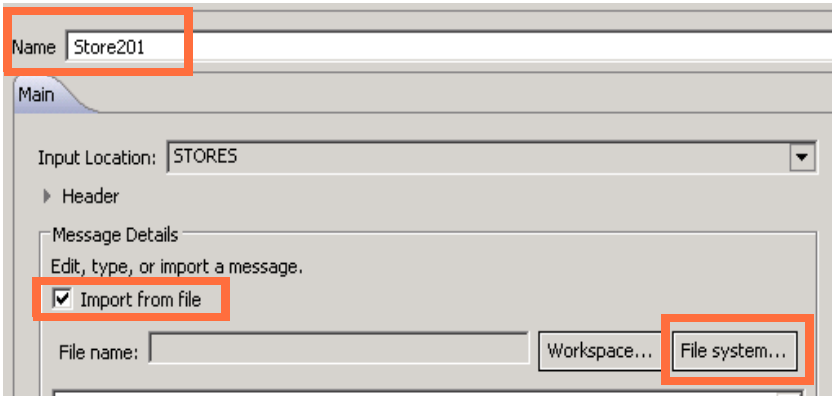
__ a. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

__ b. In the **Send Message** window, click the **New Message** icon.



__ c. For the **Name**, type: **`Store201`**

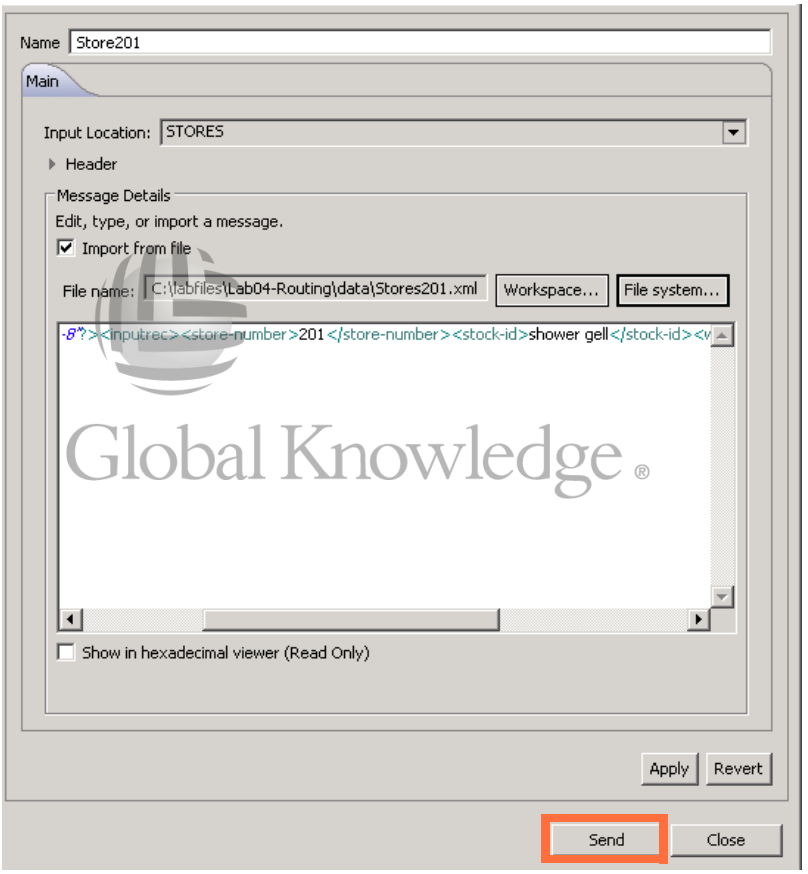
__ d. Click **Import from file** and then click **File system**.



__ e. Go to the `C:\labfiles\Lab04-Routing\data` directory, click the `Stores201.xml` file, and then click **Open**.

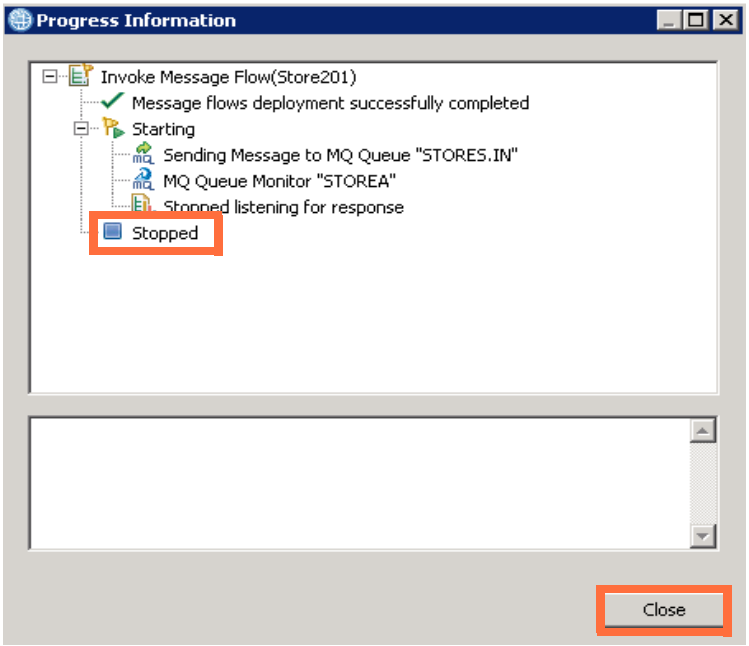
The file is imported into the Flow exerciser.

__ f. Click **Send**.

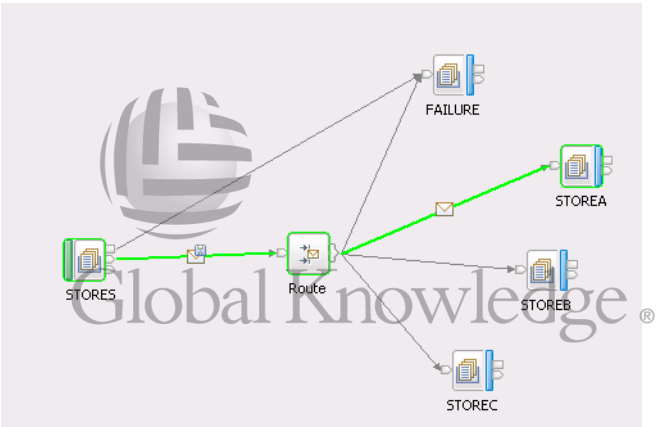


__ g. The **Progress Information** window shows that the message is sent to input queue STORES.IN. It also shows the destination, which is the STOREA queue.

When the **Progress Information** window displays **Stopped**, the test is complete. Click **Close**.



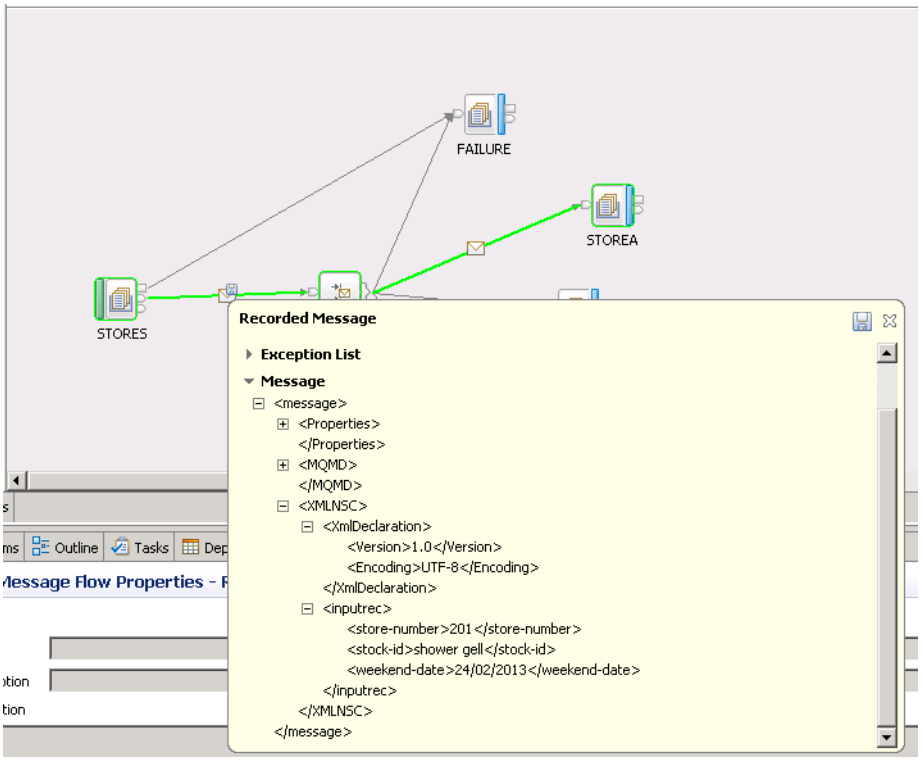
__ h. The message path is highlighted on the message flow.



For this test, the message should be routed to the STOREA queue because the store-number is less than 1000.

Verify that the message was routed to the correct queue.

__ i. Click the message icon to display the message and verify the value of the store-number in the message.



__ j. Close the message window.

Information

If you view the queues by using IBM MQ Explorer, you see that the **Current queue depth** for the output queue is empty. The Flow exerciser does not put the message to the output queue. It only shows the message path. If you want to view the messages on the queues, you can use the IBM MQ `amqsput` sample program or the Rfhutil SupportPac to test the flow by putting a message to the STORES.IN queue.

For example, to use the `amqsput` sample program to verify that the `Stores201.xml` message is put on the STORESA queue on the IIBQM queue manager, type the following command in a command prompt window:

```
amqsput STORES.IN IIBQM < C:\labfiles\Lab04-Routing\data\Store201.xml
```

__ 4. Following the same procedure that you used to import the message and test the flow with the `Stores201.xml` file, test the flow with the `Stores4444.xml` file.

The Flow exerciser message path should show that the message is routed to STOREB.

__ 5. Use the Flow exerciser to import the message and test the flow with the `Stores7777.xml` file.

The Flow exerciser message path should show that the message is routed to STOREC.

Exercise clean-up

__ 1. Close the message flow in the Message Flow editor.

__ 2. In the IBM Integration Toolkit **Integration Nodes** view, right-click the **default** integration server on **TESTNODE_iibadmin** and then click **Stop recording**.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, delete all flows and resources from **default** integration server on **TESTNODE_iibadmin**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you imported an IBM Integration Bus project interchange file that contains the XML schema that defines the message and a partially completed message flow. You completed the message by:

- Adding the Route node to the message flow
- Defining the route filter patterns and output terminals
- Wiring the Route node output terminals

In the second part of this exercise, you deployed and tested the message flow application by sending messages with different store numbers and verified that the message was routed to the correct output queue.

Having completed this exercise, you should be able to:

- Use the Route node to control message processing
- Use the XPath Expression Builder to define a filter pattern
- Create custom output terminals on the Route node
- Connect a Failure terminal to an output node to capture exceptions
- Test the message flow by importing messages into the IBM Integration Toolkit Flow exerciser

Exercise 5. Creating a DFDL model

What this exercise is about

In this exercise, you create a DFDL message model schema file in a shared library. The DFDL schema that you create defines a delimited text file. You test the model by using the Test Parse Model and Test Serialize Model options that are provided in the DFDL schema editor.

What you should be able to do

After completing this exercise, you should be able to:

- Create a DFDL message model schema file in a shared library
- Define the logical structure and physical properties of the message model elements
- Test a DFDL schema by parsing test input data
- Test a DFDL schema by serializing test data to create an output file

Introduction

In the first part of this exercise, you model the logical structure and physical properties of a complaint reply message by using DFDL.

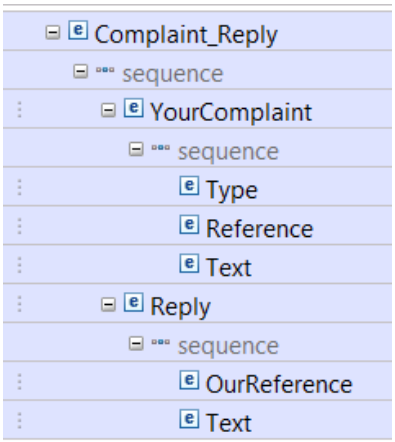
The reply file consists of two records that are delimited by a '|'. In each record, the string '+++' delimits each field in the record.


```
Delivery+++XYZ123ABC+++My order was delivered in time, but the package
was torn|C01-COM684a2da-384+++Your complaint has been received
```

This XML file describes the structure of the message.

```
<Complaint_Reply>
   <YourComplaint>
      <Type>Delivery</Type>
      <Reference>XYZ123ABC</Reference>
      <Text>My order was delivered in time, but the package was torn
      </Text>
   </YourComplaint>
   <Reply>
      <OurReference>C01-COM684a2da-384</OurReference>
      <Text>Your complaint has been received</Text>
   </Reply>
</Complaint_Reply>
```

This figure shows the DFDL message structure that you create in this exercise.



In the second part of this exercise you test the model by parsing sample data and then use the model to serialize data to ensure that the model creates the output file correctly.

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit
- The lab files in the **`C:\labfiles\Lab05-DFDL`** directory

Exercise instructions

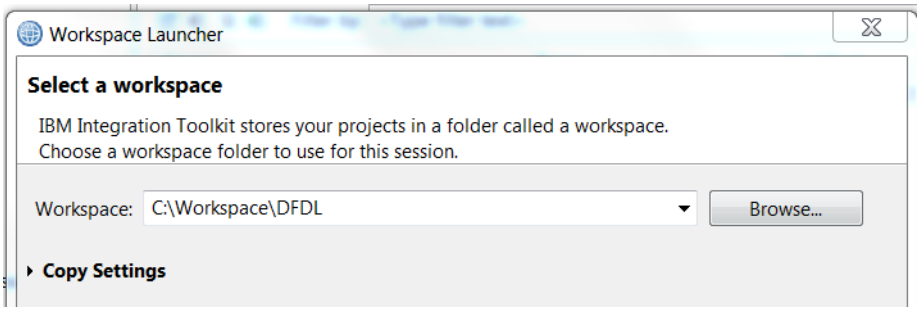
Exercise set up

__ 1. Start the IBM Integration Toolkit, if it is not already open.

__ 2. Open a new workspace.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\DFDL`**



__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

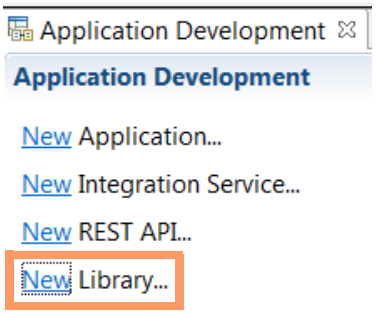
Part 1: Create the DFDL model that defines the reply message

In this part of the exercise, you create the message model that defines the reply file that is described in the exercise **Introduction**. You create the DFDL model is in a shared library.

Before starting this part of the exercise, review the file structure and contents of the file to ensure that you understand the structure of the data. A sample data file that is named `SampleComplaintReply.txt` is provided in the `C:\labfiles\Lab05-DFDL` directory.

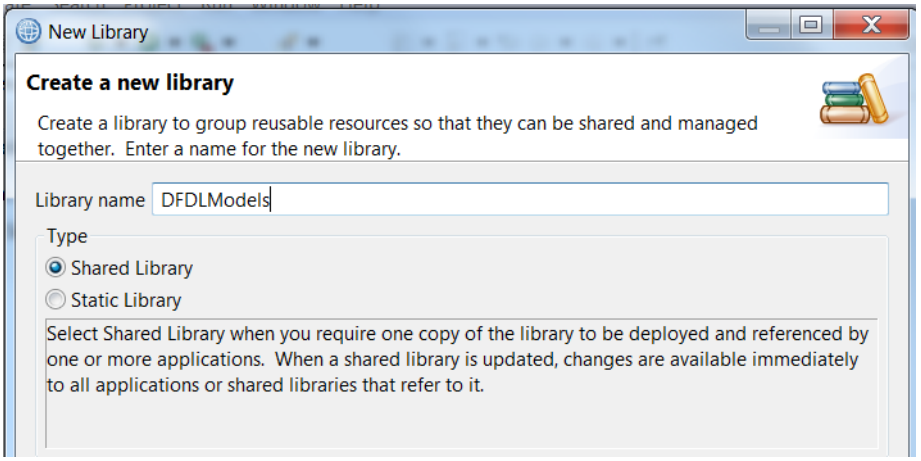
__ 1. Create a shared library to store the DFDL message model.

__ a. In **Application Development** view, click **New Library**.



__ b. For Library Name, enter: **`DFDLModels`**

__ c. Ensure that the **Shared Library** option is selected.

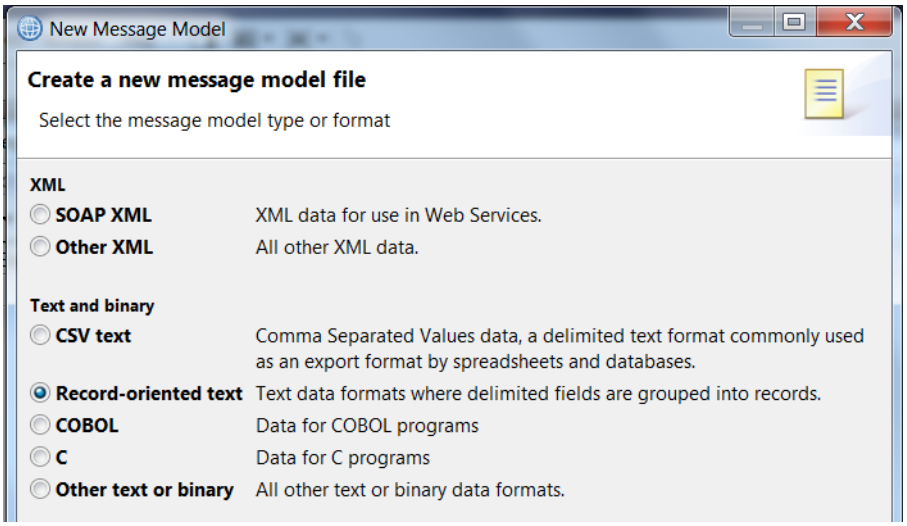


__ d. Click **Finish**.

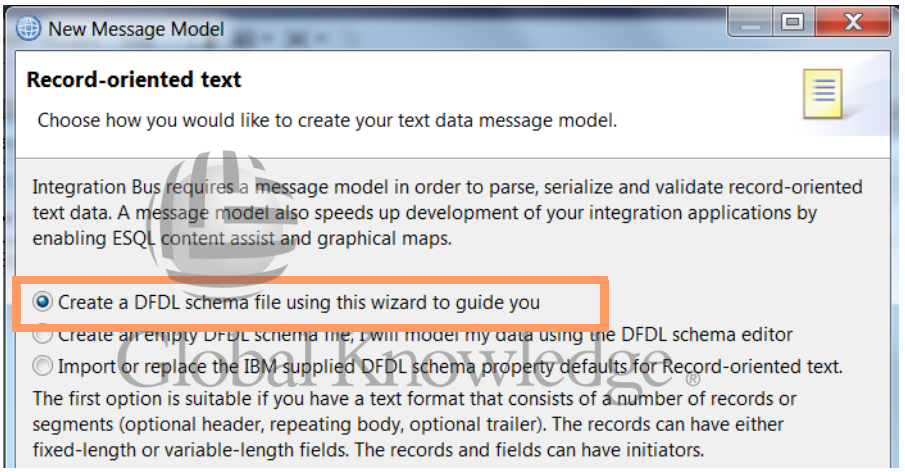
__ 2. Use the New Message Model wizard to create a DFDL schema definition file for the reply file that is described in the **Introduction** for this exercise.

__ a. Click **New** under the **DFDLModels** library folder in the **Application Development** view and then click **Message Model**.

__ b. On the **Create a new message model file** page of the wizard, select **Record-oriented text** under **Text and binary** and then click **Next**.

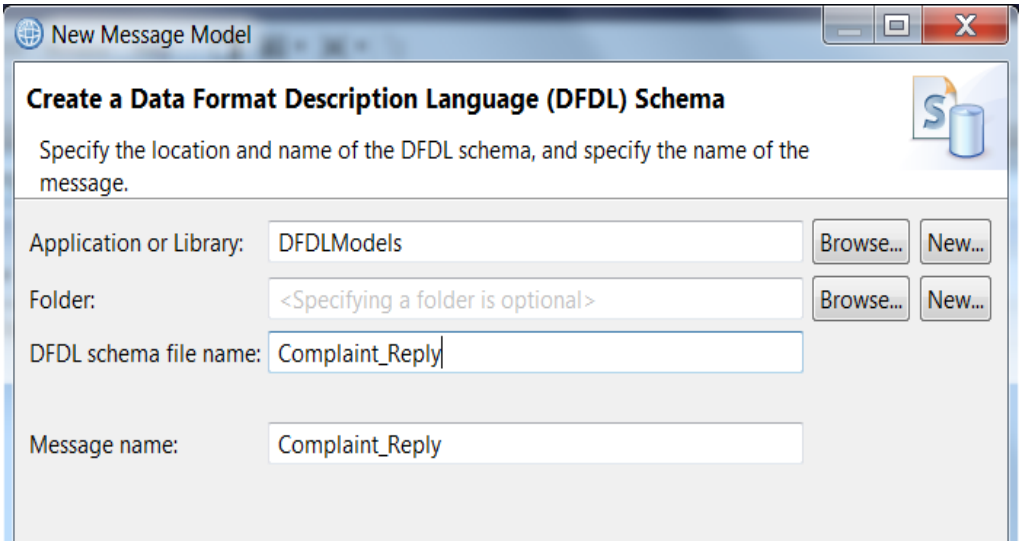


__ c. On the **Record-oriented text** page of the wizard, ensure that the option to **Create a DFDL schema file using this wizard to guide you** is selected and then click **Next**.



__ d. On the **Data Format Description Language (DFDL) Schema** page of the wizard, type **`Complaint_Reply`** for the **DFDL schema file name**.

The **Message name** field automatically uses the DFDL schema file name for the message name.



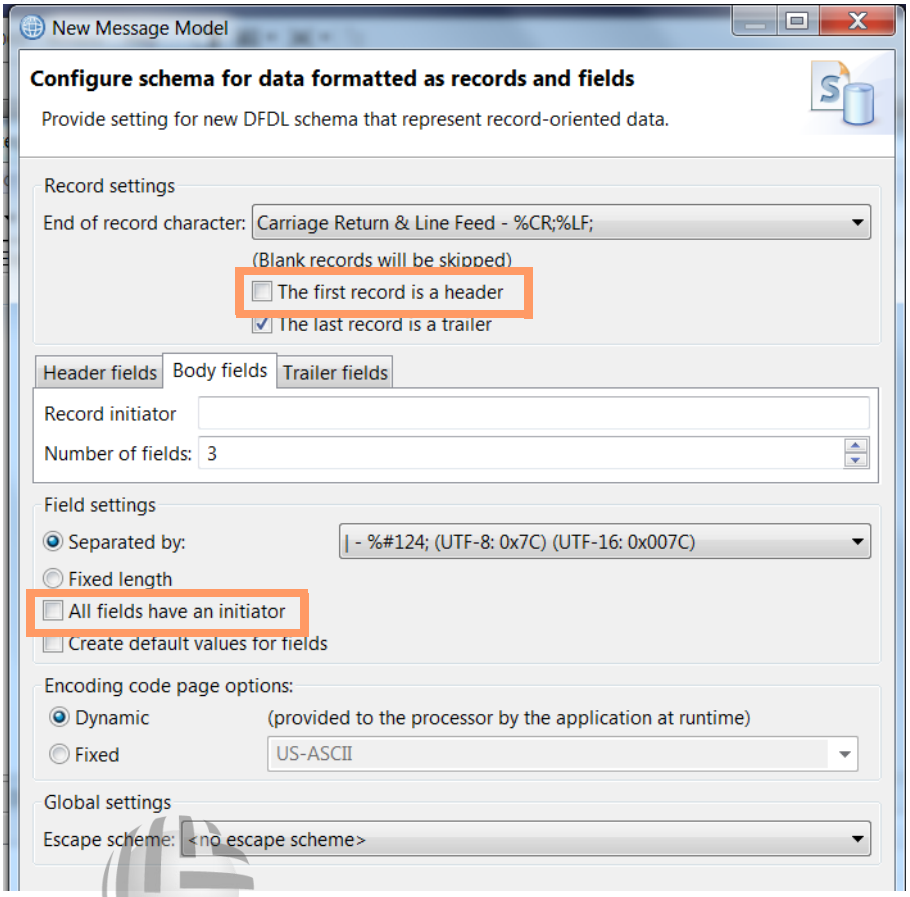
Because you started the wizard by clicking **New** under the DFDLModels library, the wizard automatically selects **DFDLModels** for the **Application or Library** field.

__ e. Click **Next**.

__ f. The Complaint_Reply file has two record types that are delimited with a '|' character. On the **Configure schema for data formatted as records and field** page, complete the following actions:

- Clear **The first record is a header** option under **Record settings**.
- On the **Body fields** tab, clear the **Record initiator** field and set **Number of fields** to 3
- On the **Trailer fields** tab, clear the **Trailer initiator** field and set **Number of fields** to 2

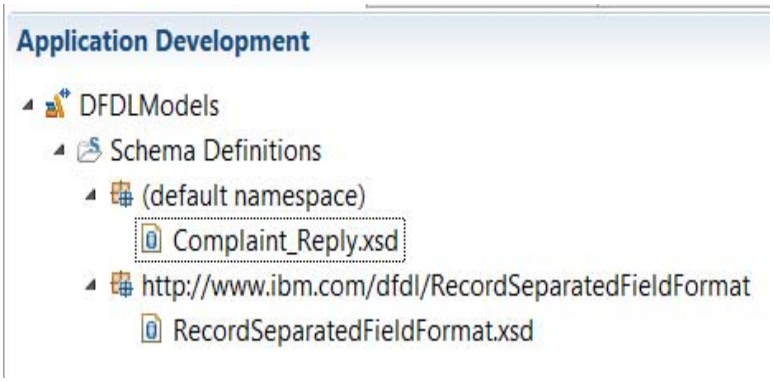
- Under **Field settings**, select **| - %#124; (UTF-8: 0x7c)(UTF-16: 0x007C)** for the **Separated by** field and clear the **All fields have an initiator** option.



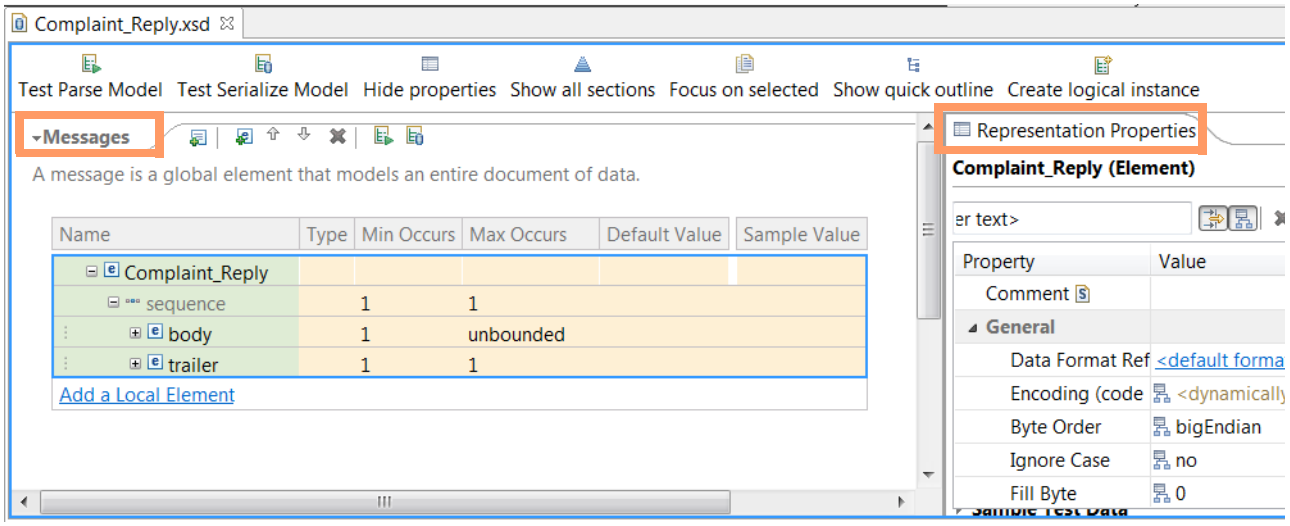
__ g. Click **Finish**.

The wizard creates a DFDL schema definition file that is named `Complaint_Reply.xsd` and opens the DFDL schema editor.

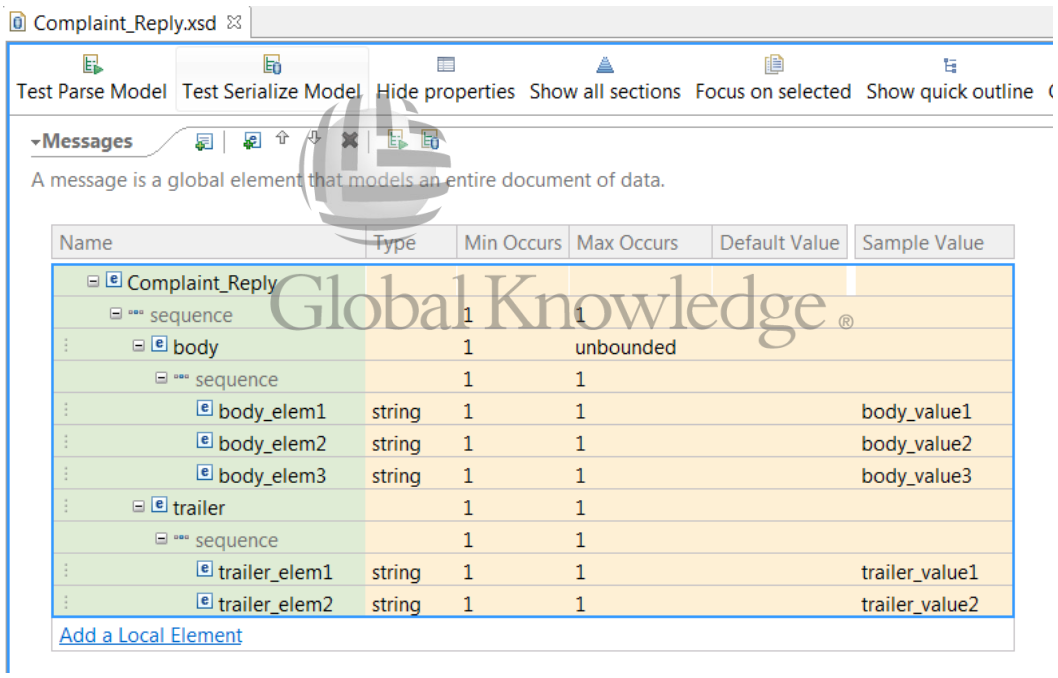
The wizard also includes the "Helper" schema `RecordSeparatedFieldFormat.xsd` in the library.



__ 3. The DFDL schema editor opens with two panes. The **Messages** pane on the left pane is where you define the message structure. The **Representation Properties** pane on the right is where you define the element properties.



__ 4. Expand the message elements so that you can see the structure and content of the Complaint_Reply message.

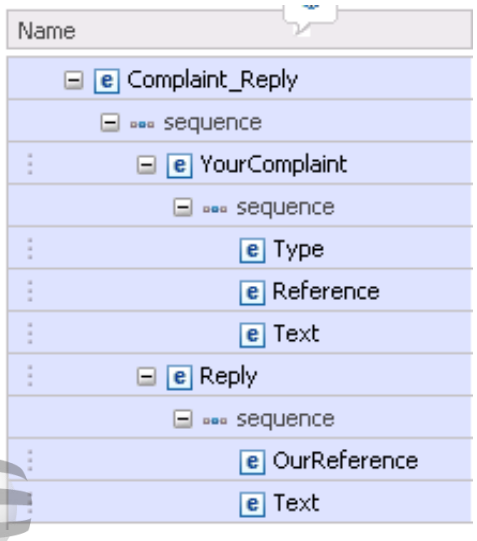


Reminder

To make it easier to see the entire view without scrolling, double-click the **Complaint_Reply.xsd** tab to expand the view. When you are done creating the DFDL model, you can double-click the tab again to return the view to the default size.

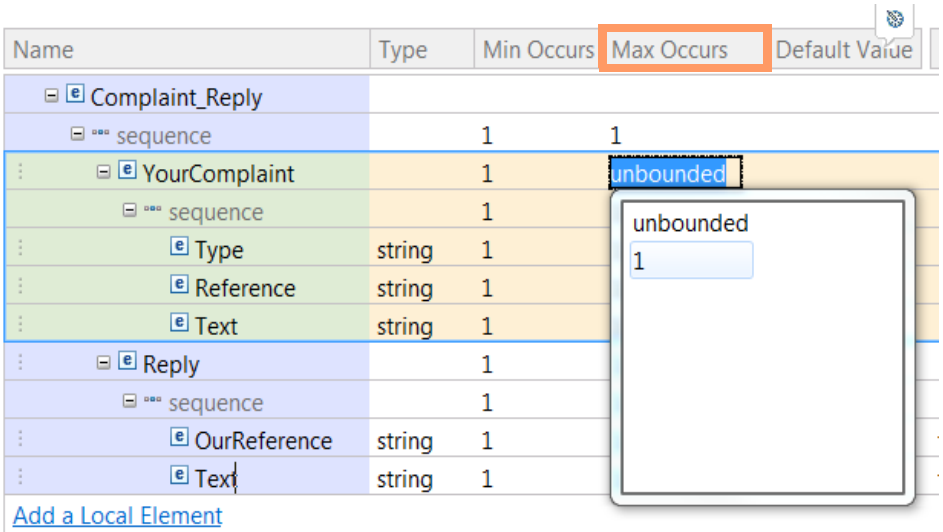
__ 5. Rename the message elements to use more descriptive names by selecting the element name and then typing a new name.

- Rename **body** to **YourComplaint**
- Rename **body_elem1** to **Type**
- Rename **body_elem2** to **Reference**
- Rename **body_elem3** to **Text**
- Rename **trailer** to **Reply**
- Rename **trailer_elem1** to **OurReference**
- Rename **trailer_elem2** to **Text**



__ 6. The Complaint_Reply message contains only one complaint.

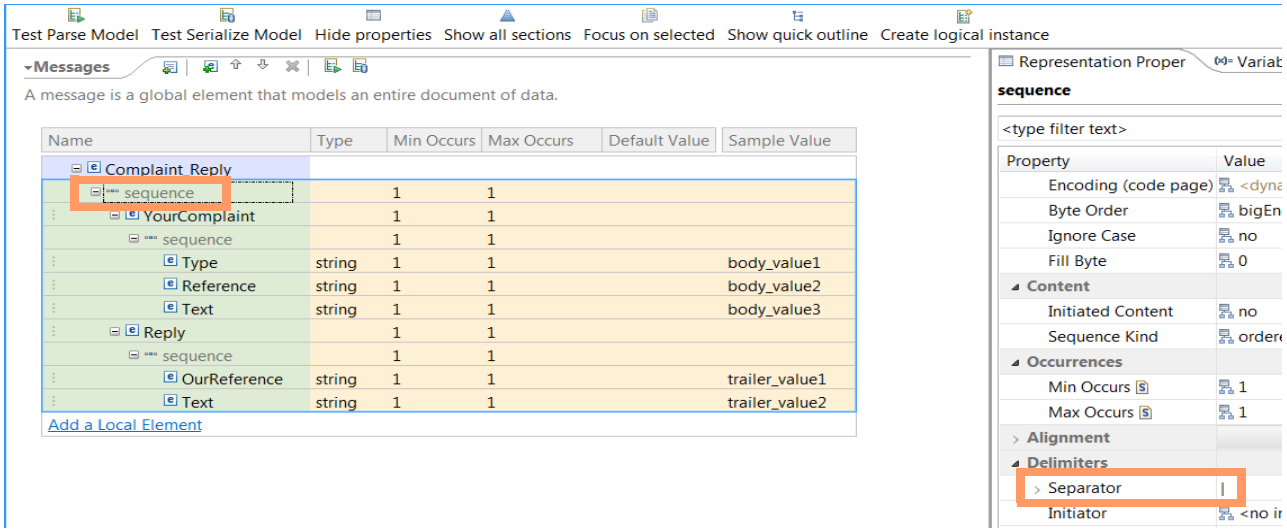
Change the **Max Occurs** value for **YourComplaint** from **unbounded** to **1** by clicking the **Max Occurs** field and selecting **1**.



__ 7. The Complaint_Reply message uses a '|' character to distinguish the first three fields (**YourComplaint**) from the fourth and fifth fields (**Reply**).

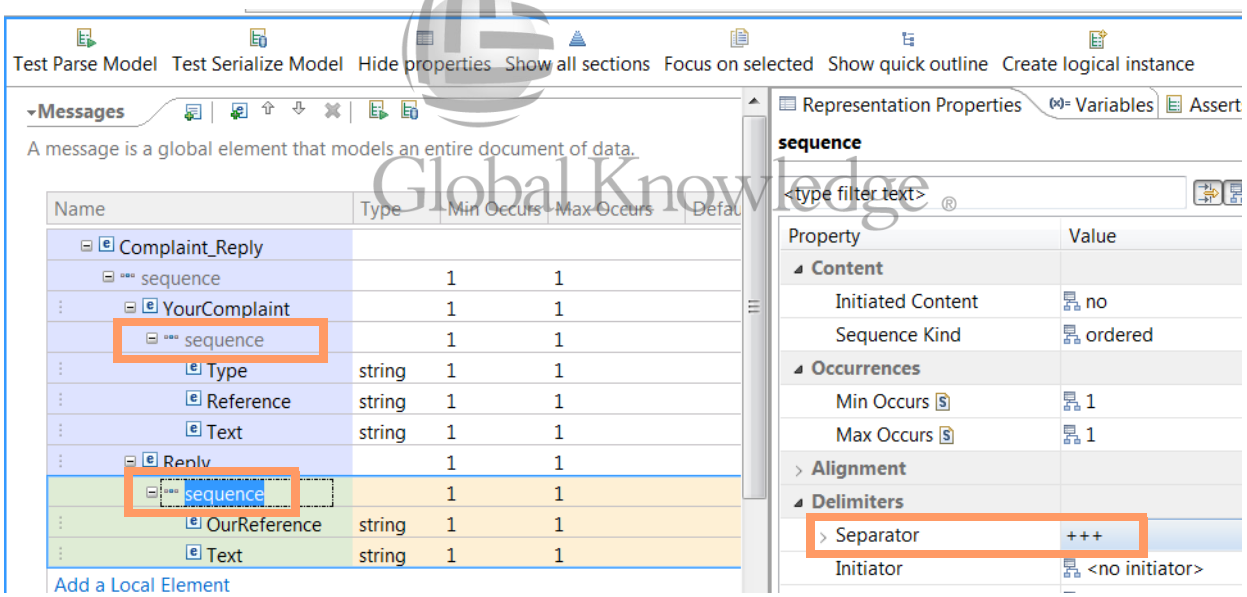
Configure the separator between **YourComplaint** and **Reply** by selecting **sequence** element under **Complaint_Reply** and changing the **Delimiters > Separator** property to **|** (vertical pipe symbol).

To hardcode a value for the **Separator**, click the **Value** field next to **Separator** and type the | character.



__ 8. The fields in **YourComplaint** and **Reply** are separated by '+++'.

Change the **Separator** value for the **sequence** elements under **YourComplaint** and **Reply** to +++.



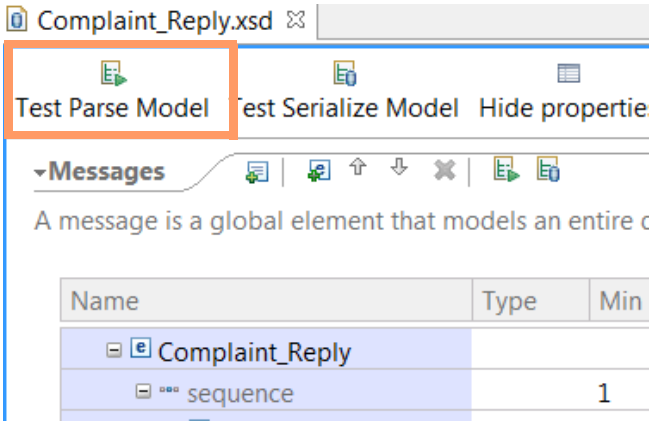
__ 9. Save the `Complaint_Reply.xsd` DFDL schema file.

Check the **Problems** view and the DFDL editor verify that no errors are flagged.

Part 2: Test the model

__ 1. To ensure that the DFDL model accurately describes the data, run the Test Parse Model option with a sample file.

__ a. In the DFDL editor, click **Test Parse Model**.



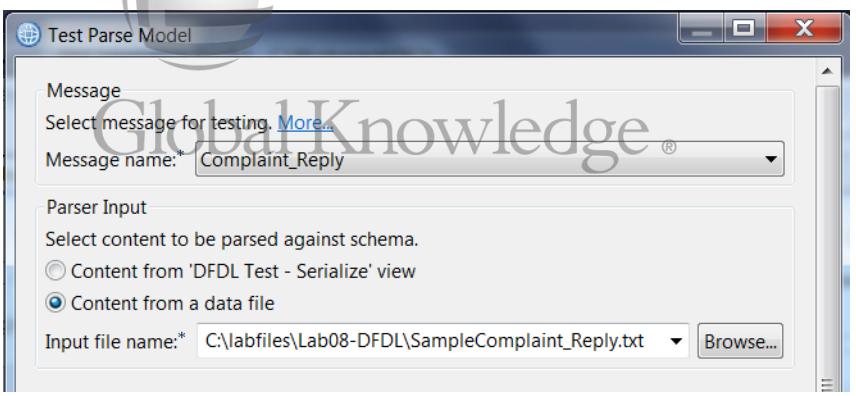
__ b. In the **Test Parse Model** window, select **Content from data file**.

__ c. For the **Input file name**, click **Browse**.

__ d. In the **File Selection** window, click **Select an input file from the file system** and then click **Browse**.

__ e. Browse to the `C:\labfiles\Lab05-DFDL` directory, select `SampleComplaint_Reply.txt`, and then click **Open**.

__ f. Click **OK** in the **File Selection** window.

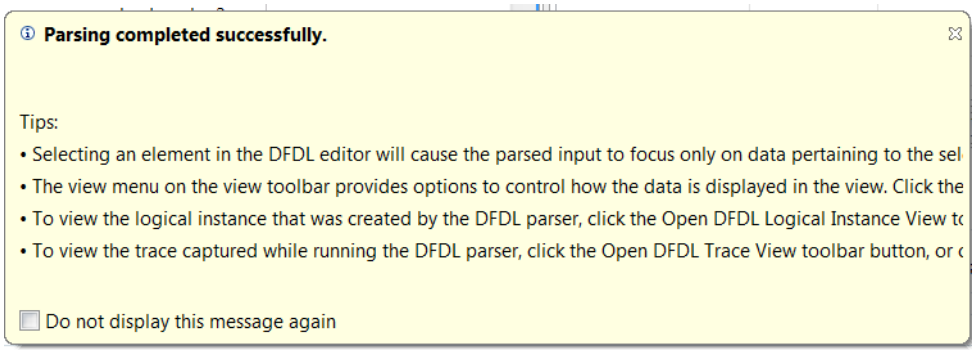


__ g. Click **OK** in the **Test Parse Model** window.

__ h. You receive a message that asks if you want to switch to the DFDL Test perspective. Click **Yes**.

__ i. The DFDL parser attempts to parse the sample data file by using the Complaint_Reply model.

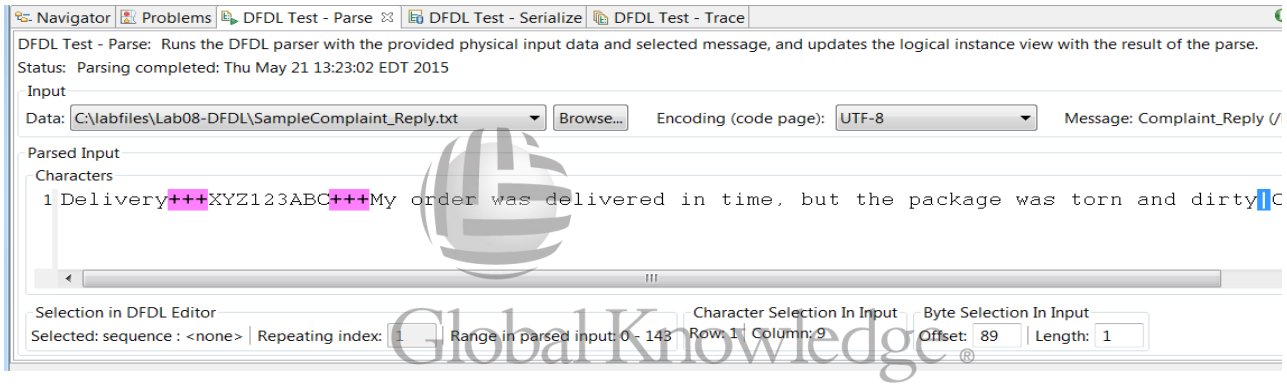
The message “Parsing completed successfully” is displayed if the DFDL parser can parse the sample data file.



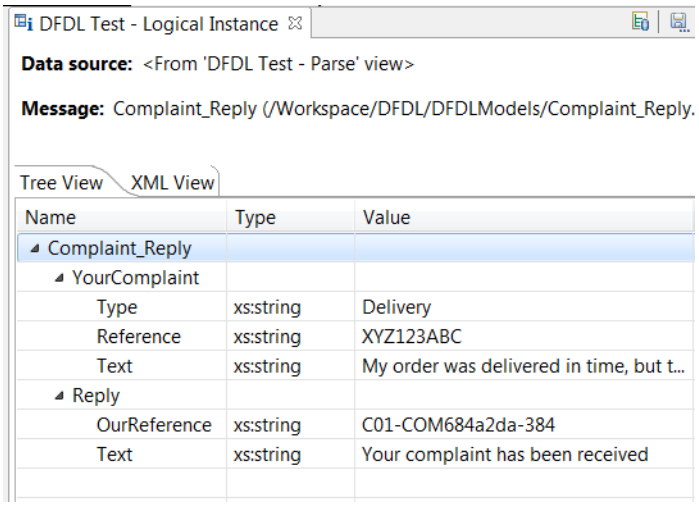
If parsing fails, an error message that describes the cause of the failure is displayed. Review the error message. If necessary, modify the DFDL Properties, save the model, and rerun the test until the Test Parse succeeds.

__ j. Close the **Parsing completed successfully** window.

__ 2. Review the parsed model in the **DFDL Test - Parse** tab. Ensure that the DFDL parser found the field separators (+++) and the record separator (|). The separators are highlighted in pink.



__ 3. Review the contents of **DFDL Test - Logical Instance** tab. This view shows you the logical view of the data upon completion of parsing by the DFDL parser.



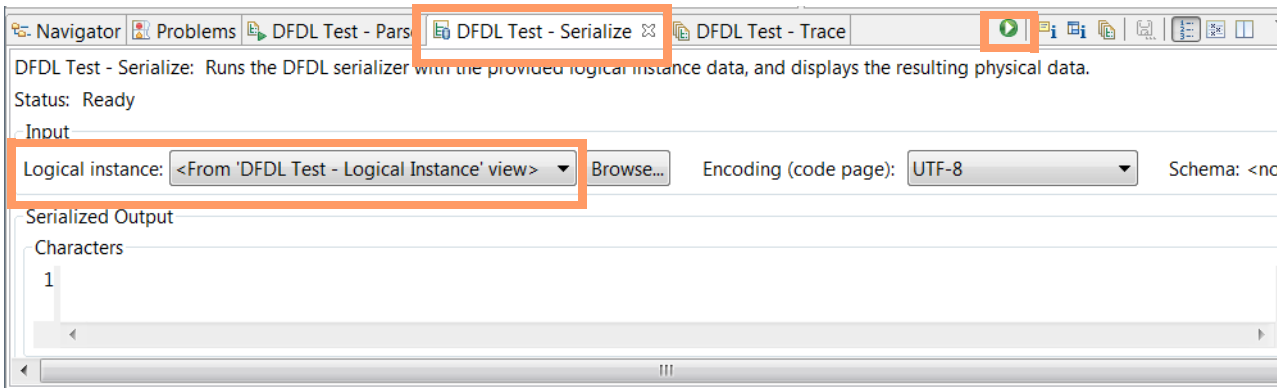
Compare this result to the data description in the lab exercise **Introduction** to ensure that the message was parsed correctly.

__ 4. This model is used to generate the **Complaint_Reply** file from a logical model on output. Run the **Test Serialize Model** option against the logical instance to ensure that the model builds the output data correctly.

__ a. In the DFDL Test perspective, click the **DFDL Test - Serialize** tab (at the bottom of the DFDL Test perspective).

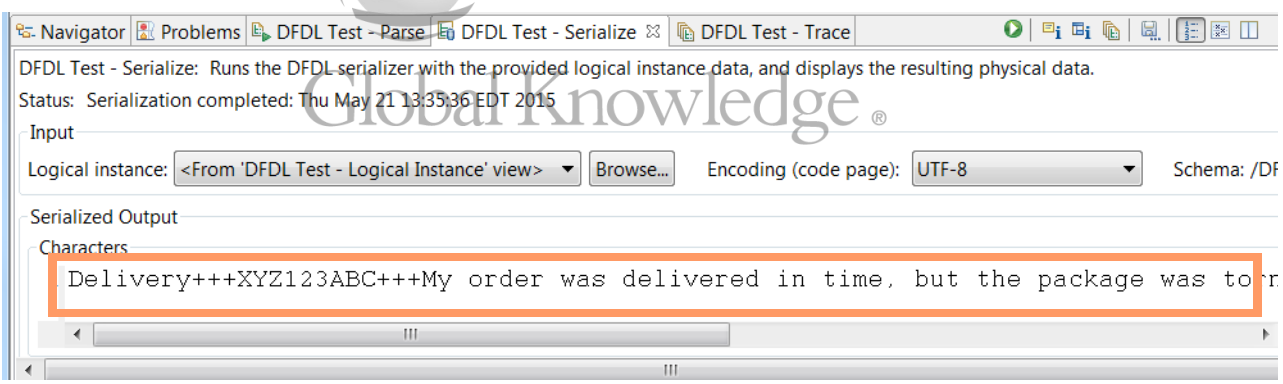
__ b. For the **Logical instance**, select **<From 'DFDL Test - Logical Instance View'>**

__ c. Click **Run Serializer** (the green and white arrow icon).



If the DFDL Serializer can successfully generate the output from the model, the message "Serialization completed successfully" is displayed. Close this window.

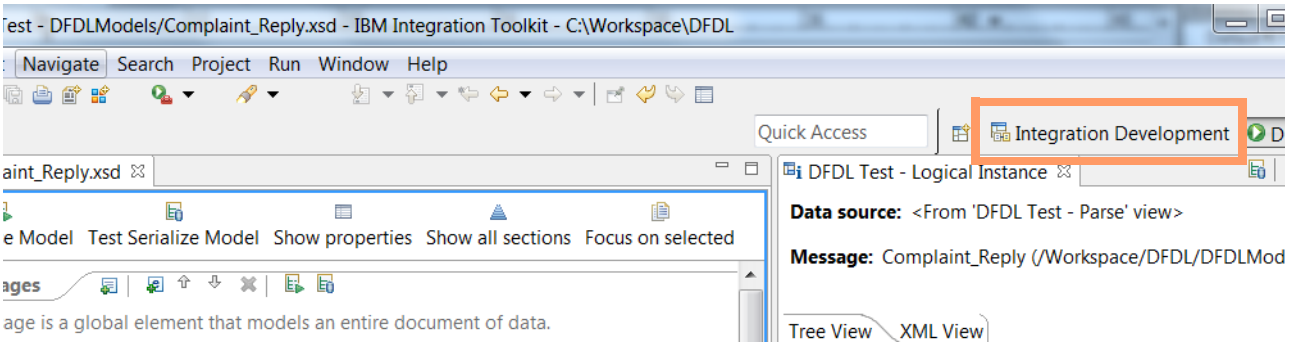
__ 5. Verify the output data that the DFDL Serializer generated. It should match the original sample input file `SampleComplaint_Reply.txt`.



Information

You can also save the result file that the DFDL Serializer creates. If you are working on a complex model, you can save the file and then use an external tool to compare the original sample input file with the file that the serializer creates. If your model accurately describes the data, the two files should match.

__ 6. Return to the Integration Development perspective by clicking **Integration Development**.



__ 7. Close the DFDL schema editor.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you created a shared library for DFDL models. You used the New Message Model wizard to build a basic DFDL model and the modified the properties to define a complaint reply message.

In the second part of this exercise, you tested the model by parsing sample data. You used the model to serialize data to ensure that the model creates the output file correctly.

Exercise 6. Processing file data

What this exercise is about

In this exercise, you create a message flow that reads an input file that contains many records, and creates a separate IBM MQ message for each record. You also import a library that contains the DFDL message model that defines the input file, and update the application to reference the library.

What you should be able to do

After completing this exercise, you should be able to:

- Use a FileInput node in a message flow
- Configure the FileInput node so that each record in the file is processed as a separate transaction
- Reference a library in a message flow application
- Use the IBM Integration Bus Toolkit Flow exerciser to view multiple output messages

Introduction

In this exercise, you create a message flow that generates one IBM MQ message for each record in an input file.

The input file, `EMPLOYEE.TXT`, is a fixed-format text file that contains four records. Each record is 60 characters and contains a 15 character surname, 10 character given name, 8 character hire date, 25 character title, and 2 character department code. A sample record is shown here:

```
Braden          Jasmine   19791224President & CEO           01
```

The input file is in the `C:\Labs\Lab06-File\data` directory.

The DFDL message model that describes the file is provided for you in the `C:\Labs\Lab06-File\` directory in a Project Interchange file that is named `EmployeeDFDLModel.zip`

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit and IBM MQ V8
- A local IBM MQ queue manager that is named IIBQM with the following local queues: DLQ, RECORDS
- Lab files in the `C:\labfiles\Lab06-Files` directory
- The user `iibadmin` is a member of the “mqm” group

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab06`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab06`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, verify that the integration node **TESTNODE_iibadmin** is started.

Start it if it is stopped.

__ 4. This exercise requires an IBM MQ queue manager that is named **`IIBQM`**.

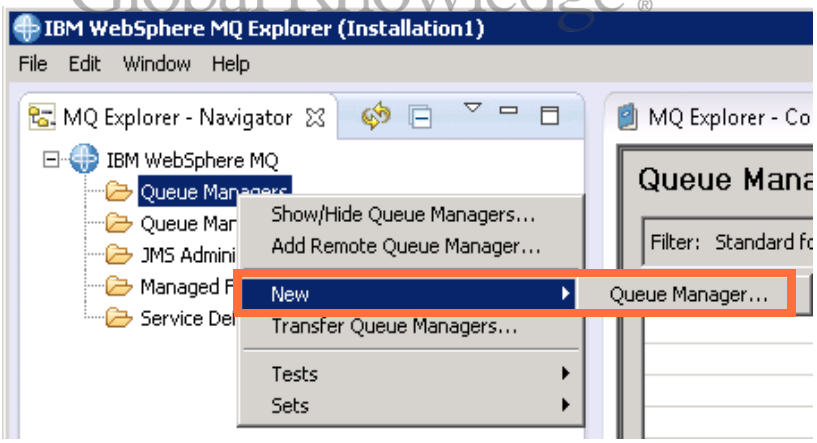
If you created this queue manager in a previous exercise, proceed to Step 5.

If you do not have a queue manager, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ` by following these instructions.

__ a. Start IBM MQ Explorer if it is not already running.

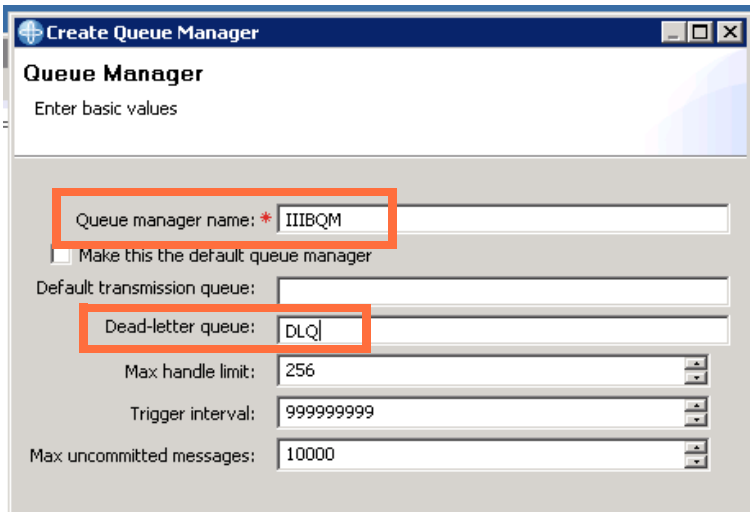
From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

__ b. In the **MQ Explorer - Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ c. For the **Queue Manager** name, type: **`IIBQM`**

__ d. For the **Dead-letter queue**, type: **DLQ**



__ e. Click **Finish**.

After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 5. Create the IBM MQ queues required for this exercise (DLQ and RECORDS) by running an IBM MQ script file.

In a command window, type:

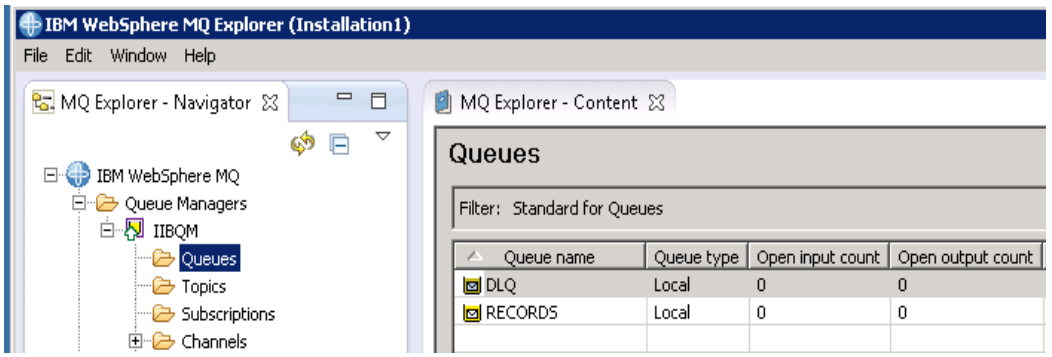
```
runmqsc IIBQM < C:\labfiles\Lab06-Files\CreateQueues.mqsc
```

__ 6. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the **MQ Explorer - Navigator** view, expand the **Queue Managers > IIBQM** folder.

__ b. Click **Queues** under the queue manager in the **MQ Explorer - Navigator** view to display the **Queues** view.

__ c. Verify that the following queues are listed in the **Queues** view: DLQ and RECORDS



Note

You might have other queues for this queue manager from a previous exercise. You do not need to delete the unused queues.

Part 1: Create the message flow

In this part of the exercise, you create the message flow that contains a FileInput node and an MQOutput node.

__ 1. Create an application that is named **FileToQueue**.

__ a. In the IBM Integration Toolkit **Application Development** view, click **New Application**.

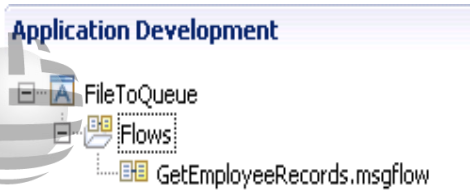
__ b. For the application name, type: **`FileToQueue`**

__ 2. Create a message flow that is named **GetEmployeeRecords**.

__ a. Under the **FileToQueue** application in the **Application Development** view, click **New** and then click **Message Flow**.

__ b. For the **Message Flow name**, type: **`GetEmployeeRecords`**

__ c. Click **Finish**. The Application Development view updates and the message flow opens in the Message flow editor.



__ 3. Add a FileInput node that is named **EmployeeFile** to the message flow.

__ a. Click the **FileInput** node in the Message flow Editor Palette **File** drawer and then click in the message flow editor canvas.

__ b. Click **File Input** under the node and then type: **`EmployeeFile`**

__ 4. Add an MQOutput node that is named **EmployeeRecords** to the message flow.

__ a. Click the MQOutput node in the Message flow Editor Palette **WebSphere MQ** drawer and then click in the message flow editor canvas.

__ b. Click **MQ Output** under the node and then type: **`EmployeeRecords`**.

__ 5. Wire the **Out** terminal of the FileInput node to the **In** terminal of the MQOutput node.

__ 6. Define properties for the MQOutput node that is named **EmployeeRecords**.

__ a. For the **Queue name** on the **Basic** properties tab, type: **RECORDS**

__ b. For the **Destination queue manager name** on the **MQ Connections** tab, type: **IIBQM**

__ 7. Save the message flow.

You should notice some errors on the message flow and **Problems** view because the **Input Directory** property is not configured on the FileInput node.

You configure the FileInput node in the next Part 3 of this exercise.

Part 2: Reference a library in a message flow application

In this part of the exercise, you import a library that contains the DFDL model that describes the input file. After you import the library, you add a reference from the application to the library so that you can access the model in the message flow properties. Finally, you modify the message flow to use the DFDL message model for input parsing.

__ 1. Import the DFDL message model for the input file.

__ a. From the IBM Integration Toolkit, click **File > Import**.

__ b. Click **IBM Integration > Project Interchange**, and then click **Next**.

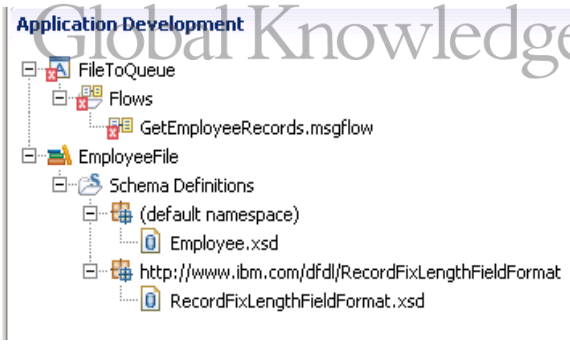
__ c. Click **Browse** next to **From .zip file**.

__ d. Browse to the `C:\Labs\Lab06-Files` directory.

__ e. Click **EmployeeDFDLModel.zip** and then click **Open**.

__ f. The Project Interchange file contains one artifact, which is a library that is named **EmployeeFile**. Ensure that it is selected and then click **Finish**.

__ 2. Verify that **EmployeeFile** library is listed in the **Application Development** view.

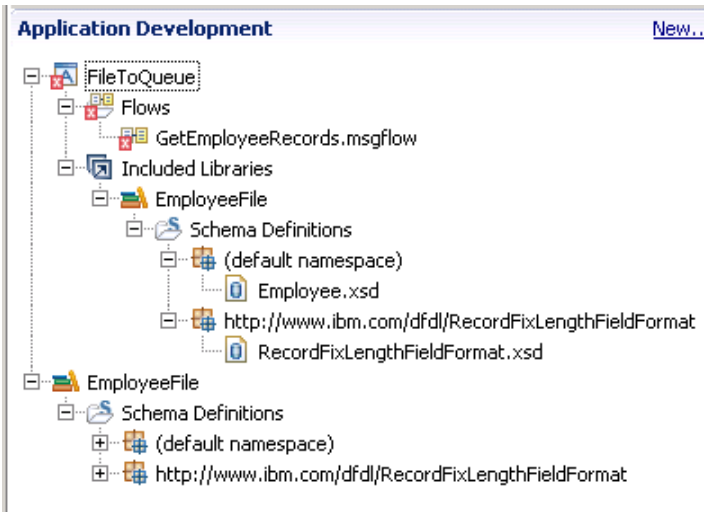


__ 3. Change the **FileToQueue** application to reference the **EmployeeFile** library.

__ a. In the **Application Development** view, right-click the **FileToQueue** application and then click **Manage Library References** from the menu.

__ b. Click **EmployeeFile** and then click **OK**.

__ c. The **EmployeeFile** library should now be displayed under the **FileToQueue** application under the **Included Libraries** folder.



Part 3: Configure the FileInput node

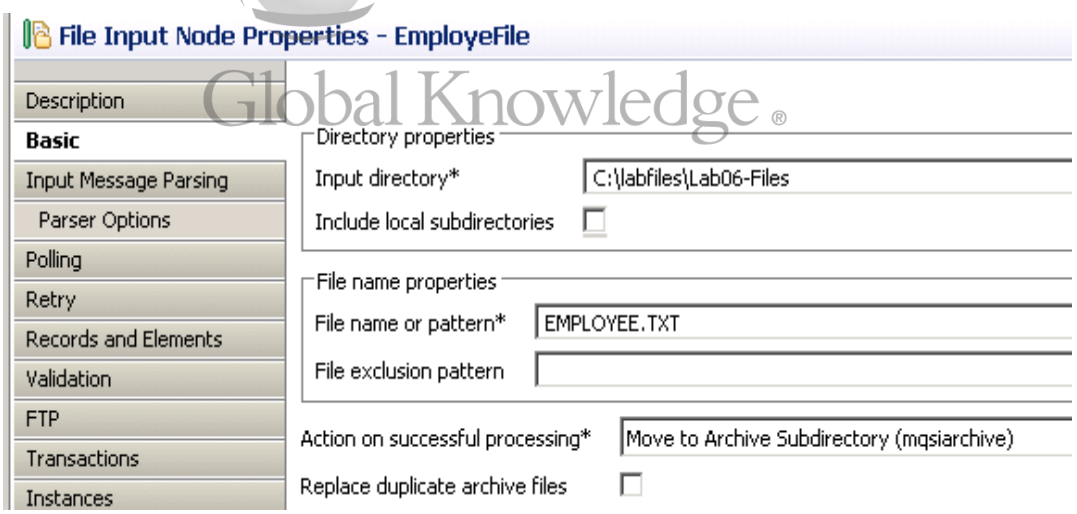
__ 1. Click the FileInput node in the Message Flow editor to display the node properties.

__ 2. Set the **Basic** properties.

__ a. For the **Input directory** property, type: **`C:\labfiles\Lab06-Files`**

__ b. For the **File name or pattern** property, type: **`EMPLOYEE.TXT`**

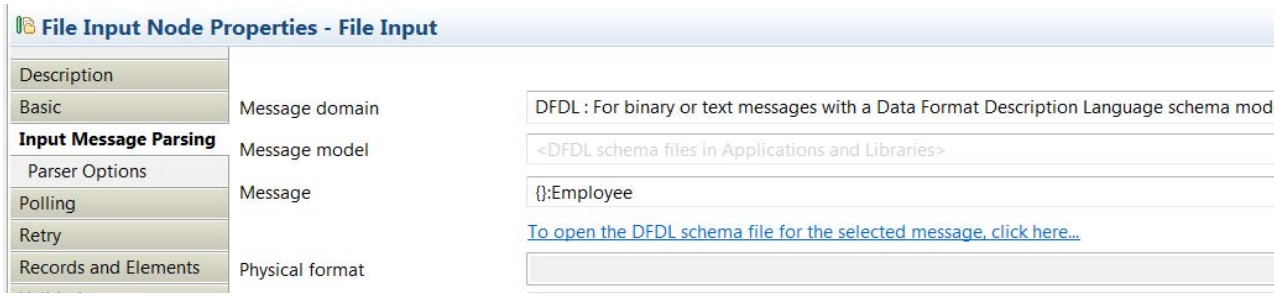
__ c. For the **Action on successful processing** property, select **Move to Archive Subdirectory**



__ 3. Set the **Input Message Parsing** properties to use the Employee message model.

__ a. For the **Message domain** property, select **DFDL**.

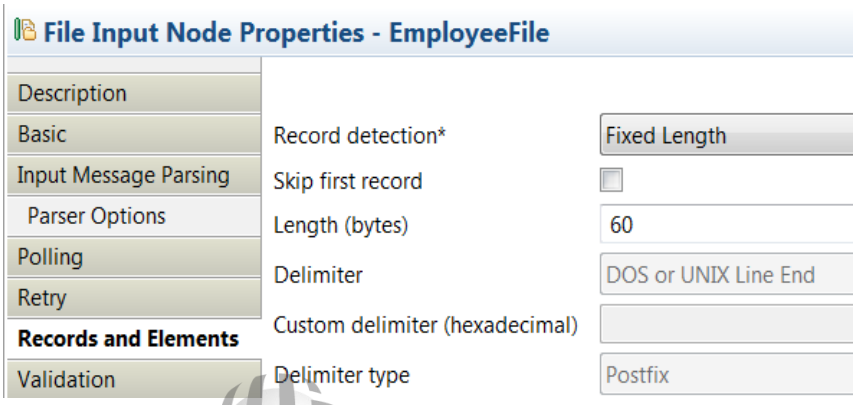
__ b. For **Message** property, click **Browse**, click **Employee {}**, and then click **OK**.



__ 4. Set the **Records and Elements** properties.

__ a. Set the **Record detection** property to **Fixed Length**.

__ b. For the **Length (bytes)** property, type: **60**.



__ 5. Save the changes to the message flow.

Part 4: Test the message flow

In this part of the exercise, you use the IBM Integration Toolkit Flow exerciser to test the message flow. The `C:\labfiles\Lab06-File\data` directory contains the test file `EMPLOYEE.TXT`.

__ 1. In the IBM Integration Toolkit Message Flow editor, start the Flow exerciser to create the BAR file, deploy the application to the **default** integration server on TESTNODE_iibadmin, and start recording.

__ 2. Using the **Deployment Log** view and the **Integration Nodes** view, verify that the message flow application deployed successfully.

__ 3. To test the flow, use Windows Explorer to copy the file `EMPLOYEE.TXT` from `C:\labfiles\Lab06-Files\data` to the `C:\labfiles\Lab06-Files` directory.

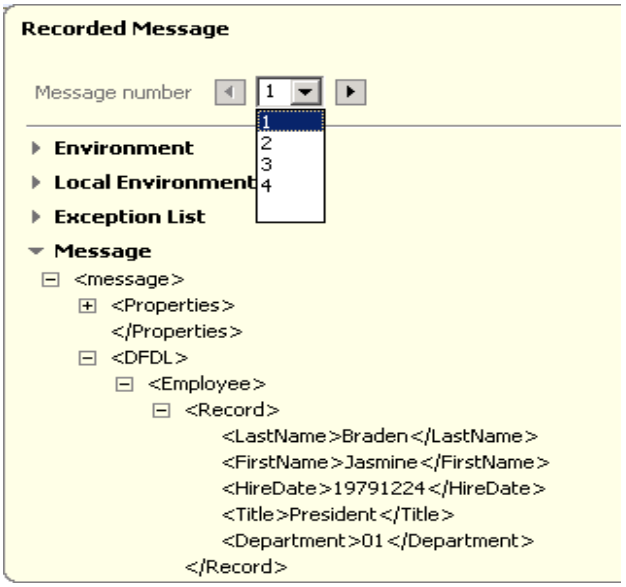
The flow runs as soon as the file is copied to the **Input directory** that is specified in the FileInput node properties.

- If the flow ran successfully, a copy of the file is also written to the `C:\labfiles\Lab06-Files\mqsiarchive` directory.
- If the flow failed, a copy of the file is written to the `C:\labfiles\Lab06-Files\mqsibackout` directory.

__ 4. Click the **View path** icon in the Flow exerciser to display the message path.

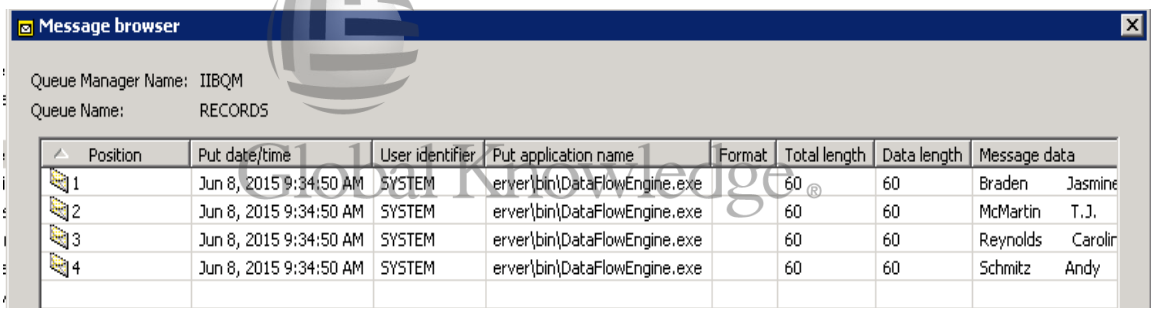
__ 5. Click the message icon on the path to display the messages that the FileInput node creates. The **Recorded Message** window should show that four messages were created from the input file.

__ 6. Select each **Message number** to view each message. You should see that the FileInput node used the Employee DFDL model to parse the message.



__ 7. Open IBM Integration Explorer to check the RECORDS queue for messages.

The RECORDS queue should contain four messages; one for each record in the input file.



Exercise clean-up

__ 1. In the IBM Integration Toolkit **Integration Nodes** view, right-click the **default** integration server on **TESTNODE_iibadmin** and then click **Stop recording**.

__ 2. In the IBM Integration Toolkit **Integration Nodes** view, delete all flows and resources from **default** integration server on **TESTNODE_iibadmin**.

__ 3. Close the message flow in the Message Flow editor.

End of exercise

Exercise review and wrap-up

In this first part of this exercise, you created a message flow that contains a FileInput node and an MQOutput node.

In the second part of this exercise, you imported a library and referenced the library from the application.

In the third part of this exercise, you configured the FileInput node to read the file EMPLOYEE.TXT and parse the file by using the Employee DFDL model.

In the fourth part of this exercise, you tested the exercise by using the IBM Integration Toolkit Flow exerciser.

Having completed this exercise, you should be able to:

- Use a FileInput node in a message flow
- Configure the FileInput node so that each record in the file is processed as a separate transaction
- Reference a library in a message flow application
- Use the IBM Integration Bus Toolkit Flow exerciser to view multiple output messages

Exercise 7. Using problem determination tools

What this exercise is about

In this exercise, you use various tools and procedures to diagnose runtime errors in message flow applications. You also learn how to add a Trace node to a message flow and customize the Trace node output.

What you should be able to do

After completing this exercise, you should be able to:

- Enable a user trace and retrieve the collected trace data
- Add a Trace node to a message flow application
- Use RFHUtil to send test data to a message flow and view messages on an IBM MQ queue
- Use the IBM Integration Toolkit Test Client and message flow debugger view to step through a message flow application
- Examine the IBM Integration Bus logs and system logs to diagnose problems

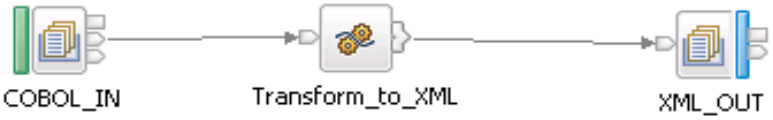
Introduction

In some cases, the IBM Integration Toolkit Flow exerciser might not provide all the information that you need to determine the cause of a problem in a message flow application. For example, the Flow exerciser does not report an error for an invalid queue name on an MQOutput node.

In this exercise, you use the system log (Event Viewer), Trace node, User Trace, Message Flow Debugger, and Unit Test Client to identify the cause of a problem in a message flow.

The message flow for this exercise contains the following nodes:

- An MQInput node that is named COBOL_IN that reads a message from a queue that is named COBOL_IN
- A Compute node that is named Transform_to_XML that uses ESQL to transform the COBOL data to XML
- An MQOutput node that is named XML_OUT that writes the XML message to the output queue that is named XML_OUT



In the first part of this exercise, you import a project interchange that contains the message flow application that you use in this exercise. You extend the message flow by adding a Trace node and then run the flow so that you can examine the trace results. You also use the IBM Integration Bus commands to turn off the Trace node and to verify the change.

In the second part of this exercise, you change the message flow so that it fails at run time. Then, you activate and examine a user trace and the system logs (Event Viewer) to identify the problem. You also use the IBM MQ SupportPac RFHUtil to put and get messages from the queues.

In the third part of this exercise, you set up the Message Flow Debugger and set breakpoints in the message flow. You then use the Debug perspective in the IBM Integration Toolkit to step through the message flow and examine the ExceptionList to identify the problem with the message flow.

In the fourth part of this exercise, you use the Unit Test Client Component Trace to identify the cause of a message flow failure and examine the ExceptionList.

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit and IBM MQ V8
- A local IBM MQ queue manager that is named IIBQM with the following local queues: DLQ, COBOL_IN, and XML_OUT
- Lab files in the `C:\labfiles\Lab07-PDTools` directory
- IBM MQ SupportPac IH03 (RFHUtil) in the `C:\Software\ih03` directory
- User `iibadmin` that is a member of the “mqm” group

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab07`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab07`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, verify that the integration node **TESTNODE_iibadmin** is started.

Start it if it is stopped.

__ 4. This exercise requires an IBM MQ queue manager.

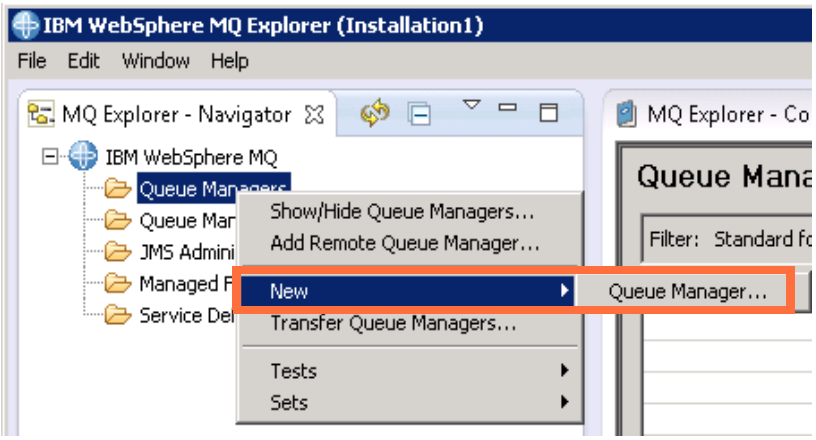
If you completed a previous exercise where you created the queue manager that is named IIBQM, you can use that queue manager in this exercise. Proceed to Step 5.

If you do not have a queue manager, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ` by following these instructions.

__ a. Start IBM MQ Explorer if it is not already running.

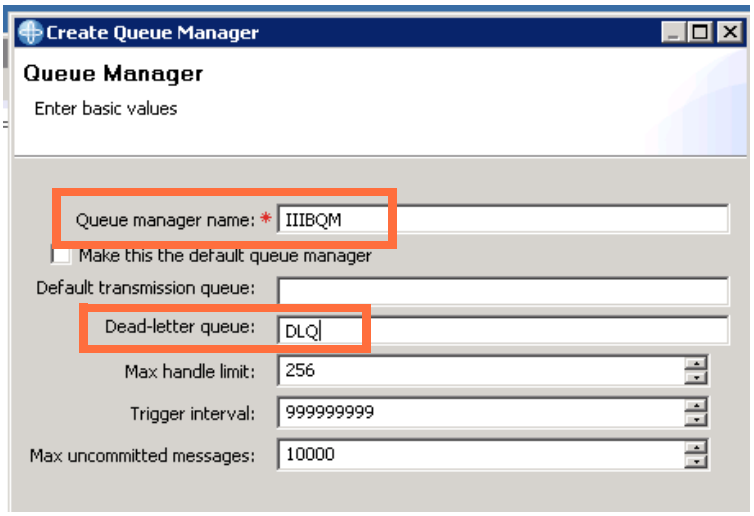
From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

__ b. In the **MQ Explorer Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ c. For the **Queue Manager** name, type: **`IIBQM`**

__ d. For the **Dead-letter queue**, type: **DLQ**



__ e. Click **Finish**.

After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 5. Create the IBM MQ queues required for this exercise (DLQ, COBOL_IN, and XML_OUT) by running an IBM MQ script file.

In a command window, type:

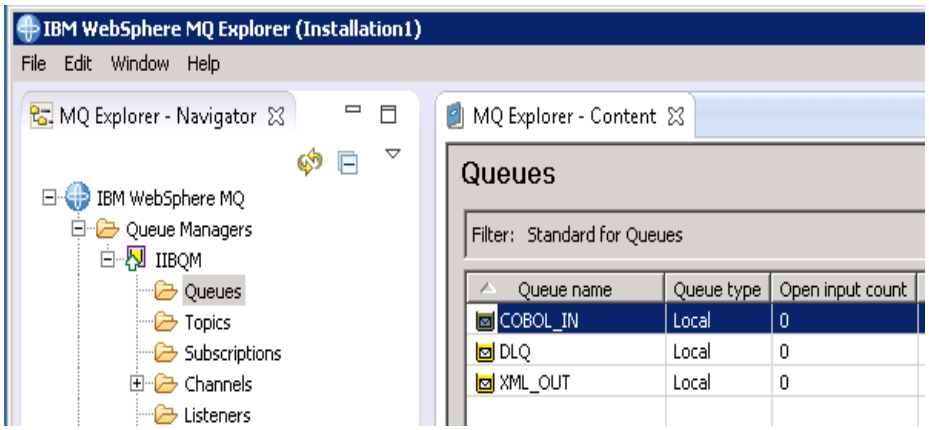
```
runmqsc IIBQM < C:\labfiles\Lab07-PDTools\CreateQueues.mqsc
```

__ 6. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the **MQ Explorer - Navigator** view, expand the **Queue Managers > IIBQM** folder.

__ b. Click the **Queues** folder the queue manager to display the **Queues** view.

__ c. Verify that the following queues are listed in the **Queues** view: COBOL_IN, DLQ, and XML_OUT



Note

You might have other queues for this queue manager from a previous exercise. You do not need to delete the unused queues.

Part 1: Extend a message flow with Trace node

In this part of the exercise, you import an application that is named **ProblemDetermination** and then add a Trace node.

The ProblemDetermination application contains:

- A DFDL schema that is named `COMPLAINT_IN.xsd` that defines the input file
- A message flow that is named `PD.msgflow`
- An ESQL file that is named `Transform_to_XML.esql`

__ 1. Import the project interchange file that is named `ProblemDeterminationApp_PI.zip` from the `C:\labfiles\Lab07-PDTools` directory into your workspace.

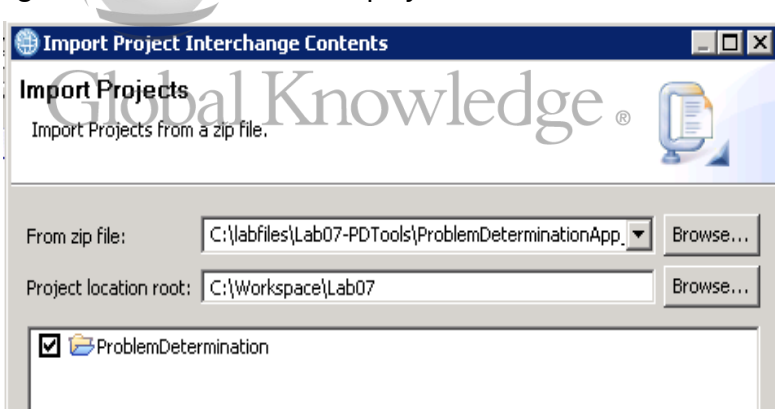
__ a. From the IBM Integration Toolkit, click **File > Import**.

__ b. Click **IBM Integration > Project Interchange** and then click **Next**.

__ c. To the right of **From zip file**, click **Browse**.

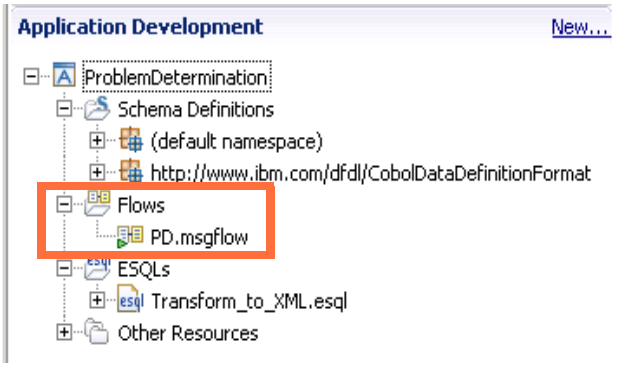
__ d. Browse to the `C:\labfiles\Lab07-PDTools` directory.

__ e. Select `ProblemDeterminationApp_PI.zip`, and then click **Open**. The **Import Project Interchange Contents** window is displayed.



__ f. Ensure that the **ProblemDetermination** application is selected and then click **Finish**.

__ 2. In the **Application Development** view, expand **Problem Determination > Flows**.

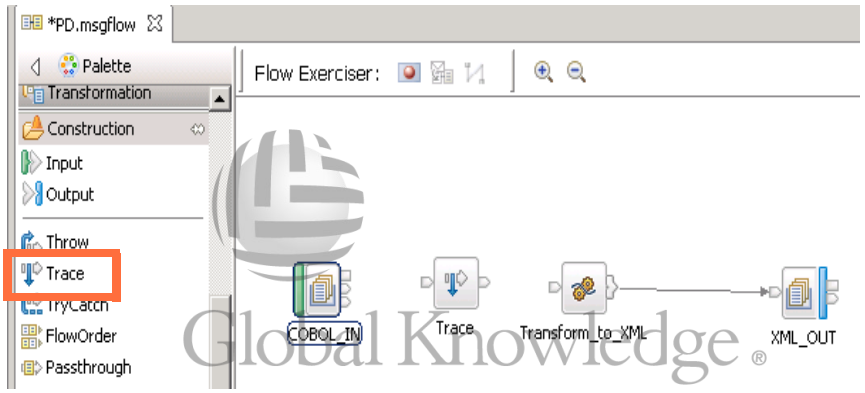


__ 3. Double-click the message flow **PD.msgflow** to open it in the Message Flow editor.

__ 4. Add a Trace node between the COBOL_IN MQInput node and the Transform_to_XML Compute node.

__ a. Delete the connection between the COBOL_IN MQInput node and the Transform_to_XML Compute node by selecting the connection and clicking **Delete**.

__ b. In the Message Flow editor Palette **Construction** drawer, click the Trace node, and then click between the COBOL_IN node and the Transform_to_XML node.



__ c. Connect the **Out** terminal of the COBOL_IN node to the **In** terminal of the Trace node.

__ d. Connect the **Out** terminal of the Trace node to the **In** terminal of the Transform_to_XML node.

__ 5. Configure the Trace node properties to output a time stamp and the entire root tree to a file.

__ a. Double-click the Trace node to display the node **Properties** view.

__ b. On the **Basic** tab, set **Destination** to **File**.

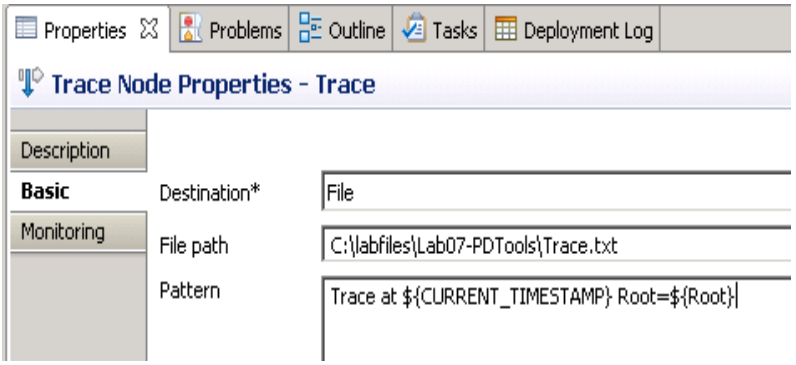
__ c. For the **File Path**, type: `C:\labfiles\Lab07-PDTools\Trace.txt`

Important

Verify that the directory that you specify in the Trace node exists. If the directory does not exist, the Trace node does not write any output, nor does it warn that it was not able to do so.

__ d. In the **Pattern** field, type: **`Trace at ${CURRENT_TIMESTAMP} Root=${Root}`**

The syntax is case-sensitive. A text file that is named `Cut&Paste.txt` in the `C:\labfiles\Lab07-PDTools` directory contains the text for the **Pattern** field. To ensure that the syntax is correct, you can copy the text from this file and paste it into the **Pattern** field.



__ 6. Set the properties on the MQInput node that is named COBOL_IN:

__ a. Verify that the **Queue name** property on the **Basic** tab is: `COBOL_IN`

__ b. Set the **Destination queue manager name** on the **MQ Connections** tab to: `IIBQM`

__ 7. Set the properties on the MQOutupt node that is named XML_OUT.

__ a. Verify that **Queue name** property on the **Basic** tab is: `XML_OUT`

__ b. Set the **Destination queue manager name** on the **MQ Connections** tab to: `IIBQM`

__ 8. Save the message flow.

__ 9. Verify that the ProblemDetermination application does not contain any errors.

__ a. Verify that there are no red flags next to any components in the project.

__ b. Verify that no problems are displayed in the **Problems** view.

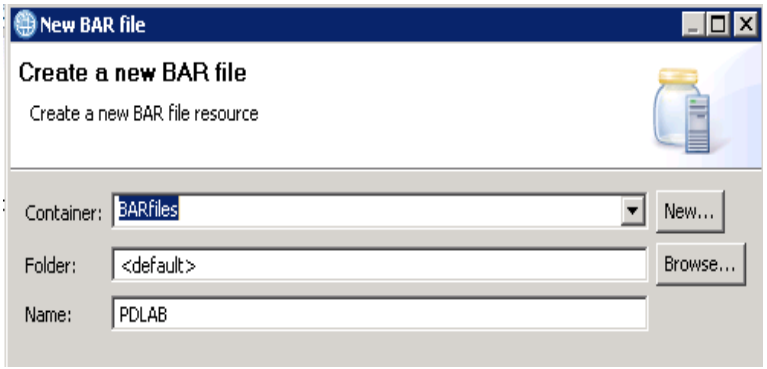
If any errors are indicated, you can review them in the **Problems** view. Correct any errors and then save and verify that no more errors are indicated in the project.

In most cases, warning messages can be ignored.

__ 10. Create a BAR file that is named **`PDLAB.bar`**.

__ a. In the IBM Integration Toolkit, click **File > New > BAR file**.

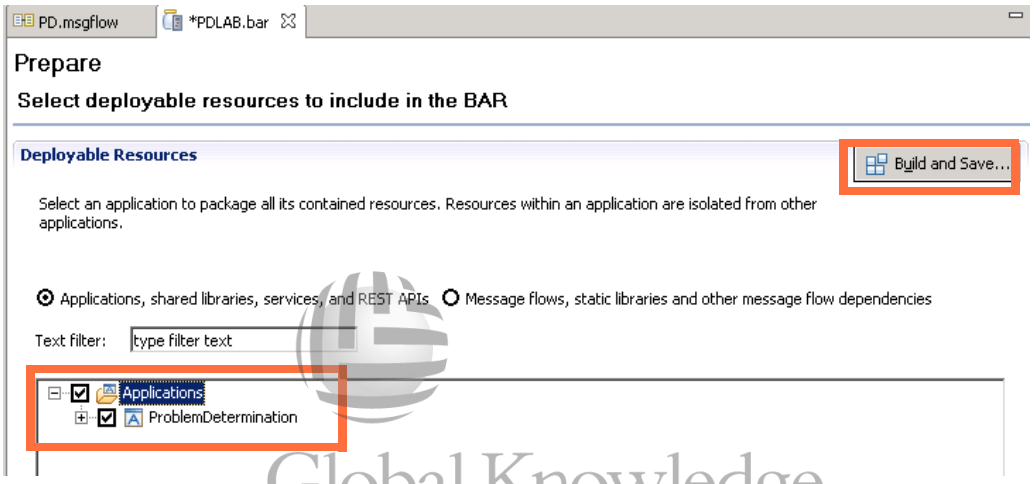
__ b. For the **Name** of the BAR file, type: PDLAB



__ c. Click **Finish**. The BAR File editor opens on the **Prepare** tab.

__ d. Select the components to include in the BAR file. In the **Deployable Resources** section, select the **ProblemDetermination** application.

__ e. Click **Build and Save**. The BAR file is built.

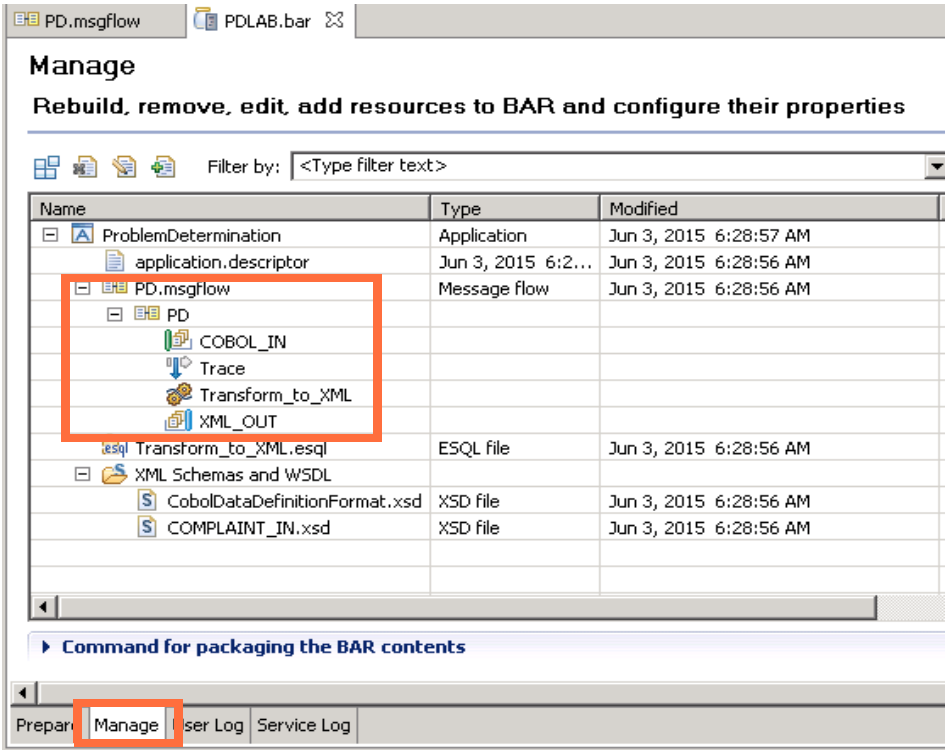


__ f. Click **OK** to the **Operation completed successfully** message.

__ g. In the **Application Development** view, verify that a folder named **BARs** was created, and that it contains the PDLAB.bar file.

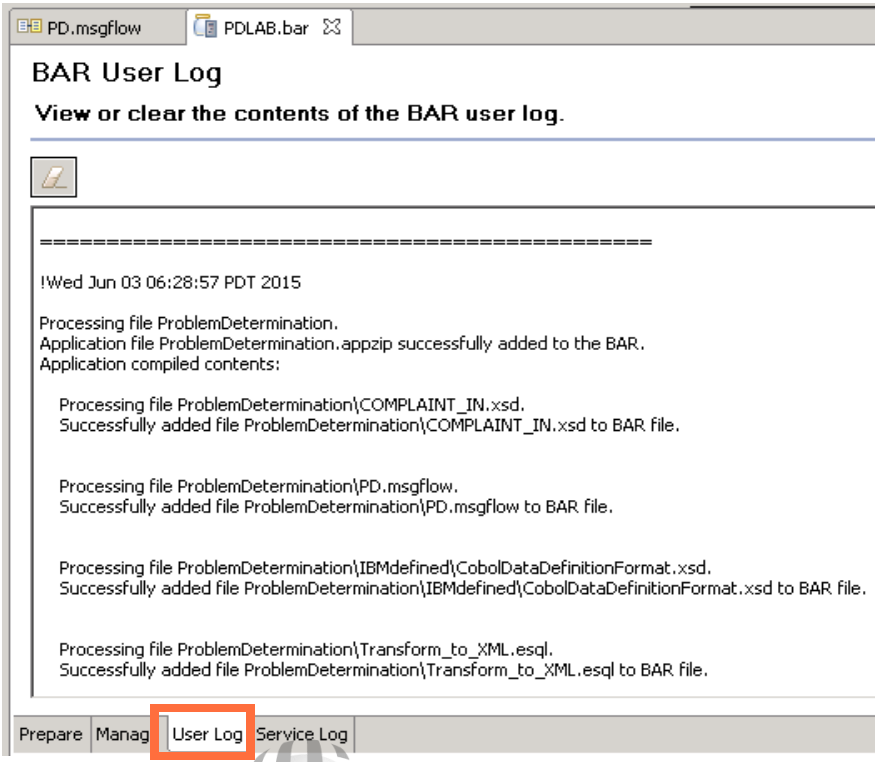
__ h. At the bottom of the BAR File editor window, click the **Manage** tab.

__ i. Verify that the `PD.msgflow`, `Transform_to_XML.esql`, `CobolDataDefinitionFormat.xsd`, and `COMPLAINT_IN.xsd` files are included in the BAR file and that they show current time stamps.



__ j. Expand the message flow file (`PD.msgflow`) in the Bar File editor to see the message flow nodes that are included in the message flow.

__ k. At the bottom of the BAR File editor, click the **User Log** tab. Review the messages that were generated during the creation of the BAR file. The messages indicate that the message flow was added to the BAR file. Again, in this instance, you can disregard any warning messages.



In most cases, if an object cannot be added to the BAR file, it is because there is an error in the message flow or the message model.

Any problems are also listed in the **Problems** view.

__ l. Correct any problems, and then save the application, to verify that no more errors are shown. After you fix all problems, build and save the BAR file again.

__ 11. In the **Integration Nodes** view, verify that the integration node TESTNODE_iibadmin is connected and running.

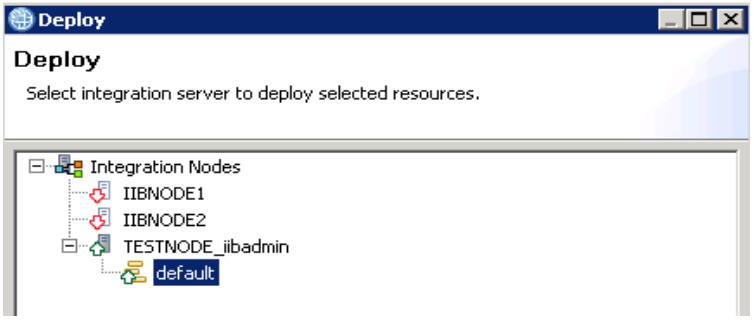
If the integration node is not running, right-click it and then click **Start**.

__ 12. Deploy the BAR file to the **default** integration server on TESTNODE_iibadmin.

__ a. In the **Application Development** view, expand the **BARs** folder (if it is not already expanded).

__ b. Right-click the PDLAB.bar file, and then click **Deploy**. The Deploy window is displayed.

__ c. Click **default** to select it as the integration server, and then click **Finish**.

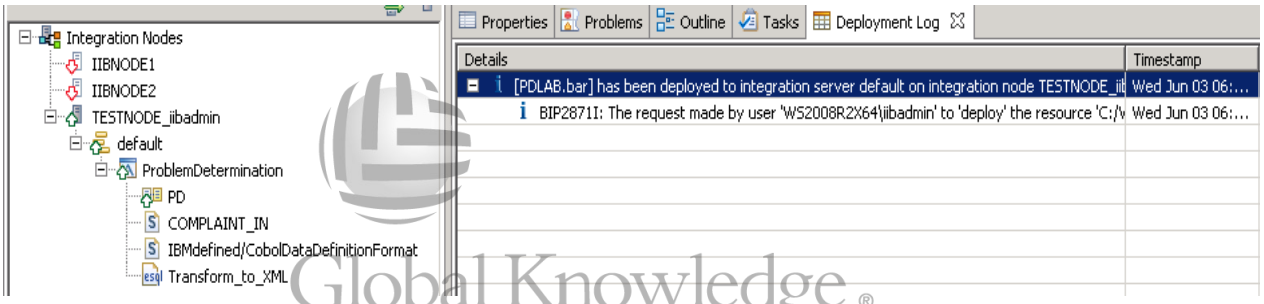


Information

You can also deploy a BAR file by dragging the BAR file from the **Application Development** view onto the integration server in the **Integration Nodes** view.

__ d. After a few seconds, the BAR file is deployed and the components are displayed under the integration server in the **Integration Nodes** view.

To ensure that the deployment operation completed successfully, check the deployment log in the **Deployment Log** view.



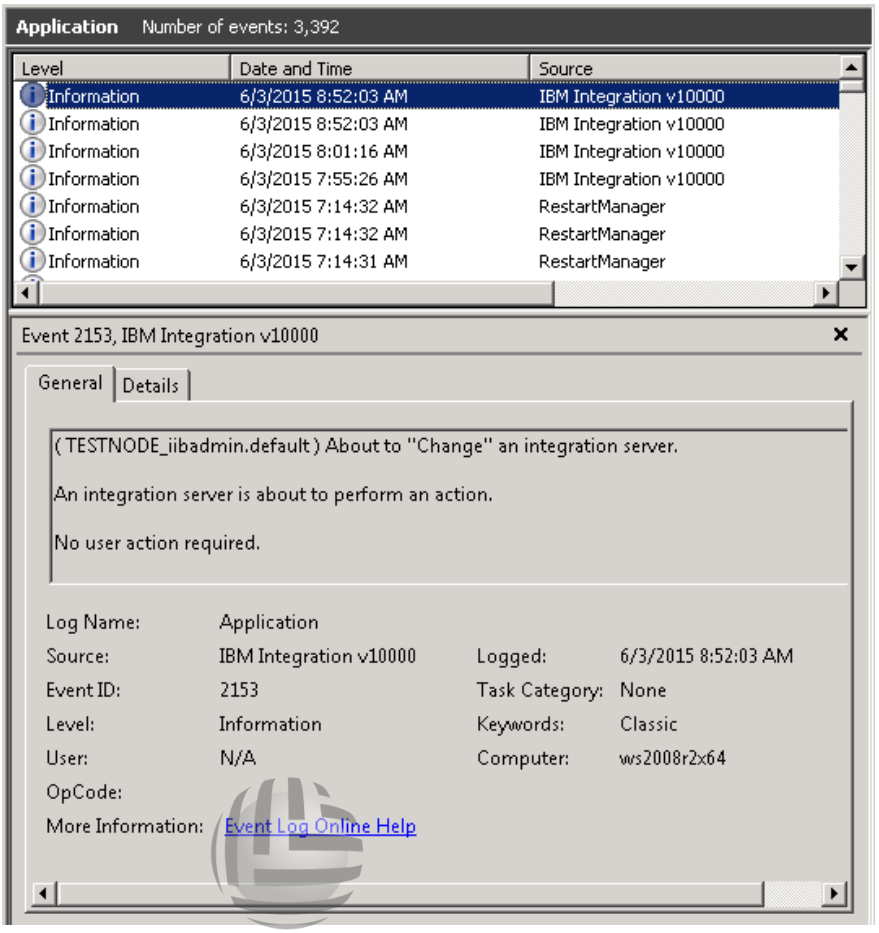
If the **Deployment Log** does not show any messages for the current date and time, then there might be a problem with the integration node.

__ 13. Check the local error log by using the Windows Event Viewer.

__ a. In Windows, click **Start > Administrative Tools > Event Viewer**.

__ b. Expand **Windows Logs**.

__ c. Select **Application**. Step through the messages and look at the deployment messages for IBM Integration Bus. Verify that the level for the deployment messages is **Information** and not **Error**.



__ d. Minimize the Event Viewer.

__ 14. Run the **mqsireporttrace** command to verify that Trace nodes are enabled on the **default** integration server and the **PD** message flow.

__ a. Open an IBM Integration Console window by double-clicking the IBM Integration Console shortcut icon on the desktop or by clicking **Start > IBM Integration Bus 10.0.0.0 > IBM Integration Console 10.0.0.0**.

__ b. In the IBM Integration Bus console window, type:

```
mqsireporttrace TESTNODE_iibadmin -n -e default
```

The results should indicate that the Trace node is “on” for the integration server that is named **default** and the **ProblemDetermination/PD** message flow.

Information

You can also check the status of the Trace nodes at the integration server level by using the IBM Integration web user interface.

__ 15. Use the RFHUtil program to send test data to the input queue and cause the message flow to run.

__ a. On the Windows desktop, double-click the RFHUtil shortcut icon on the desktop.

You can also start RFHUtil by using Windows Explorer to browse to the `C:\Software\ih03` directory and then double-clicking `rfhutil.exe`.

RFHUtil opens on the **Main** tab.

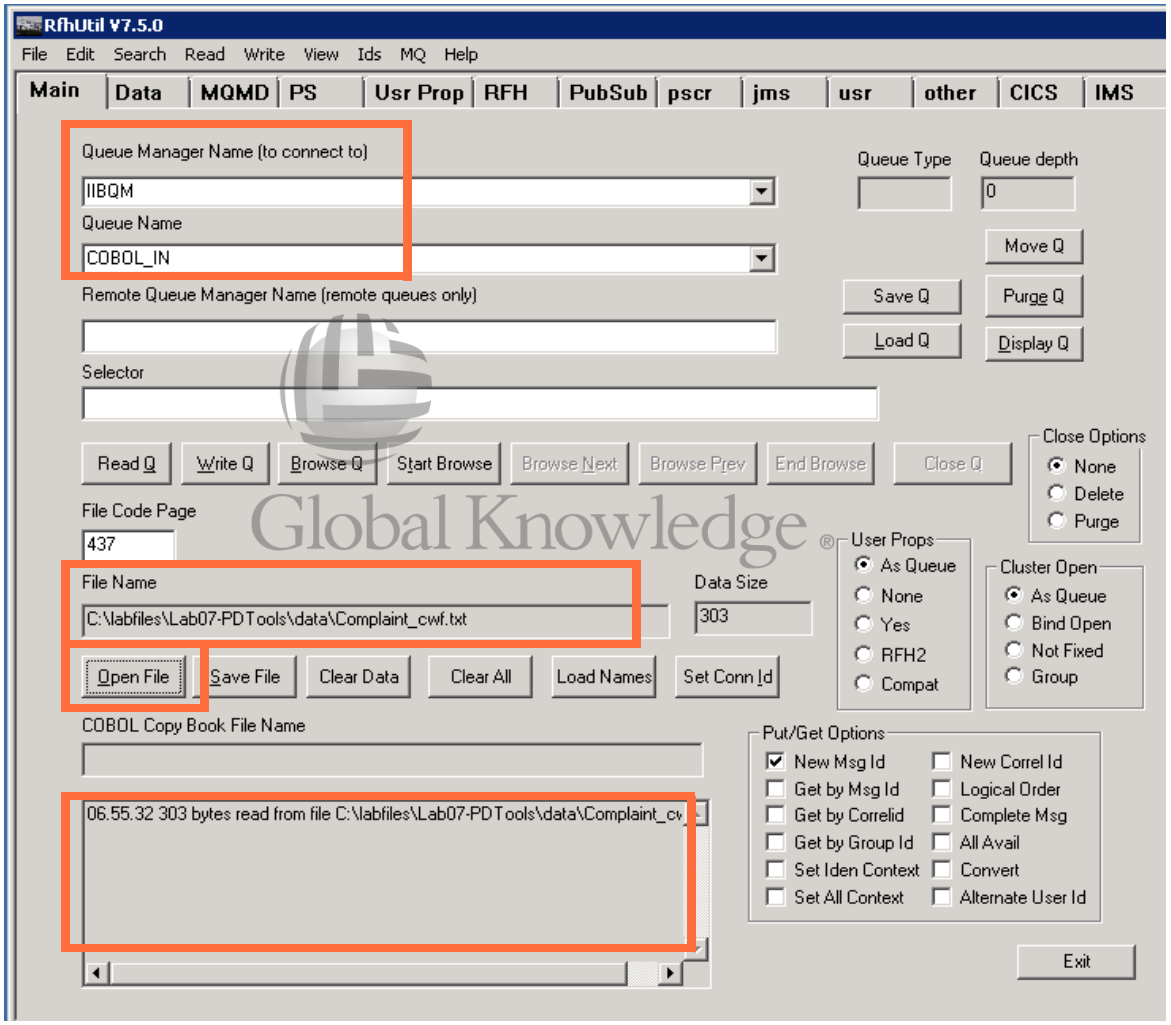
__ b. For **Queue Manager Name**, select **IIBQM**.

__ c. For **Queue Name**, select **COBOL_IN**.

__ d. Click **Open File**.

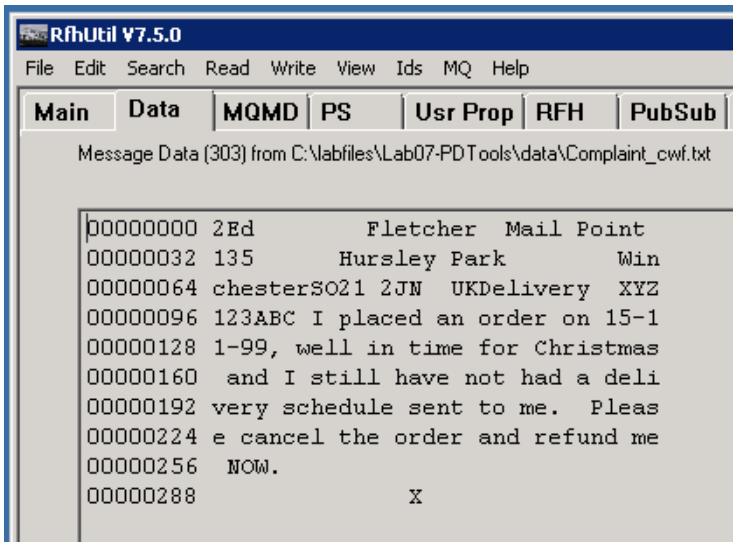
__ e. Select `C:\labfiles\Lab07-PDTools\data\Complaint_cwf.txt` and click **Open**.

The message window displays a time stamp with information about the data read from the file.



__ f. Use RFHUtil to review the test data.

Click the **Data** tab. The data is presented in character format by default.



__ g. On the **Main** tab, click **Write Q** to send the message to the COBOL_IN queue. You should see an entry in the status window on the **Main** tab that a message was sent to the COBOL_IN queue.

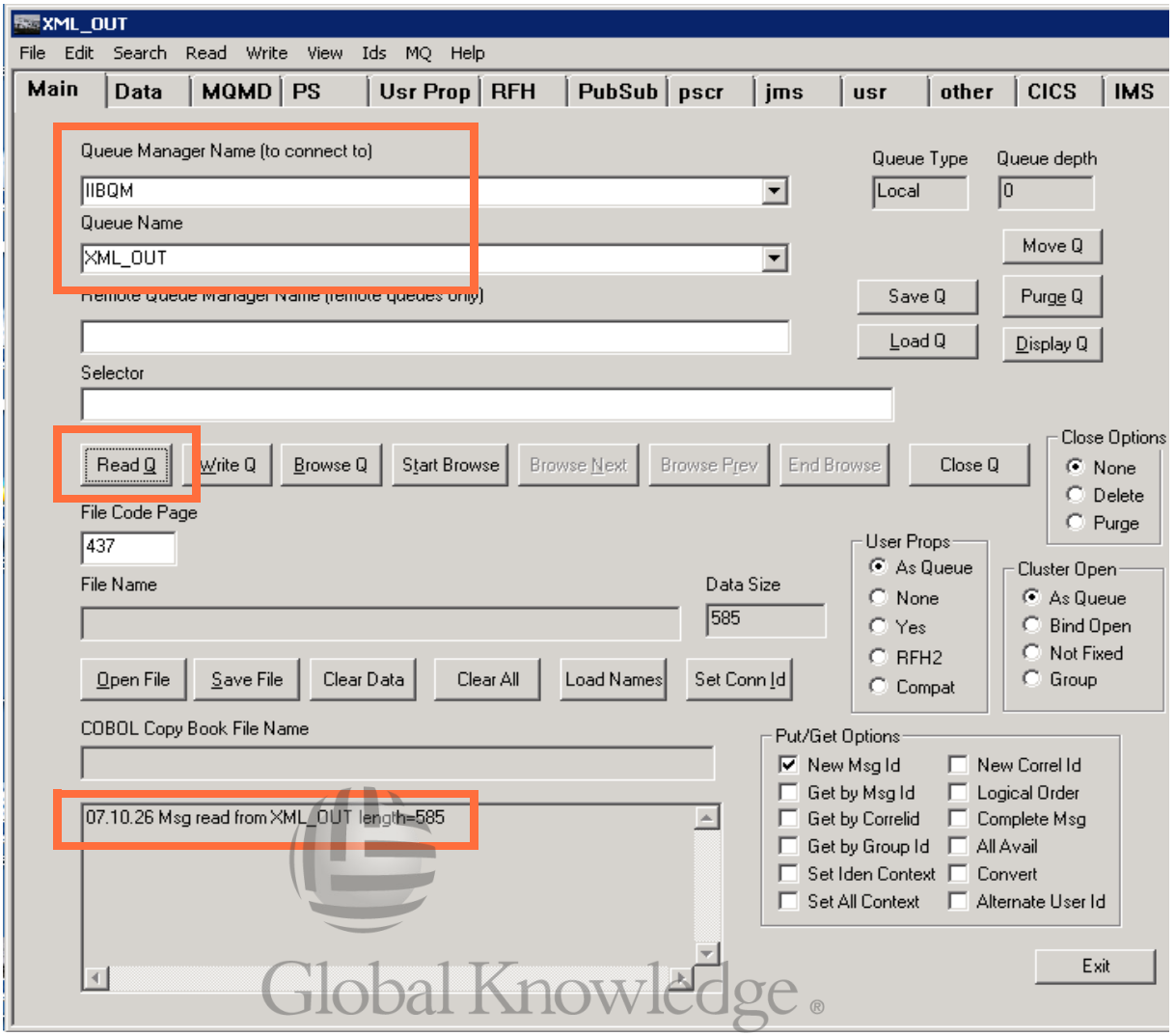
__ 16. Review the output message on queue XML_OUT by using RFHUtil.

__ a. Start a second instance of RFHUtil.

__ b. For the **Queue Manager Name**, select **IIBQM**.

__ c. For **Queue Name**, select **XML_OUT**

__ d. Click **Read Q**. The message is read from the queue, and RFHUtil displays information about the message that was read.

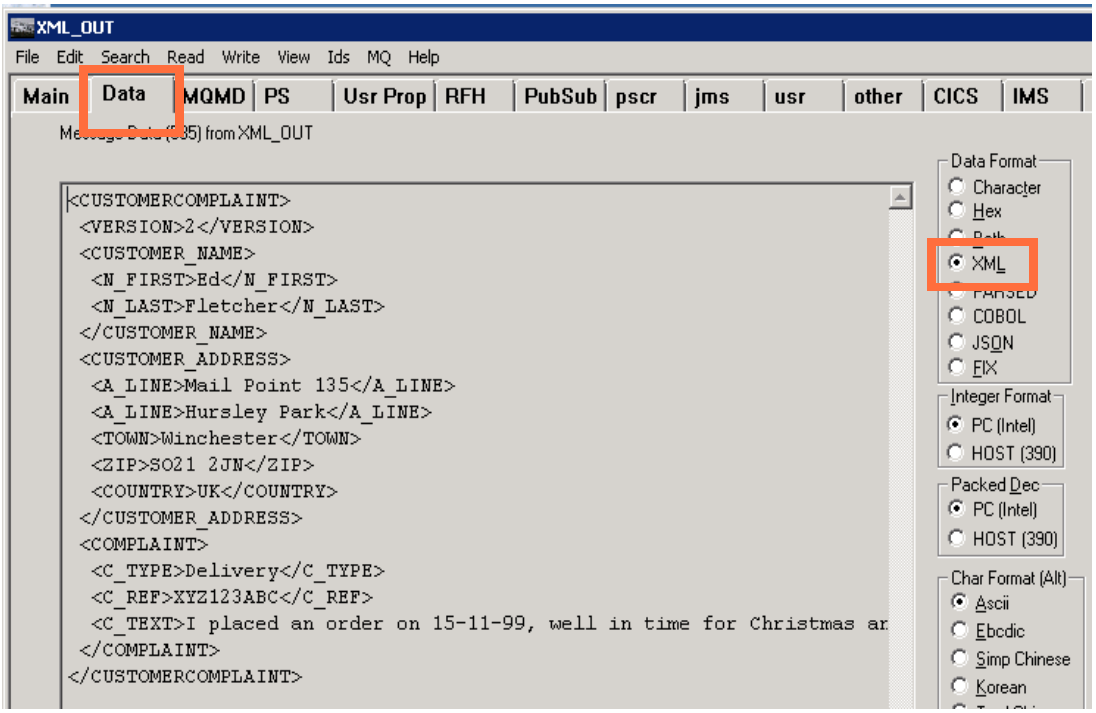


Information

Read Q sends an MQGET, which removes the message from the queue. If you want to see the contents of the first message in the queue without removing it from the queue, use the **Browse Q** option instead.

__ e. Click the **Data** tab to view the message data.

The message is an XML message. To view the message as XML, click **XML** under the **Data Format** section.



__ 17. Review the output from Trace node.

__ a. Using Windows Explorer, browse to file `C:\labfiles\Lab07-PDTools` directory and verify that the trace file (`Trace.txt`) was created.

If you cannot find the Trace file, verify the settings in the Trace node **Properties**.

__ b. Double-click the file to open it with the default editor (Notepad).

__ c. Review the trace file contents (the first few lines of the file are shown here). You should see the text that you included in the trace node configuration with the contents of the ESQL variables CURRENT_TIMESTAMP and ROOT.

```
Trace at 2015-06-03 07:02:15.967855 Root=( ['MQROOT' : 0x2c3d8bf0]
  (0x01000000:Name):Properties = ( ['MQPROPERTYPARSER' : 0x28ca59d0]
    (0x03000000:NameValue):MessageSet              = '' (CHARACTER)
    (0x03000000:NameValue):MessageType             = '{}:CUSTOMERCOMPLAINT'
(CHARACTER)
    (0x03000000:NameValue):MessageFormat           = '' (CHARACTER)
    (0x03000000:NameValue):Encoding                = 546 (INTEGER)
    (0x03000000:NameValue):CodedCharSetId          = 437 (INTEGER)
    (0x03000000:NameValue):Transactional           = TRUE (BOOLEAN)
    (0x03000000:NameValue):Persistence             = FALSE (BOOLEAN)
    (0x03000000:NameValue):CreationTime           = GMTTIMESTAMP '2015-06-03
14:02:15.940' (GMTTIMESTAMP)
    (0x03000000:NameValue):ExpirationTime          = -1 (INTEGER)
    (0x03000000:NameValue):Priority                = 0 (INTEGER)
    (0x03000000:NameValue):ReplyIdentifier         =
X'000000000000000000000000000000000000000000000000' (BLOB)
    (0x03000000:NameValue):ReplyProtocol           = 'MQ' (CHARACTER)
    (0x03000000:NameValue):Topic                   = NULL
    (0x03000000:NameValue):ContentType             = '' (CHARACTER)
    (0x03000000:NameValue):IdentitySourceType      = '' (CHARACTER)
    (0x03000000:NameValue):IdentitySourceToken     = '' (CHARACTER)
```

__ 18. The Trace node requires extra system resources. Now that you are finished by using the Trace node, turn it off.

You can turn off the Trace nodes at the integration server level or for a specific message flow. In this exercise, you turn off the Trace node at the message flow level by providing the message flow name (**-f**) and application name (**-k**) in the command.

__ a. In the IBM Integration Console window, type:

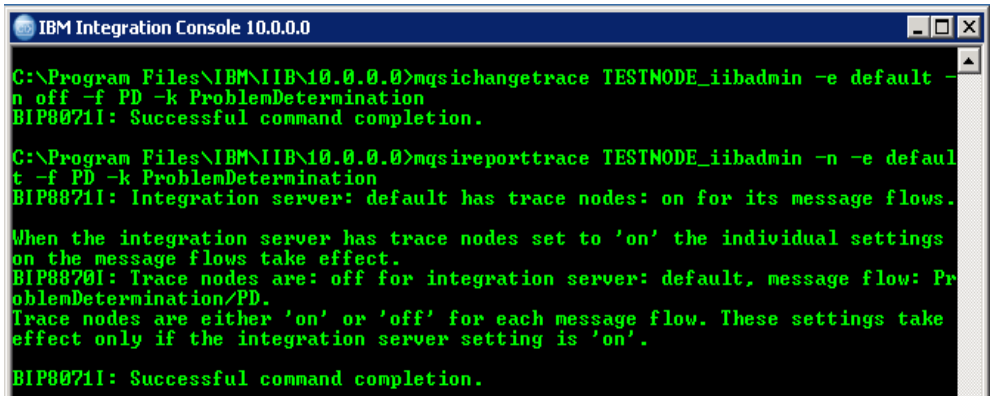
```
mqsichangetrace TESTNODE_iibadmin -e default -n off -f PD
-k ProblemDetermination
```

You should receive a message that the command completed successfully.

__ b. Run the **mqsireporttrace** command to verify that Trace node is now off.

```
mqsireporttrace TESTNODE_iibadmin -n -e default -f PD
-k ProblemDetermination
```

The result message should indicate that Trace nodes are off for the message flow PD on the **default** integration server.



Part 2: Run the flow with User Trace

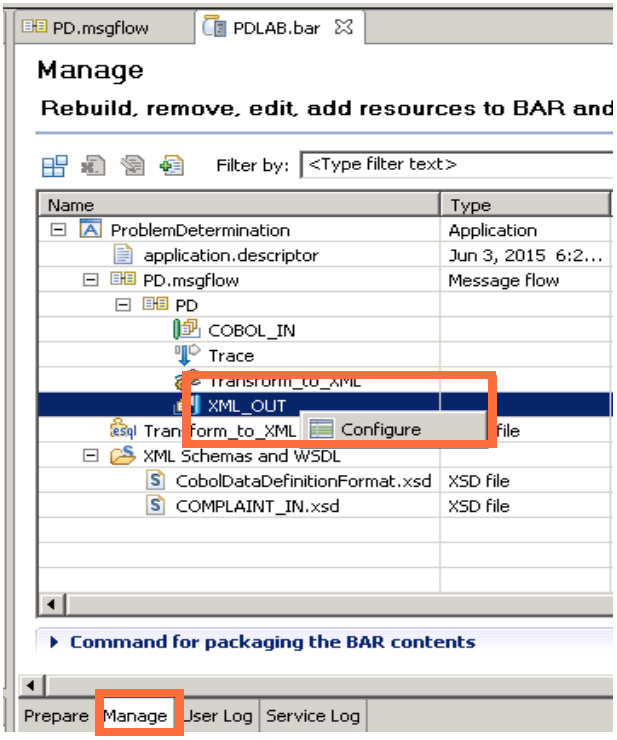
In this part of the exercise, you run the message flow with an invalid queue name for the MQOutput node that is named XML_OUT. This action causes a runtime error, which you diagnose by using a User Trace and the Event Viewer.

__ 1. Modify the BAR file to specify an output queue that does not exist to generate a runtime error when the flow runs.

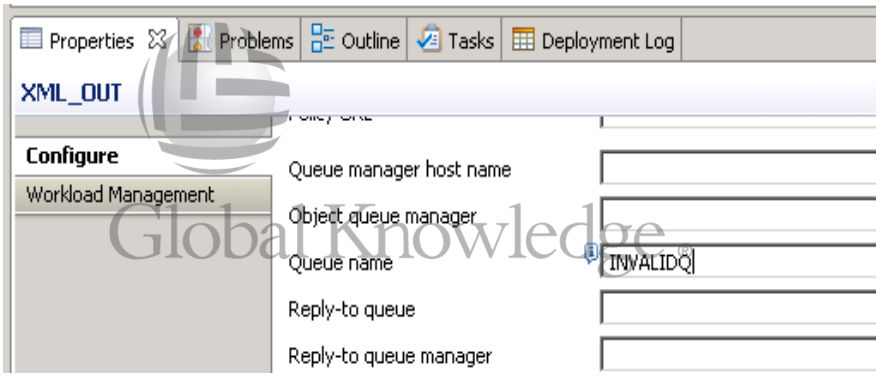
__ a. If the **PDLAB.bar** file is not already open, double-click the **PDLAB.bar** file in the **Application Development** view to open it in the BAR File editor.

__ b. On the **Manage** tab, expand **ProblemDetermination > PD.msgflow > PD** to display the message flow nodes.

__ c. Right-click **XML_OUT** and then click **Configure**.



__ d. In the **Properties** tab, type `INVALIDQ` for the **Queue name**.



__ e. Save the BAR file.

__ 2. Redeploy the modified BAR file to the **default** integration server.

__ 3. Enable a user trace at the debug level on the **default** integration server on TESTNODE_iibadmin.

__ a. In the IBM Integration Console window, type:

```
mqsichangetrace TESTNODE_iibadmin -u -e default -l debug
```

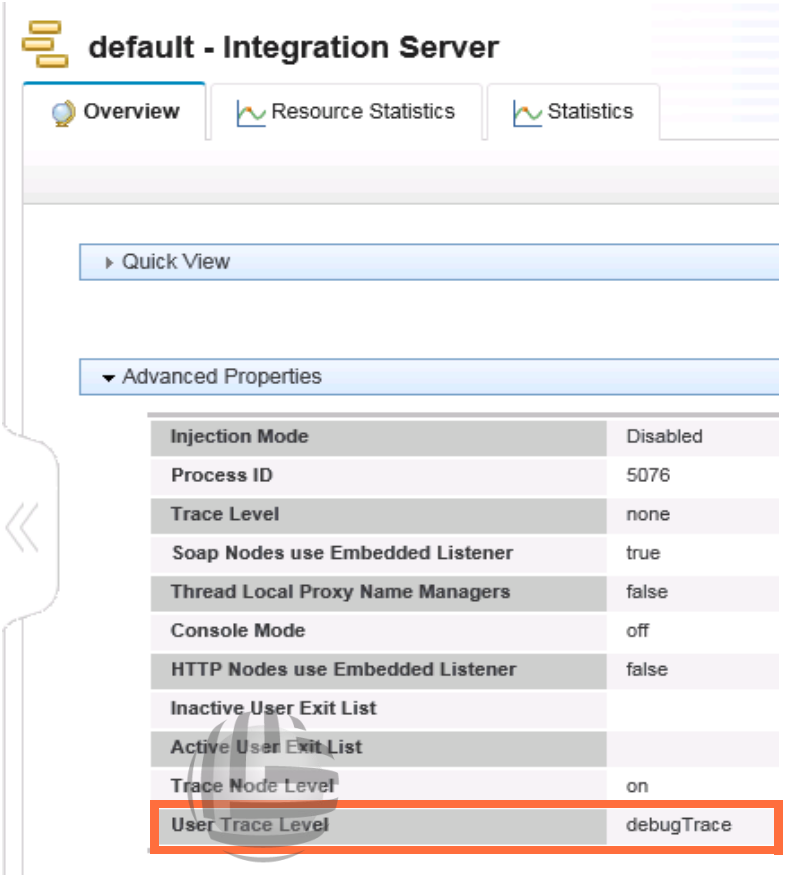
You should see a message that the command was successful.

__ b. Verify that the user trace is active. In the IBM Integration Console window, type:

```
mqsireporttrace TESTNODE_iibadmin -u -e default
```

Information

You can also view the status of a User Trace for the integration server by using the IBM Integration web user interface.



The IBM Integration web user interface **Monitoring > Admin Log** view shows that the user trace status is updated to “debugTrace”.

Message	Source	Timestamp	Message Detail
BIP2880I	Change Notification	2015-06-03 08:54:35.347 Pacific Daylight Time	The property 'This/userTraceLevel' has changed from 'none' to 'debugTrace' on object 'default' of type 'ExecutionGroup' with parent 'TESTNODE_iibadmin' of type 'Broker'.
BIP2884I	Change Notification	2015-06-03 08:52:06.953 Pacific Daylight Time	The resource 'ProblemDetermination' of type 'Application' was redeployed to the integration server 'default' on integration node 'TESTNODE_iibadmin'.

__ 4. Using the RFHUtil session that you already configured to put a message to COBOL_IN, send a test message by clicking **Write Q**.

The message flow fails because the output queue does not exist. Instead, the message goes to the queue manager’s dead-letter queue (DLQ) for the following reasons:

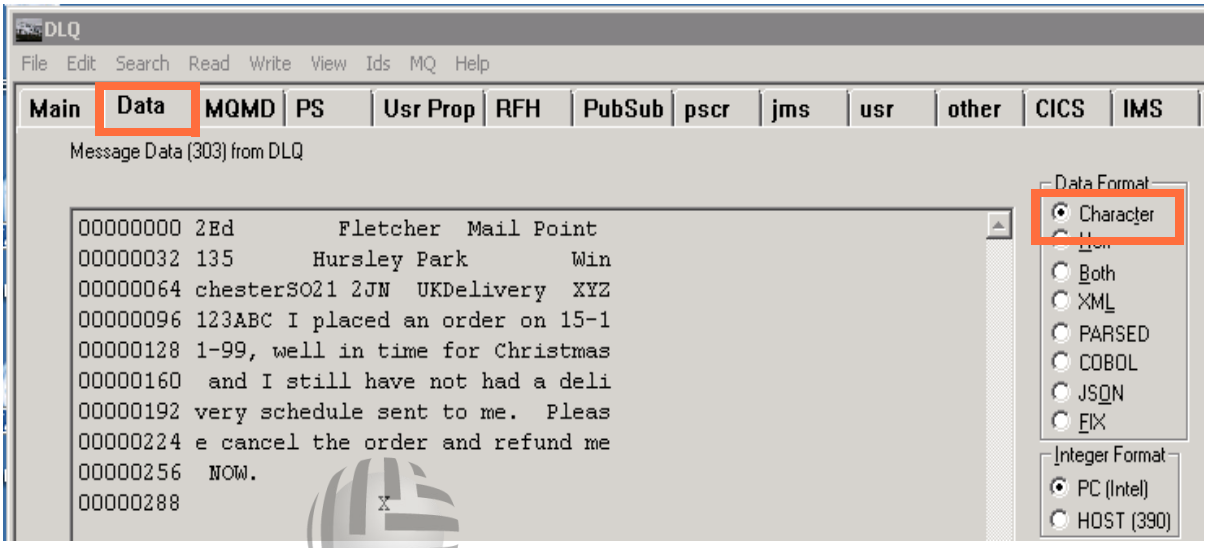
- The flow is transactional (default mode for the MQInput node).
- The **Catch** and **Failure** terminals are not wired on the COBOL_IN MQinput node.
- No back-out queue is configured for the queue COBOL_IN.

__ 5. Using the second instance of RFHutil, verify that the message is on the queue manager's dead-letter queue.

__ a. On the **Main** tab, change the **Queue Name** to **DLQ**.

__ b. Click **Browse Q**.

__ c. On the **Data** tab, change the **Data Format** to **Character** to verify that the original input message is now on the dead-letter queue (DLQ).



Information

You can also use the IBM MQ Explorer to verify that the message is on the queue that is named DLQ.

__ 6. Review the user trace.

__ a. Run the command to read the log and convert it to an XML file that is named `UserTrace.xml` and save it in the `C:\labfiles` directory.

In the IBM Integration Console window, type:

```
mqsireadlog TESTNODE_iibadmin -u -e default -f -o C:\labfiles\UserTrace.xml
```

__ b. Convert the user trace file from XML to a plain text file.

In the IBM Integration Console window, type:

```
mqsiformatlog -i C:\labfiles\UserTrace.xml -o C:\labfiles\UserTrace.txt
```

__ c. Open the user trace file `UserTrace.txt` with Windows Notepad session and review the contents.

__ d. To find the error, search for the string MessageException. This line gives you information about the error and the 4-digit IBM MQ reason code.

An excerpt of the message is shown here.

```
MessageException  BIP2666E: An error occurred when opening queue
''INVALIDQ'' on destination queue manager ''IIBQM''. State = '-1'
''MQW101'' '2085' ''''
An error occurred when opening a queue. The reason code from the MQOPEN is
displayed as the 3rd (native error) state.
Check the WebSphere MQ completion and reason codes in the WebSphere
MQApplication Programming Reference manual to establish the cause of the
error, taking any appropriate action.  It may be necessary to restart the
integration node after you have performed this recovery action.
```

__ e. When you are done reviewing the messages, close Notepad.

__ 7. To find out what that IBM MQ reason code means, type the following command in the IBM Integration Console window:

mqrc 2085

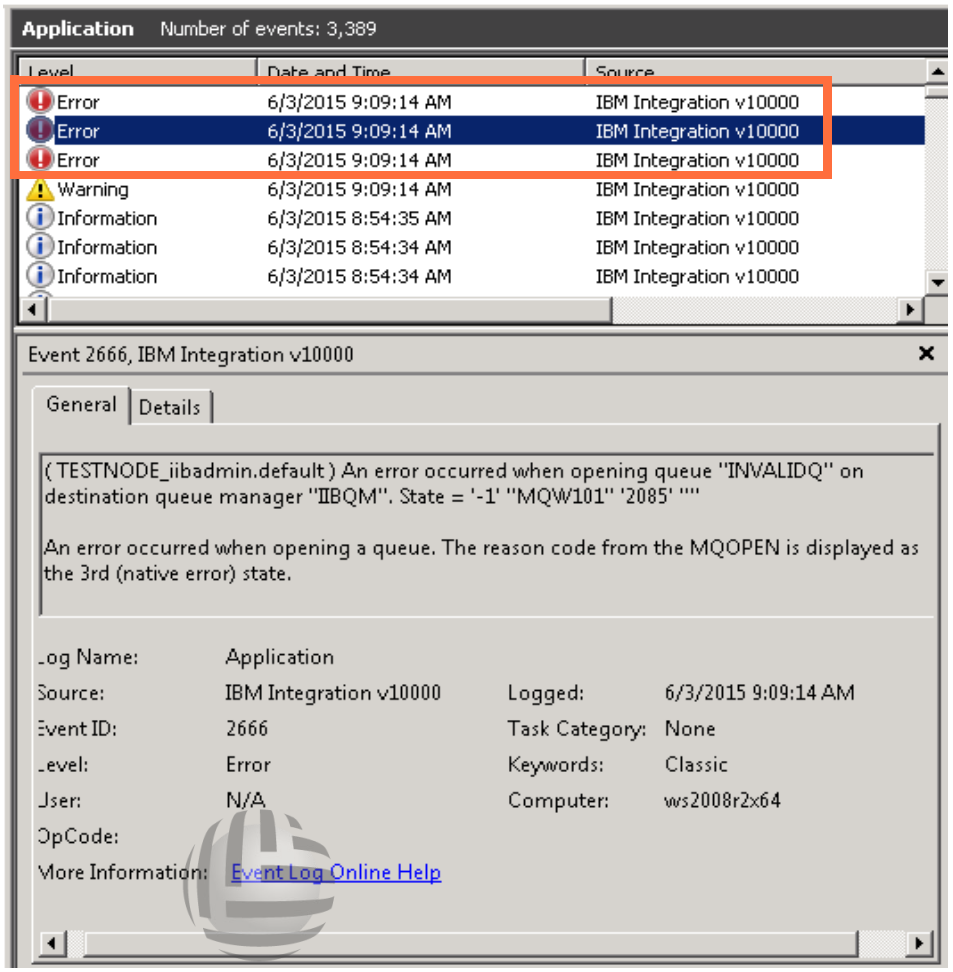
The reason code MQRC_UNKNOWN_OBJECT_NAME indicates that a non-existent output queue name (INVALIDQ) was specified in the message flow.

__ 8. Use the Windows Event Viewer to view the error.

__ a. If it is not already open, open the Windows Event Viewer.

If the Windows Event Viewer is already open, click **Refresh** to update the event log.

__ b. On the **Windows Logs > Application Logs** view, you should see some entries for IBM Integration Bus that are marked with **Error**.



__ c. Double-click the error message that has the earliest time stamp to see the **Event Properties** details.

__ d. Review the Description to determine the cause of the error.

__ e. Subsequent error messages often provide more information about the error, or about processing that took place in response to the error.

Review the other messages that are marked as **Error**.

__ f. Close the Event Viewer.

__ 9. Turn off user trace. In the IBM Integration Console window, type:

```
mqsichangetrace TESTNODE_iibadmin -u -e default -l none
```

You should see a message that the command was successful.

__ 10. Verify that the user trace is active. In the IBM Integration Console window, type:

```
mqsireporttrace TESTNODE_iibadmin -u -e default
```

You should see that the user trace level is now set to “none”.

Part 3: Use the Message Flow Debugger

In this part of the exercise, you configure the Message Flow Debugger in the IBM Integration Toolkit and use it for problem determination.

__ 1. Set the debug port for the default integration server in IBM Integration Toolkit.

__ a. Right-click the **default** integration server on the **Integration Nodes** view and then click **Launch Debugger**.

__ b. You receive a message that indicates that the debug port is not set. Click **Configure**.

__ c. Enter 2311 as the **Flow Debug Port**, and then click **OK**.

The integration server must be restarted before the property change takes effect. The restart happens automatically when you click **OK**.

You might also notice the status icon in front of the **default** installation server change from started (green upward-pointing arrow) to stopped (red downward-pointing arrow) and back to started.

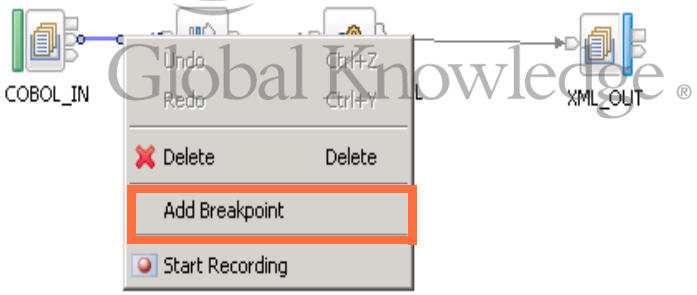
You can also view the change to the integration server in the **Administration Log** in the IBM Integration web user interface.

__ d. After the installation server restarts, you receive a confirmation message that the debugger is started on port 2311. Click **OK**.

__ 2. Set breakpoints in the message flow and ESQL program.

__ a. If it is not already open, open the `PD.msgflow` file in the Message Flow editor.

__ b. Right-click the connection between the COBOL_IN MQInput node and the Trace node and then click **Add Breakpoint**.

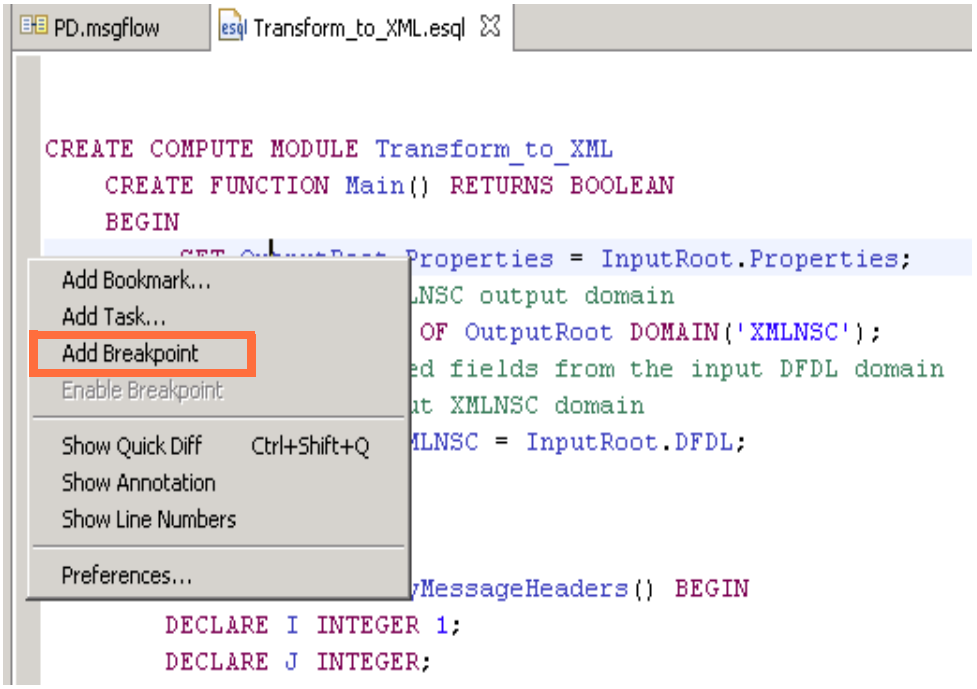


A blue circle is displayed on the wire, indicating that a breakpoint is set.

__ c. Double-click the Transform_to_XML Compute node to open the ESQL editor.

__ d. Click the line `SET OutputRoot.Properties = InputRoot.Properties;` to highlight it.

__ e. Right-click in the gray margin to the left of the line of code, and then click **Add Breakpoint**.

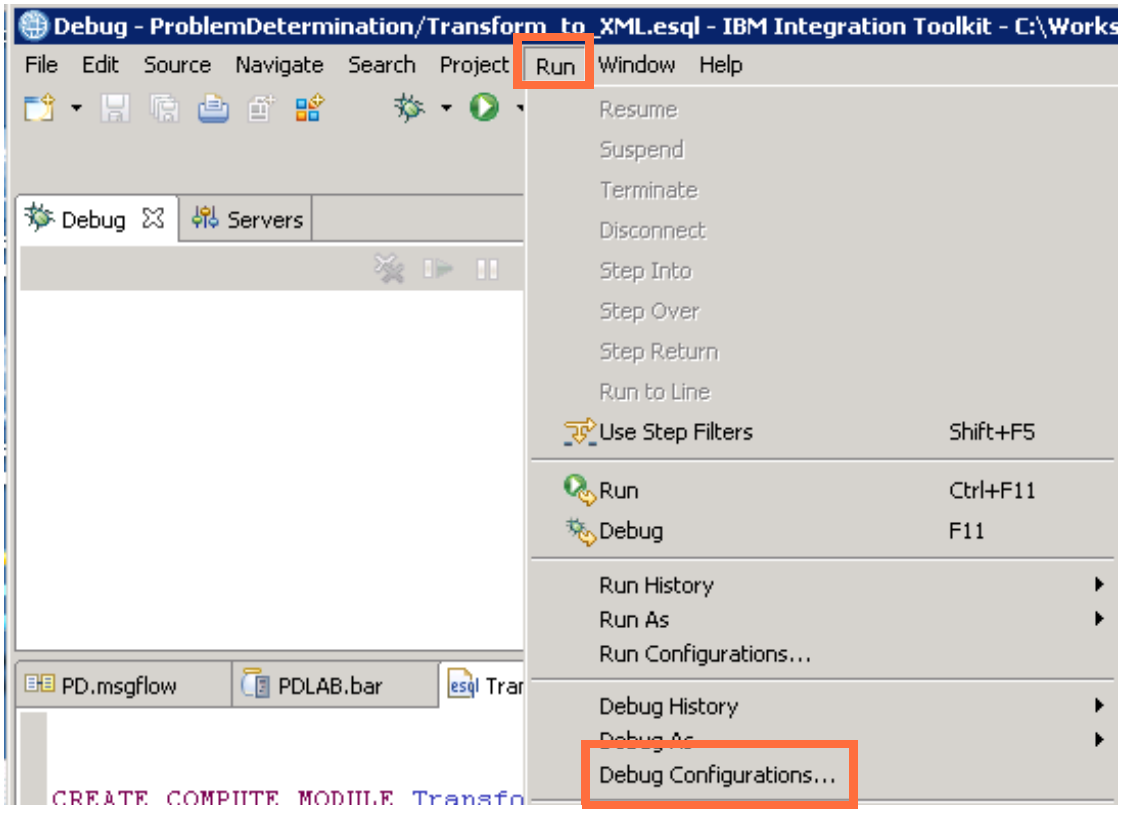


A blue circle is displayed in the margin, indicating that a breakpoint is set.

__ 3. In the IBM Integration Toolkit Debug perspective, configure the debugger, and attach it to the **default** integration server.

__ a. Click **Window > Open Perspective > Debug** and then click **OK**. The Debug perspective opens.

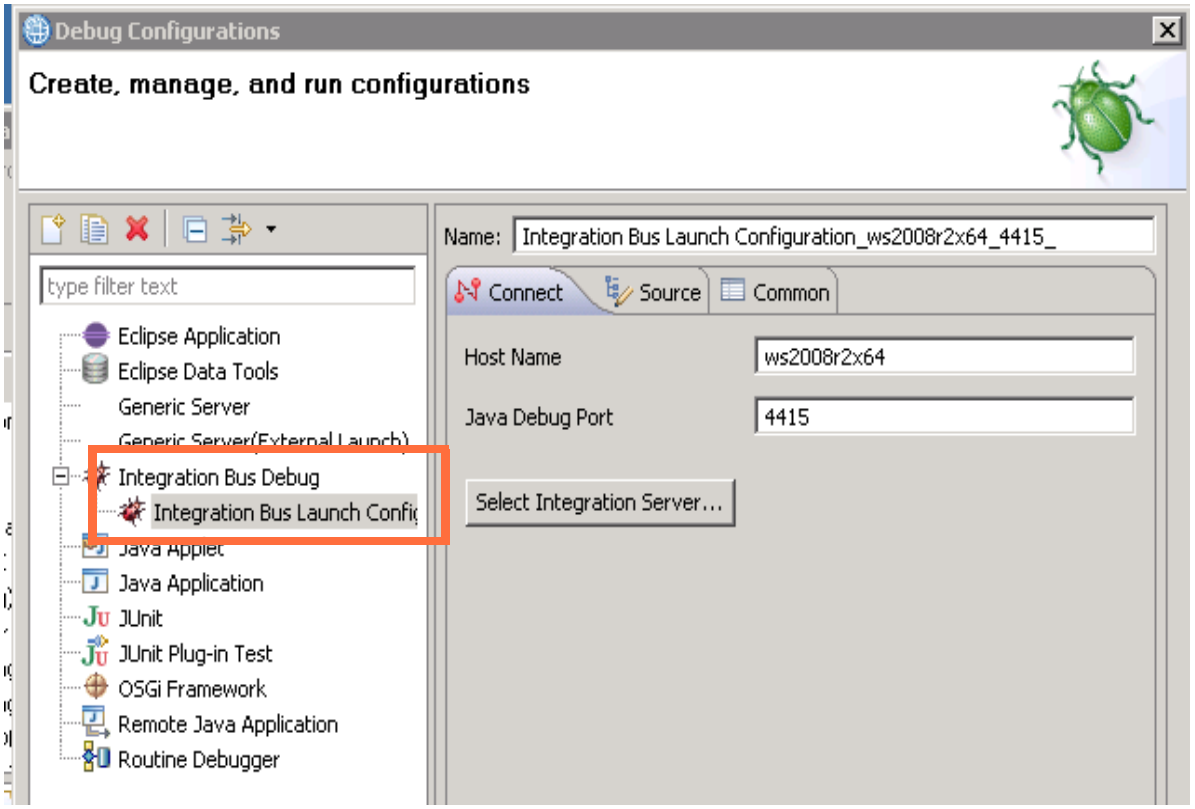
__ b. In the Debug perspective, click **Run > Debug Configurations**.



The **Debug Configurations** window is displayed.

__ c. Because you are using the Debugger for the first time, you must create an IBM Integration Bus debug configuration.

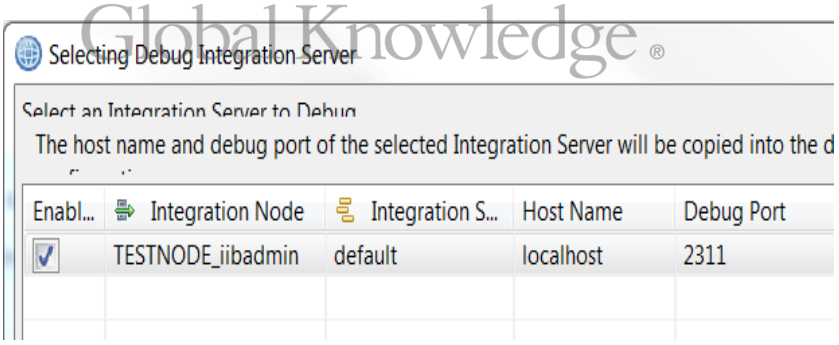
Click **Integration Bus Debug > Integration Bus Launch Configuration**.



__ d. Click **Select Integration Server**.

__ e. Verify that **Enabled** is selected for the row in the table for integration node TESTNODE_iibadmin with integration server **default** and that the port is 2311.

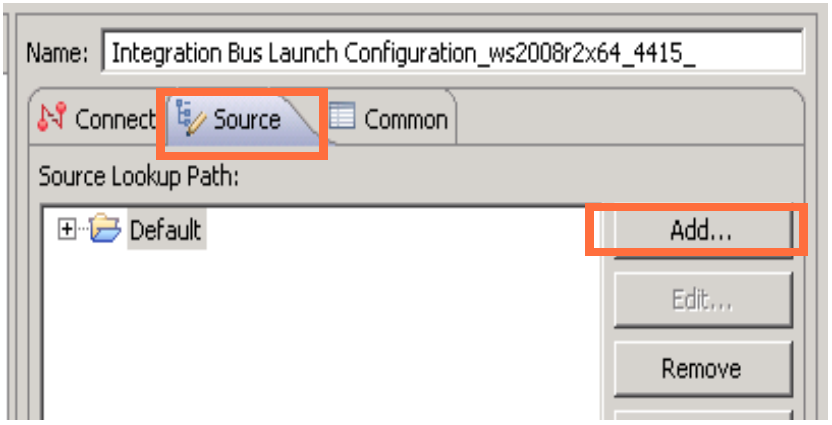
Click **Cancel**.



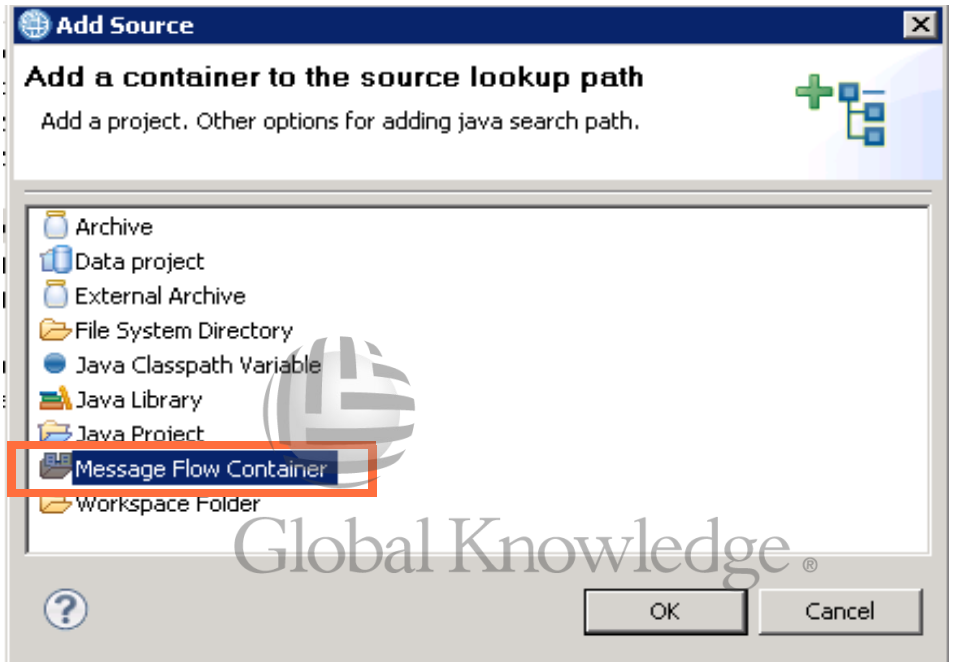
__ 4. To display the source code during debugging, you must identify the location of the source code to the debugger.

__ a. Click the **Source** tab.

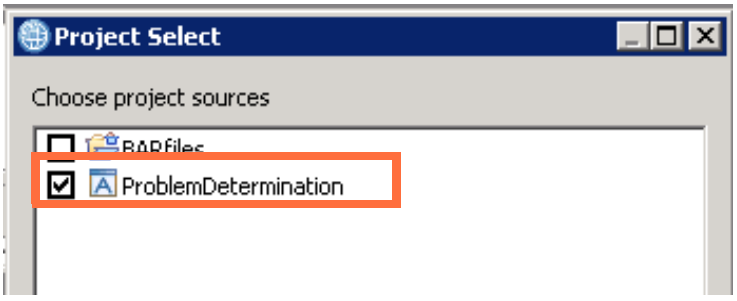
__ b. Click **Add**.



__ c. On the **Add Source** window, click **Message Flow Container**, and then click **OK**.



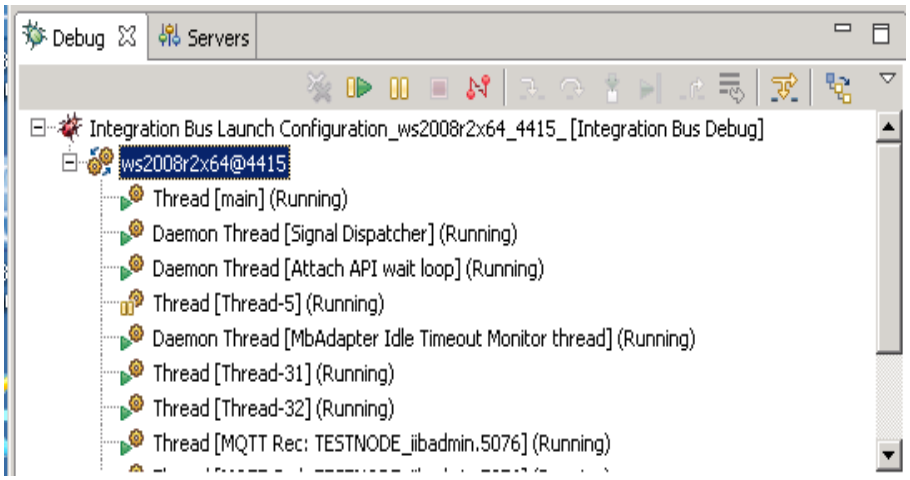
__ d. On the **Project Select** window, select **ProblemDetermination**, and then click **OK**.



__ e. Click **Apply**.

__ f. Click **Close**.

__ 5. The debugging session starts. The name of the debug session is shown in the **Debug** view with the host name and debug port (4415).

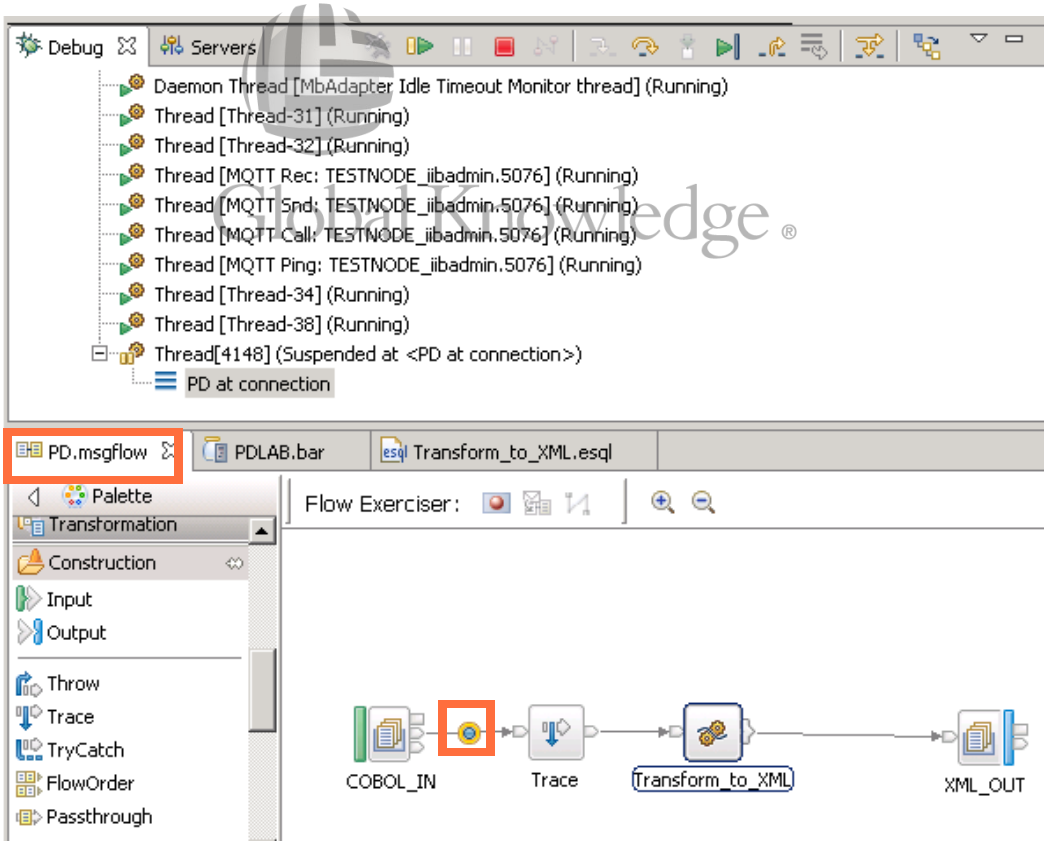


Your screen might not exactly match the screen capture that is shown in this exercise.

__ 6. Using the RFHUtil session that is already running for queue COBOL_IN, send a test message by clicking **Write Q**.

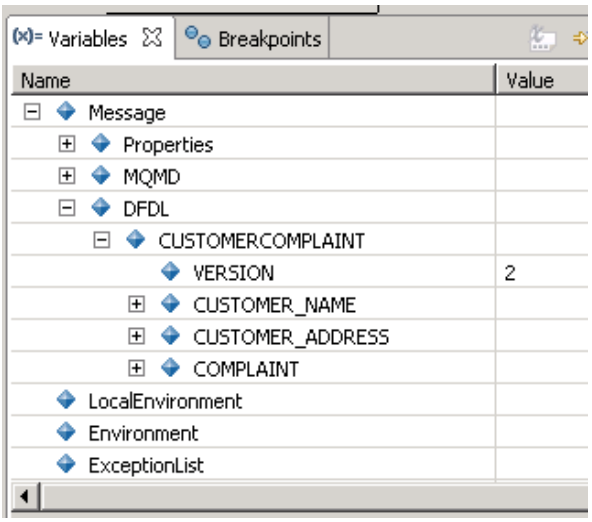
__ 7. The **Debug** view shows that the message flow is suspended.

The message flow is shown on the **PD.msgflow** tab, with the breakpoint that paused the flow. The yellow circle around the breakpoint indicator shows that it is the current breakpoint at which flow is halted.



__ 8. The **Variables** view now shows the elements of the message assembly: Message, Local Environment, Environment, and the Exception List.

__ 9. Expand **Message > DFDL** and a few of the subordinate components such as CUSTOMER_NAME and CUSTOMER_ADDRESS.



__ 10. It is possible to modify the value of a message field while running a message flow in the Debug perspective. Change the value of the VERSION field in the message.

__ a. In the **Variables** view, expand the Message tree until you see the VERSION field.

__ b. Click the VERSION **Value** column and type over its current value (2) with another value.

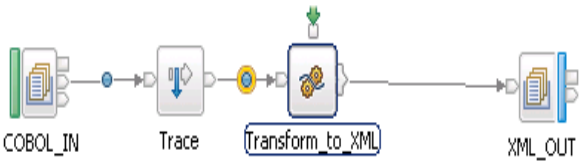
Note

In this exercise, modifying this field does not change the runtime results. This step in the exercise is intended to show that you can change a value during debugging when necessary.

__ 11. Step through the flow.

__ a. Use the Step icons in the Debug view toolbar or right-click anywhere in the Debug view and click **Step over**. You can also press F6 to step over.

Step over pauses the debugger after the next node (the Trace node) runs. You see a blue circle is displayed on the wire between the Trace node and the Compute node, which acts as a temporary breakpoint.

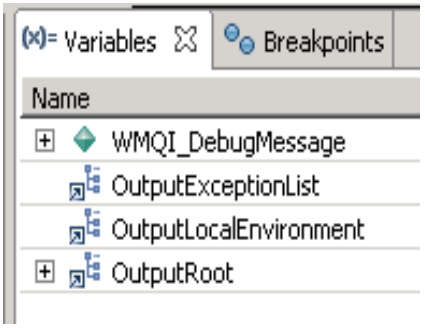


__ b. Click **Step over** again (or press F6), or click **Resume** (or press F8). The message flow resumes until the next breakpoint.

The next breakpoint is in the Transform_to_XML Compute node, where you set the breakpoint in the ESQL code.

__ c. Examine the **Variables** view.

The incoming message is shown as **WMQI_DebugMessage**, and the new (transformed) output message is shown in OutputRoot.

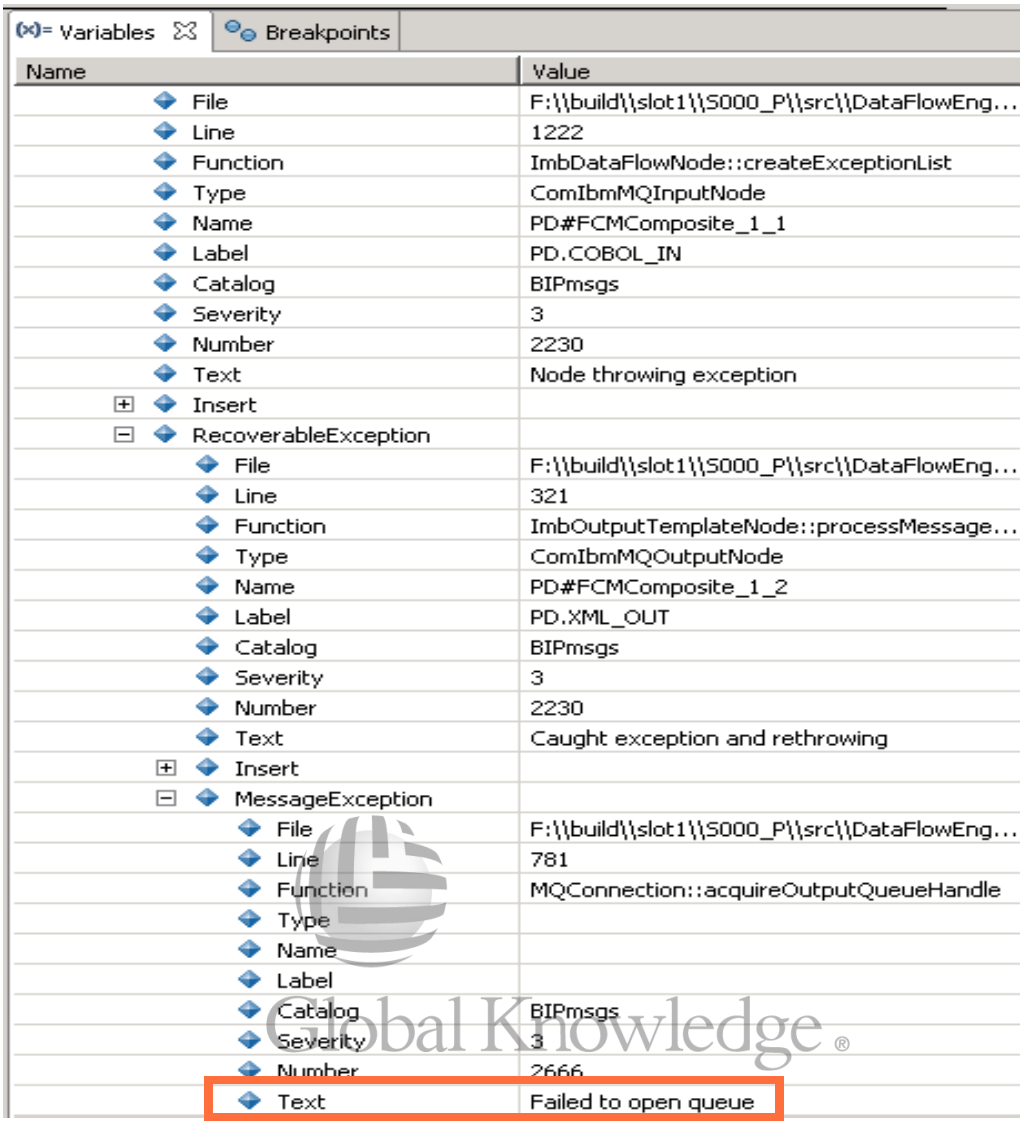


__ d. Click **Resume** again to continue processing the message.

You should see a red exclamation point that is displayed over the COBOL_IN node to indicate that an error occurred in the message flow.

__ 12. On the **Variables** tab, expand the **ExceptionList** to find the cause of the exception.

The ExceptionList contains multiple levels. It might be necessary to expand multiple levels in the ExceptionList to find the original error.

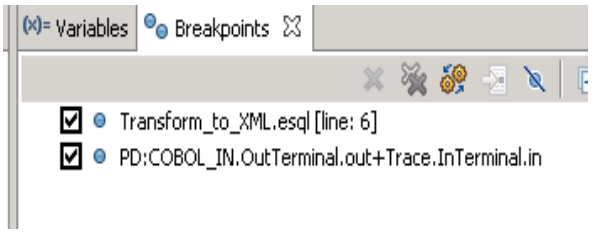


__ 13. Disconnect from the Message Flow debugger by right-clicking the first line in the **Debug** view and then clicking **Disconnect**.

You can also click **Terminate and Remove**.

__ 14. Remove all breakpoints from the message flow and ESQL.

__ a. Click the **Breakpoints** tab (in the same view area as the **Variables** view) to display the **Breakpoints** view. The list of current breakpoints is displayed.



__ b. Right-click anywhere in the **Breakpoints** view and then click **Remove all**.

__ c. Click **Yes** to confirm the removal of the breakpoints.

__ 15. In the Debug perspective, click **Integration Development** (above the **Breakpoints** and **Variables** tab) to return to the Integration Development perspective.

Part 4: Use Component Trace in the Unit Test Client

In this part of the exercise, you enable the Unit Test Client in the IBM Integration Toolkit and then use the Test Client to generate a Component Trace.

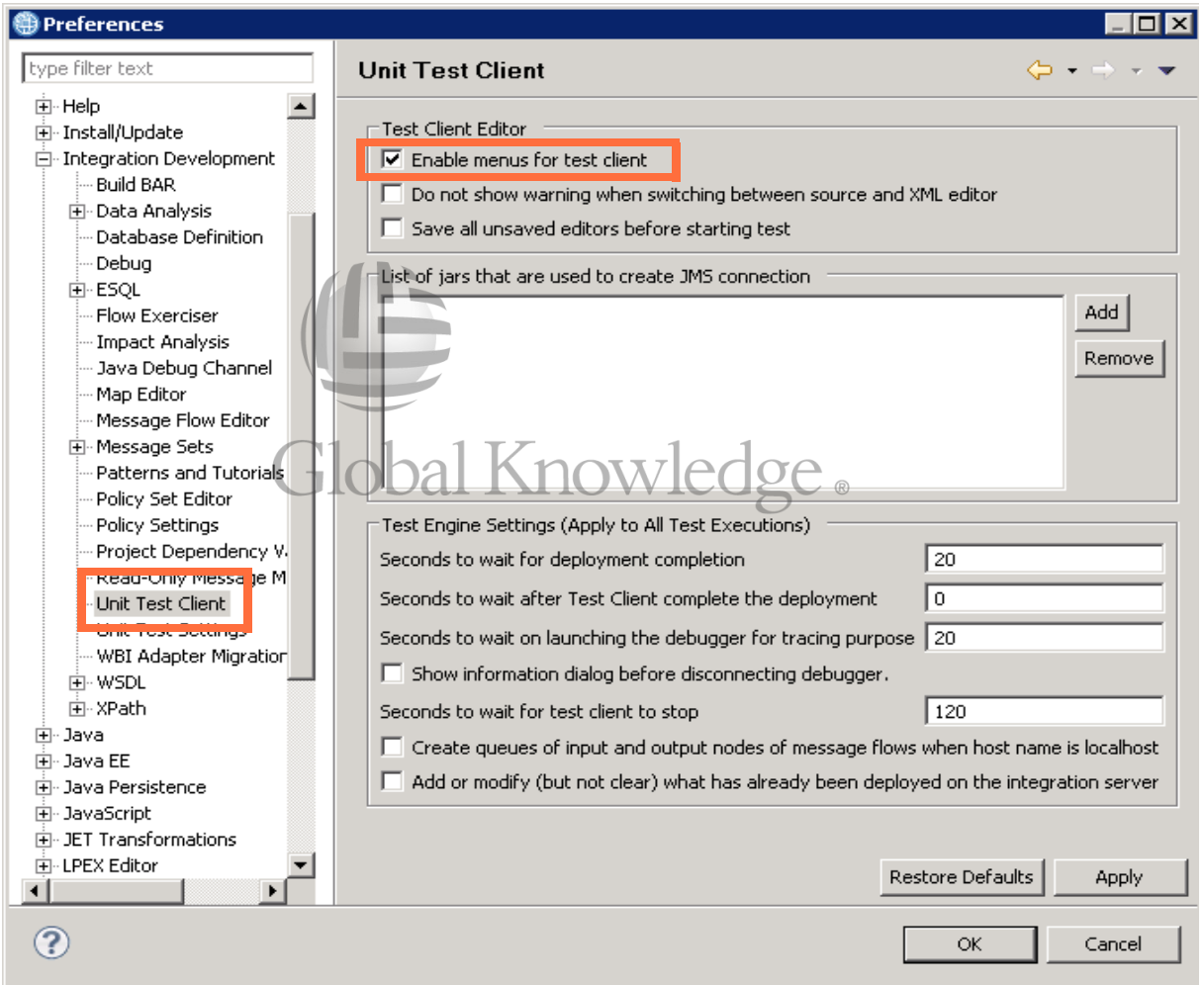
__ 1. Enable the Unit Test Client.

__ a. Click **Window > Preferences** to open the IBM Integration Toolkit Preferences window.

__ b. Expand **Integration Development**.

__ c. Click **Unit Test Client**.

__ d. Click **Enable menus for test client** and then click **OK**.



__ 2. Start the Test Client for the message flow.

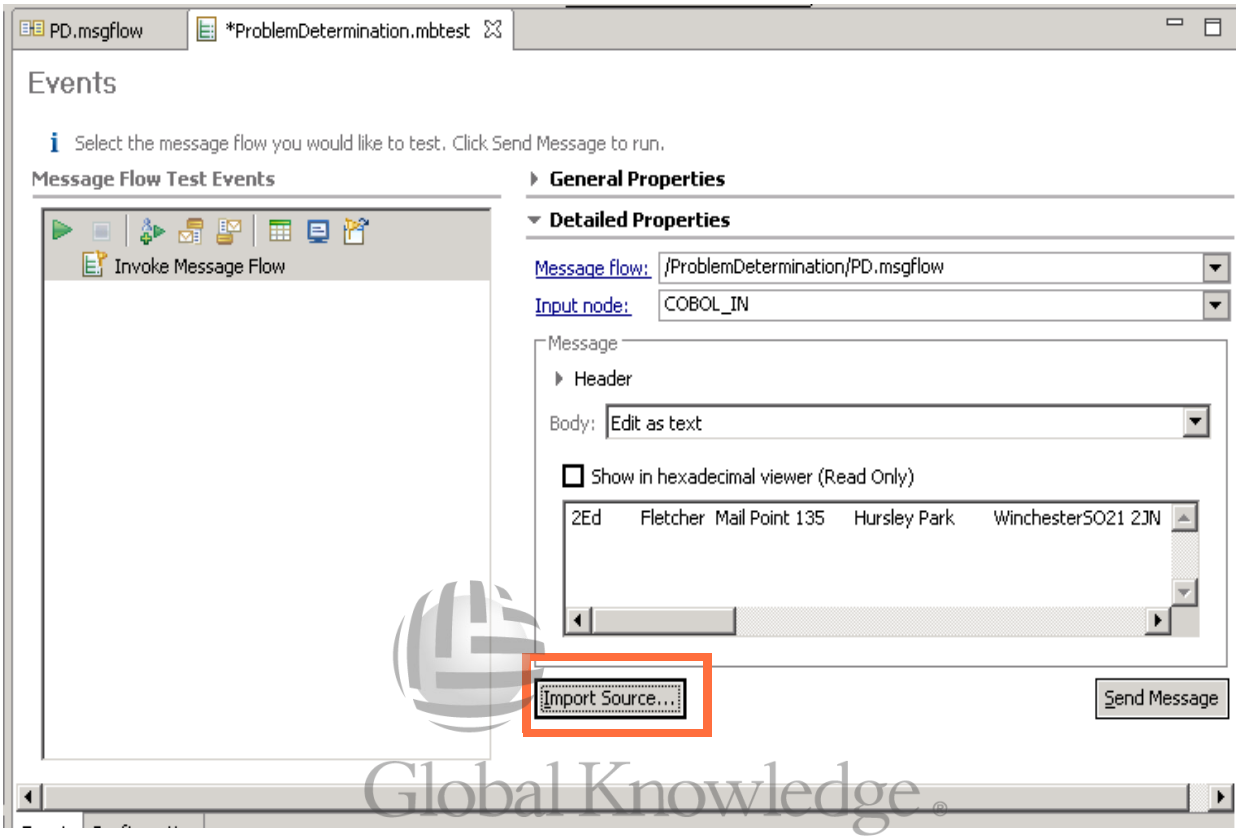
__ a. In the Message Flow editor, right-click the COBOL_IN MQInput node and then click **Test**.

__ b. Click **OK** to the confirmation message that reminds you that an application owns the message flow.

The message flow Test Client file is created and the Test Client view opens. The Test Client file is named `ProblemDetermination.mbtest` and is saved in the **TestClientBarFiles** folder under the **Independent Resources** project.

__ 3. Import the message into the Test Client.

__ a. Click **Import Source**.



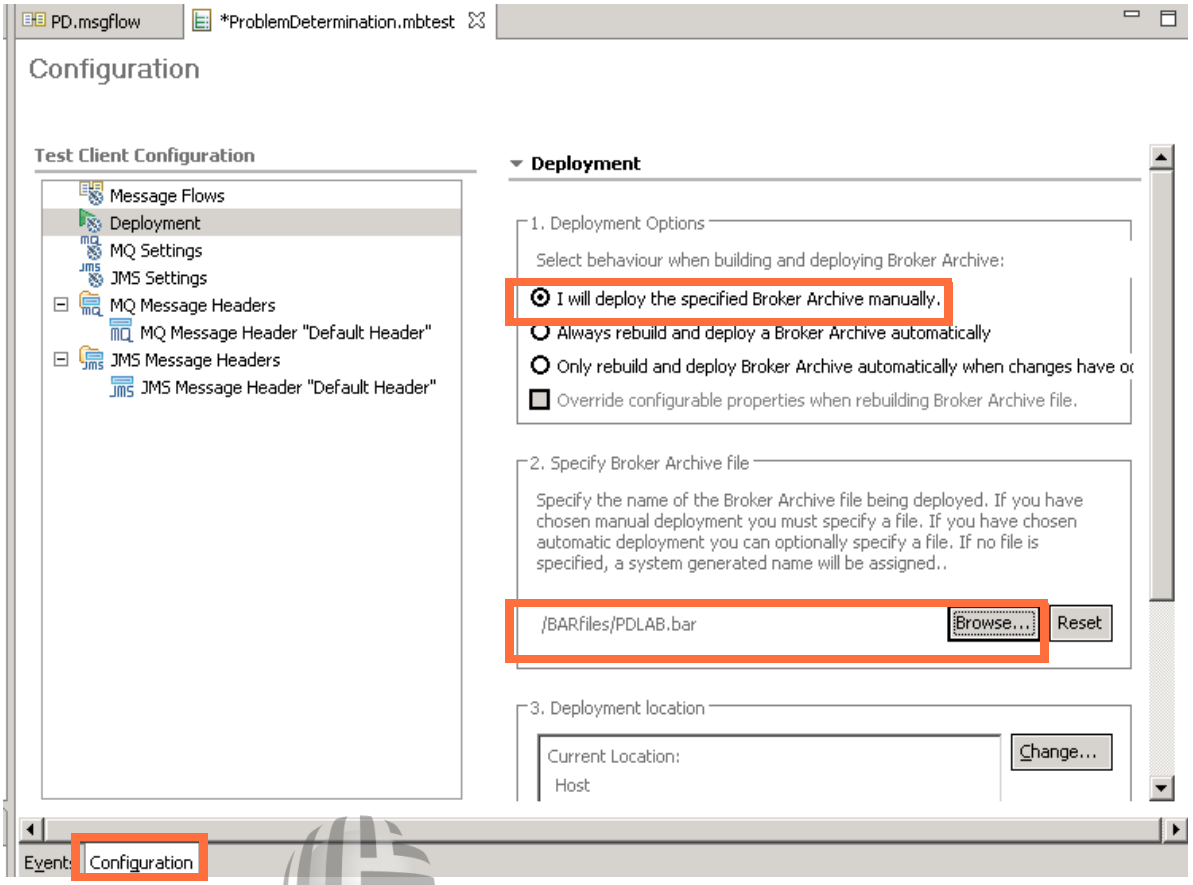
__ b. Browse to `C:\labfiles\Lab07-PDTools\data` directory, click `Complaint_cwf.txt`, and then click **Open**.

__ 4. Configure the Test Client for Component Trace to use the existing BAR file.

__ a. Click the **Configuration** tab.

__ b. In the **Deployment Options** section, click **I will deploy the specified Broker Archive manually**.

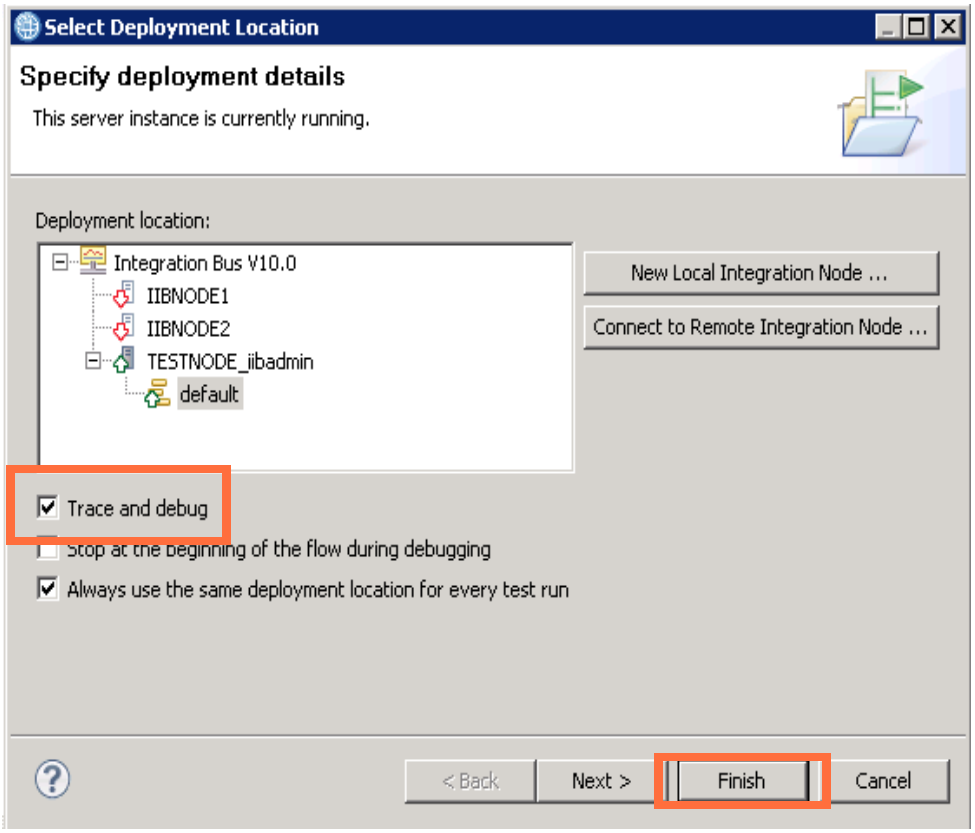
__ c. In the **Specify Broker Archive file** section, click **Browse**, click the **PDLAB.bar** file, and then click OK.



__ 5. Enable the component trace.

__ a. In the **Deployment Location** section on the **Configuration** tab, click **Change**.

__ b. Select **Trace and debug**, and then click **Finish**.



__ 6. Send a test message with the Test Client.

__ a. On the **Events** tab of the Test Client, click **Send Message**.

The debugger starts automatically. However, because you removed all of the breakpoints, the flow is not suspended.

__ b. If you are prompted to switch to the debug perspective, click **No**.

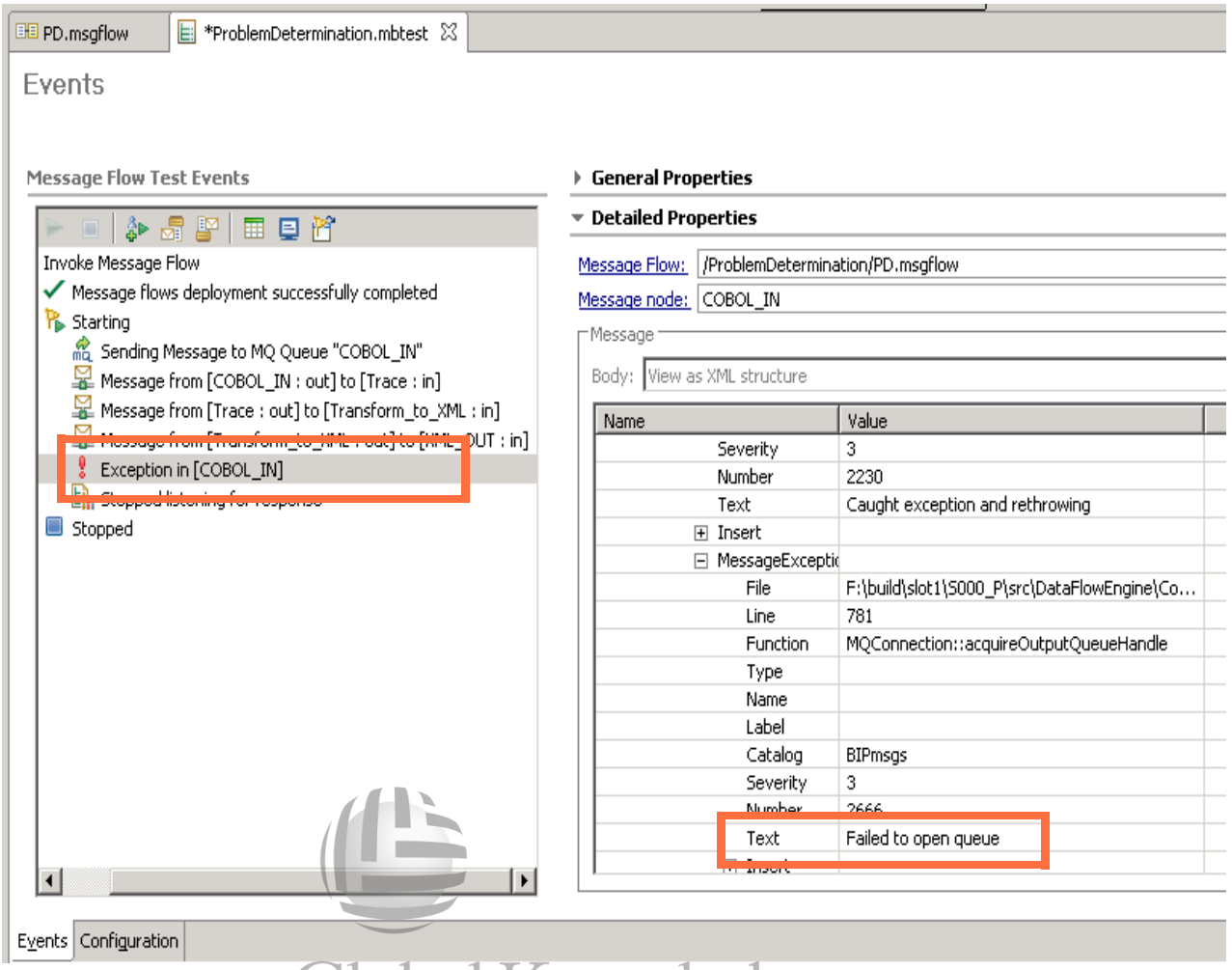
In most cases, you would choose **Yes**, but because no breakpoints are set, switching to the debug perspective is not necessary.

__ 7. Review the Component Trace output.

__ a. Review the Message Flow Test Events in the Test Client **Events** tab. Observe the exception that is shown in the **Events** list.

__ b. Click the exception, and review the **Detailed Properties** section.

__ c. Expand the **WMQI_ExceptionList**. You see several nested exceptions. Recall that it is generally the innermost exception that describes the error that occurred.



__ d. Review the innermost exception and the **Insert** messages. One of the **Insert** messages shows the 2085 status code, which you reviewed earlier in this exercise.

Clean up the environment

__ 1. Close all open editors.

__ 2. Stop the Message Flow debugger. In the **Integration Nodes** view, right-click the **default** integration server and click **Terminate Debugger**.

__ 3. Remove all active flows from the integration server. In the Integration Nodes view, right-click the default integration server and then click **Delete > All Flows and Resources**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you extended the message flow by adding a Trace node and then ran the flow so that you could examine the trace results. You also used the IBM Integration Bus commands to turn off the Trace node and to verify the change.

In the second part of this exercise, you changed the message flow so that it failed at run time. Then, you activated and examined a user trace and the system logs (Event Viewer) to identify the problem. You also used the IBM MQ SupportPac RFHUtil to put and get messages from the queues.

In the third part of this exercise, you set up the Message Flow Debugger and set breakpoints in the message flow. You then used the Debug perspective in the IBM Integration Toolkit to step through the message flow and examine the ExceptionList to identify the problem with the message flow.

In the fourth part of this exercise, you used the IBM Integration Toolkit Unit Test Client to identify the cause of a message flow failure and examine the ExceptionList.

Having completed this exercise, you should be able to:

- Enable a user trace and retrieve the collected trace data
- Add a Trace node to a message flow application
- Use RFHUtil to send test data to a message flow and view messages on an IBM MQ queue
- Use the IBM Integration Toolkit Test Client and message flow debugger view to step through a message flow application
- Examine the IBM Integration Bus logs and system logs to diagnose problems

Exercise 8. Implementing explicit error handling

What this exercise is about

In this exercise, you implement message processing nodes that control the paths that messages take in a message flow. You also write a general-purpose subflow to handle errors that occur during message processing.

What you should be able to do

After completing this exercise, you should be able to:

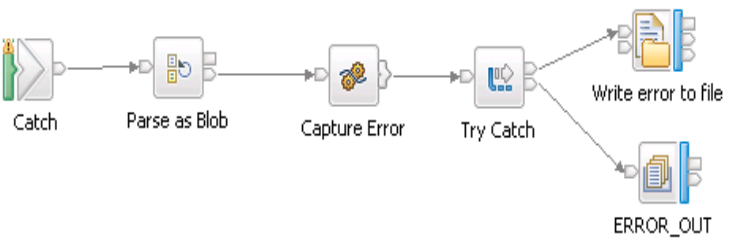
- Implement a generic error handling routine in the form of a subflow
- Use a ResetContentDescriptor node to force the message to be reparsed according to the parser domain that is specified in the node properties
- Use the TryCatch node to provide a special handler for exception processing

Introduction

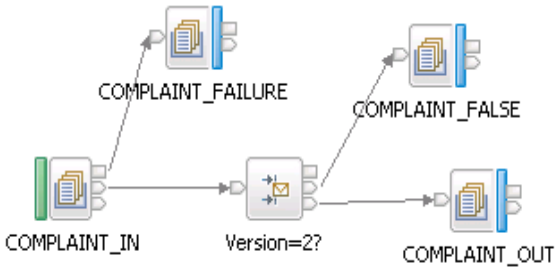
In a typical IBM Integration Bus environment, it is useful to have a standard way to handle runtime errors that occur. Runtime errors can be handled locally (at the node where they occur) for many types of message processing nodes by wiring the **Failure** terminal. However, in many instances an organization might choose to implement “standard” code that is used for displaying and reporting runtime errors.

In this exercise, you add a general-purpose error handler to a message flow application. The error handler is written in the form of a subflow.

The subflow gets the exception from the ExceptionList. It uses a TryCatch node to report the exception information with the input message. The TryCatch node tries to write a message to an IBM MQ queue that is named ERROR_OUT. If the queue is not available, the TryCatch node sends the information to the **Catch** terminal is connected to a FileOutput node, which writes error information to a file.



The main message flow routes messages as determined by the VERSION number in the message.



The message flow reads the IBM MQ message from a queue that is named COMPLAINT_IN. If the VERSION element is set to 2, the message is routed to the COMPLAINT_OUT queue, otherwise the message is routed to the COMPLAINT_FALSE queue.

The XML schema `Complaint.xsd` defines the message. A sample input message is shown here.

```
<CUSTOMERCOMPLAINT>
   <VERSION>2</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <COUNTRY>UK</COUNTRY>
   </CUSTOMER_ADDRESS>
   <COMPLAINT>
      <C_TYPE>Delivery</C_TYPE>
      <C_REF>XYZ123ABC</C_REF>
      <C_TEXT>My order was delivered in time, but the package was
torn.</C_TEXT>
   </COMPLAINT>
</CUSTOMERCOMPLAINT>
```


In the first part of this exercise, you create the error handler subflow.

In the second part of this exercise, you add the error handler subflow to the main flow.

In the third part of this exercise, you use the IBM Integration Toolkit Flow exerciser to test the message flow and the error handler subflow.

Requirements

- A lab environment with IBM Integration Bus V10 and IBM MQ V8
- Lab files in the `C:\labfiles\Lab08-ErrorHandler`
- The following local queues are on the IIBQM queue manager: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_FALSE, COMPLAINT_OUT, and ERROR_OUT
- The user “iibadmin” is a member of the “mqm” group

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab08`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab08`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, verify that the integration node **TESTNODE_iibadmin** is started.

Start it if it is stopped.

__ 4. This exercise requires an IBM MQ queue manager that is named **`IIBQM`**.

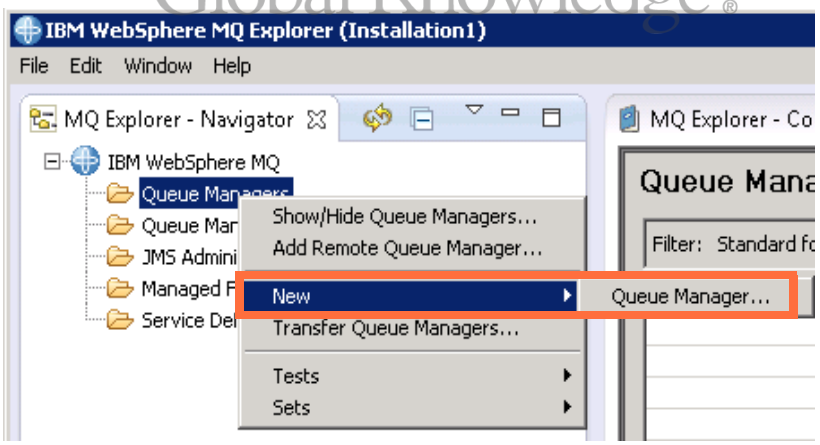
If you created this queue manager in a previous exercise, proceed to Step 5.

If you do not have a queue manager, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ` by following these instructions.

__ a. Start IBM MQ Explorer if it is not already running.

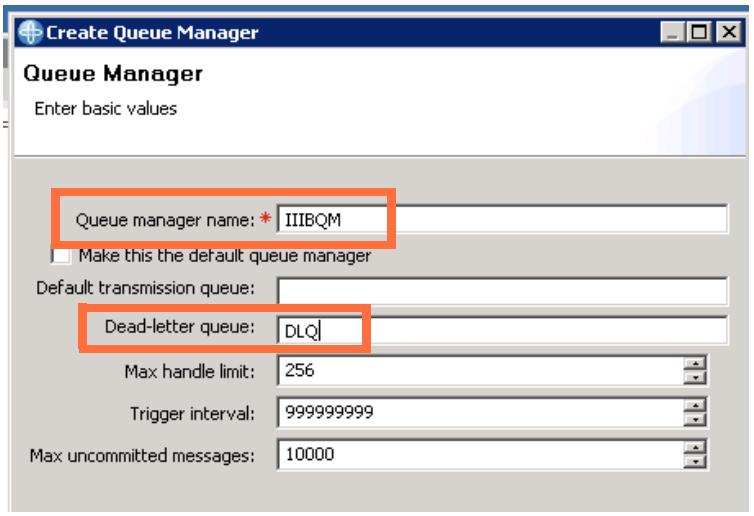
From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

__ b. In the **MQ Explorer - Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ c. For the **Queue Manager** name, type: **`IIBQM`**

__ d. For the **Dead-letter queue**, type: **DLQ**



__ e. Click **Finish**.

After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 5. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

In the command window, type:

```
runmqsc IIBQM < C:\labfiles\Lab08-ErrorHandler\CreateQueues.mqsc
```

__ 6. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the **MQ Explorer - Navigator** view, expand the **Queue Managers > IIBQM** folder.

__ b. Click **Queues** under the queue manager in the **MQ Explorer - Navigator** view to display the **Queues** view.

__ c. Verify that the following queues are listed in the **Queues** view: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_FALSE, COMPLAINT_OUT, DLQ, and ERROR_OUT

Note

You might have other queues for this queue manager from a previous exercise. You do not need to delete the unused queues.

__ 7. Import the project interchange file that is named `RouteComplaint_PI.zip` file from the `C:\labfiles\Lab08-ErrorHandler` directory into your workspace.

__ a. From the IBM Integration Toolkit, click **File > Import**.

__ b. Click **IBM Integration > Project Interchange** and then click **Next**.

__ c. To the right of **From zip file**, click **Browse**.

__ d. Browse to the `C:\labfiles\Lab08-ErrorHandler` directory.

__ e. Select `RouteComplaint_PI.zip`, and then click **Open**. The **Import Project Interchange Contents** window is displayed.

__ 8. Ensure that the **RouteComplaint** application is selected and then click **Finish**.

The RouteComplaint application contains the message flow, **`MyFlow.msgflow`**, and the XML schema, **`Complaint.xsd`** that defines the input file.

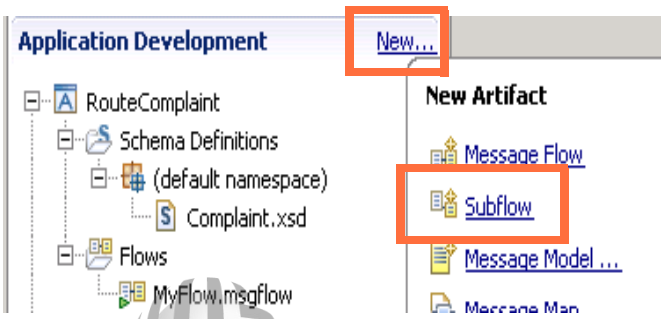
Part 1: Create the ErrorHandler subflow

In this part of the exercise, you use the IBM Integration Toolkit to construct an error handler subflow.

__ 1. Create the message flow container for the subflow.

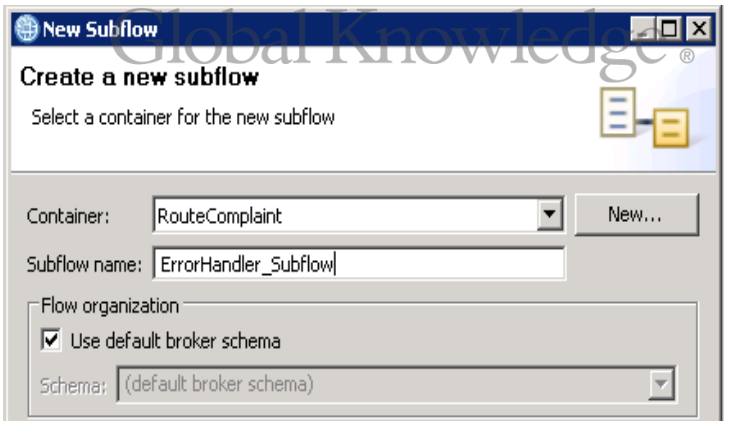
__ a. In the **Application Development** view, click **New**.

__ b. Click **Subflow** in the **New Artifact** menu.



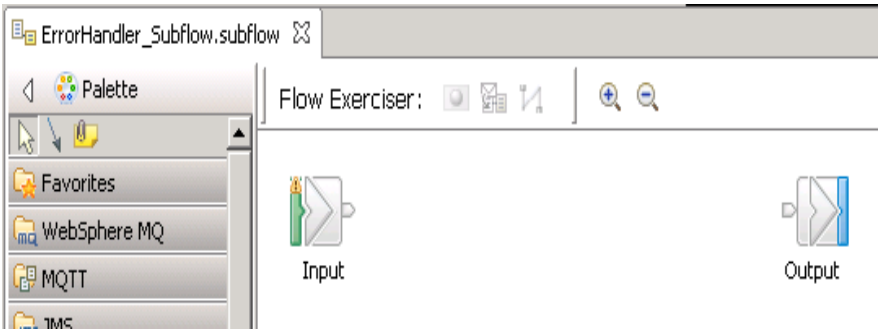
__ 2. For **Container**, select **RouteComplaint**.

__ 3. For **Subflow name**, type: **`ErrorHandler_Subflow`**



__ 4. Click **Finish**.

The message flow editor opens. The subflow Input node and Output node are already shown on the canvas.



In the next several steps, you add and configure the message processing nodes for the subflow.

__ 5. Rename the Input node to **Catch**.

On the **Description** tab of the node **Properties**, change the **Node name** property to **`Catch`**.

__ 6. This subflow is not intended to send data back to the main flow.

Delete the Output node from the subflow Message Flow editor canvas.

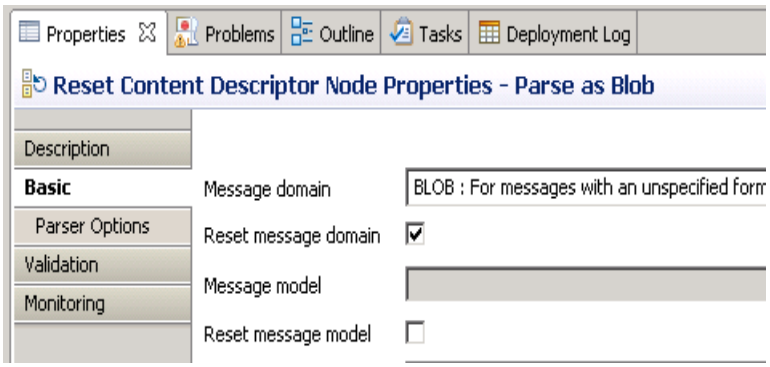
__ 7. Add a ResetContentDescriptor node to the subflow after the subflow Input node.

This node forces the message to be reparsed according to the parser domain you specify in the node properties. Regardless of the domain of the incoming message, it can be converted to a BLOB by using this technique.

__ a. Add a ResetContentDescriptor node to the canvas from the **Construction** drawer of the Message Flow editor Palette.

__ b. Rename the ResetContentDescriptor node to: **`Parse as BLOB`**

__ c. On the **Basic** tab, set the **Message domain** property to BLOB and select the **Reset message domain** check box.



__ 8. Add a Compute node to the subflow.

The Compute node in the subflow uses the ExceptionList tree, the Environment tree, integration node variables, and shared variables to construct a message that contains the exception information and the original input message (packaged as a BLOB). Some of the code is taken directly from the IBM Knowledge Center for IBM Integration Bus.

The ESQL code in the Compute node is called if a runtime error occurs only. The code retrieves the information about the error from the ExceptionList, formats it, and places it in the `OutputRoot.MQRFH2.usr` folder.

__ a. From the **Transformation** drawer, add a Compute node to the canvas after the ResetContentDescriptor node (**Parse as Blob)**.

__ b. Rename the Compute node to: **`Capture Error`**

__ c. On the **Basic** properties tab, change the **ESQL Module** property to:

`{default}:Capture_ExceptionList`

__ d. In the Message Flow editor, double-click the Compute node to open the ESQL editor.

__ e. The code for Compute node is provided for you.

Open the file `Cut&Paste.txt` file in the `C:\labfiles\Lab08-ErrorHandler` directory with Notepad.

Copy the entire contents of the file and replace the entire existing contents of the ESQL module.

The code is provided here for your reference.

Note

You learn more about ESQL in a later unit in this course.

```
DECLARE s_Counter SHARED INT 0;

CREATE COMPUTE MODULE Capture_ExceptionList

CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN

     DECLARE ID CHAR;
     DECLARE mNumber integer;
     DECLARE mText char;

     CALL CopyMessageHeaders();

     SET OutputRoot.Properties.Transactional = false;
      -- Report the integration node, integration server and message
      -- flow with the exception information
     SET ID = BrokerName || '/' ||
              ExecutionGroupLabel || '/' ||
              MessageFlowLabel;

     SET Environment.Variables.ID = ID;

     -- Temporarily store ExceptionList in OutputRoot.XMLNSC
     -- to profit from implicit casting of XML parser into CHAR
     -- which is required for RFH2.usr

     -- Pass the ExceptionList to return back the inner most child
     -- error number and Error Reason
     CALL getLastExceptionDetail(InputExceptionList, mNumber, mText);

     -- Track the number of messages that error and add this counter
     -- to fields tracked in RFH2 USR folder
     BEGIN ATOMIC
       SET s_Counter = s_Counter + 1;
       set OutputRoot.XMLNSC.ExceptionDump.ErrCounter = s_Counter;
       SET OutputRoot.XMLNSC.ExceptionDump.ErrList = InputExceptionList;
       set Environment.Variables.ErrCounter = s_Counter;
       Set Environment.Variables.errorNum = mNumber;
       Set Environment.Variables.errorReason = mText;
     END;
```

```
      SET OutputRoot.MQRFH2.usr=OutputRoot.XMLNSC;
      SET OutputRoot.MQRFH2.usr.ErrorHandler.ID = ID;
      SET OutputRoot.MQRFH2.usr.Counter = s_Counter;
      SET OutputRoot.MQRFH2.usr.ErrorNumber = mNumber;
      SET OutputRoot.MQRFH2.usr.ErrorReason = mText;

      -- Delete the temporary XML body
      DELETE FIELD OutputRoot.XMLNSC;

      -- copy original message body
      SET OutputRoot.BLOB = InputBody;

      RETURN TRUE;

   END;  /* main */

  CREATE PROCEDURE getLastExceptionDetail(IN  InputTree reference,
                                    OUT messageNumber integer,
                                         OUT messageText char)

/*****************************************************************************
    * A procedure that gets the details of the last exception from a message
    * IN InputTree:  The incoming exception list
    * IN messageNumber:  The last message number.
    * IN messageText: The last message text.

*****************************************************************************
*/
BEGIN

-- Create a reference to the first child of the exception list
   declare ptrException reference to InputTree.*[1];

   -- keep looping while the moves to the child of exception list work
   WHILE lastmove(ptrException) DO

      -- store the current values for the error number and text
      IF ptrException.Number is not null THEN
            SET messageNumber = ptrException.Number;
            SET messageText = ptrException.Text;
      END IF;

      -- now move to the last child which should be the next exception list
      move ptrException lastchild;

   END WHILE;
END; /* getLastException */
```



```
CREATE PROCEDURE CopyMessageHeaders()
BEGIN
   DECLARE I INTEGER;
   DECLARE J INTEGER;
   SET I = 1;
   SET J = CARDINALITY(InputRoot.*[]);
   WHILE I < J DO
      SET OutputRoot.*[I] = InputRoot.*[I];
      SET I = I + 1;
   END WHILE;
END;

END MODULE;
```

__ f. Save the ESQL file by pressing Ctrl+S.

__ g. Close the ESQL editor

Information

The ESQL code for exception handling in this exercise is based partially on information that is available in the IBM Knowledge Center for IBM Integration Bus in the “Accessing the ExceptionList using ESQL” topic.

__ 9. Add a TryCatch node.

From the **Construction** drawer, add a TryCatch node to the canvas after the Capture **Error Compute** node.

You do not need to rename this node.

__ 10. Add an MQOutput node for the “Try” destination for the original message and the error information.

__ a. From the **WebSphere MQ** drawer, add an MQOutput node message flow after the Try Catch node.

__ b. Rename the MQOutput node to **ERROR_OUT**.

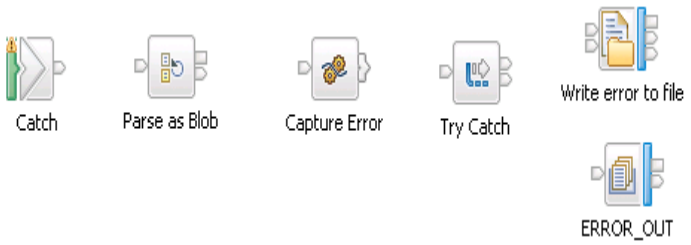
__ c. For the **Queue name** property on the **Basic** tab, type: **ERROR_OUT**

__ d. For the **Destination queue manager name** property on the **MQ Connections** tab, type: **IIBQM**

__ 11. Add a FileOutput node the “Catch” destination for the original message and error information.

__ a. From the **File** drawer, add a FileOutput node to the canvas above the ERROR_OUT MQOutput node.

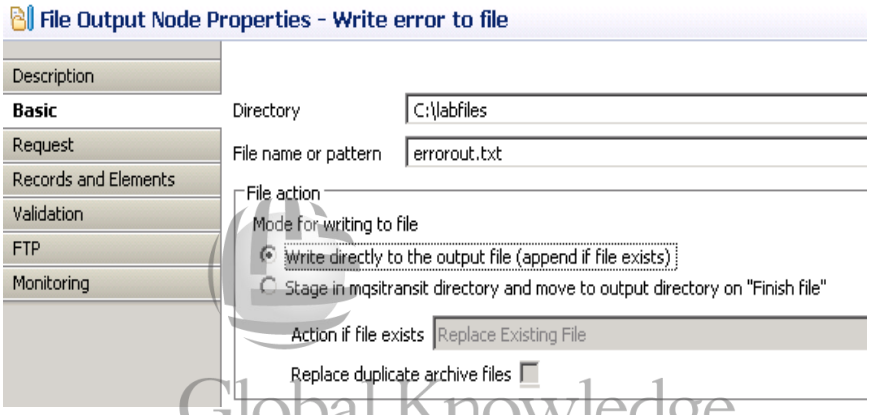
__ b. Rename the FileOutput node to **`Write error to file`**.



__ c. For the **Directory** property on the **Basic** tab, type: **`C:\labfiles`**

__ d. For **File name or pattern** on the **Basic** tab, type: **`errorout.txt`**

__ e. For File action property on the Basic tab, select **Write directly to the output file (append if file exists)**.



__ f. For the **Data location** property on the **Request**, type: **$Root**

__ 12. Connect the nodes.

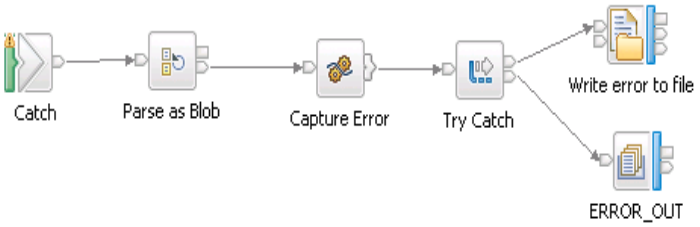
__ a. Wire the Input node (**Catch**) **Out** terminal to the ResetContentDescriptor node (**Parse as Blob**) **In** terminal.

__ b. Wire the ResetContentDescriptor node (**Parse as Blob**) **Out** terminal to the Compute node (**Capture Error**) **In** terminal.

__ c. Wire the Compute node (**Capture Error**) **Out** terminal to the TryCatch node **In** terminal.

__ d. Wire the TryCatch node **Try** terminal to the MQOutput node (**ERROR_OUT**) **In** terminal.

__ e. Wire the TryCatch node **Catch** terminal to the FileOutput node (**Write error to file**) **In** terminal.



__ 13. Save the subflow

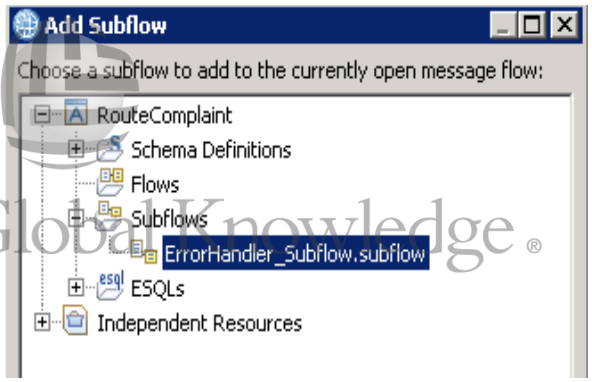
You now see the subflow file in the **Application Development** view, with the main flow that you started with at the beginning of the exercise.

Part 2: Add the subflow reference to the main flow

__ 1. If the main flow **Myflow.msgflow** (under **RouteComplaint > Flows**) is not open in the Message Flow editor, double-click it in the **Application Development** view to open it.

__ 2. Right-click in the Message Flow editor canvas, and click **Add Subflow**. The **Add Subflow** window is displayed.

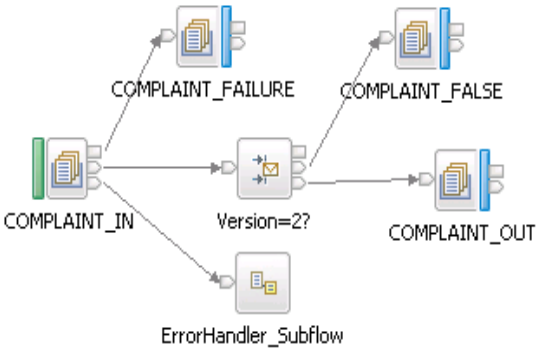
__ 3. In the Add Subflow window, expand **RouteComplaint > Subflows**, click **ErrorHandler_Subflow.subflow**, and then click **OK**.



The Subflow node **ErrorHandler_Subflow** is added to the Message Flow editor canvas.

__ 4. Position the Subflow node below and to the right of the MQInput node (COMPLAINT_IN).

__ 5. Wire the MQInput node (COMPLAINT_IN) **Catch** terminal to the **Input** terminal of the ErrorHandler_Subflow node.



__ 6. Save the message flow.

Part 3: Test the application

In this part of the exercise, you use the IBM Integration Toolkit Flow exerciser to test the message flow application. You also use the IBM MQ Explorer and RFHUtil SupportPac to view the exception message.

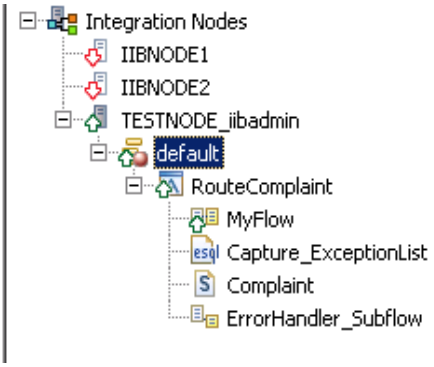
Test 1: Testing normal message flow processing

In the test, you verify that the basic message flow successfully processes a message. In this test, the input message should be put to the COMPLAINT_OUT queue.

__ 1. In the IBM Integration Toolkit Message Flow editor, start the Flow exerciser in the main message flow (**MyFlow.msgflow**).

The Flow exerciser creates the BAR file, deploys the application to the **default** integration server on TESTNODE_iibadmin, and starts recording.

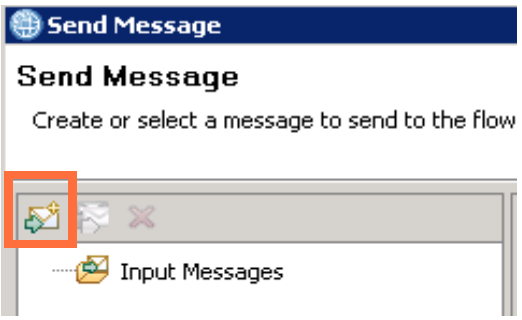
__ 2. Using the **Deployment Log** view and the **Integration Nodes** view, verify that the message flow application deployed successfully.



__ 3. Test the message flow with the `Complaint.xml` file in the `C:\labfiles\Lab08-ErrorHandler\data` directory by importing the file from the file system and sending the message with the Flow exerciser.

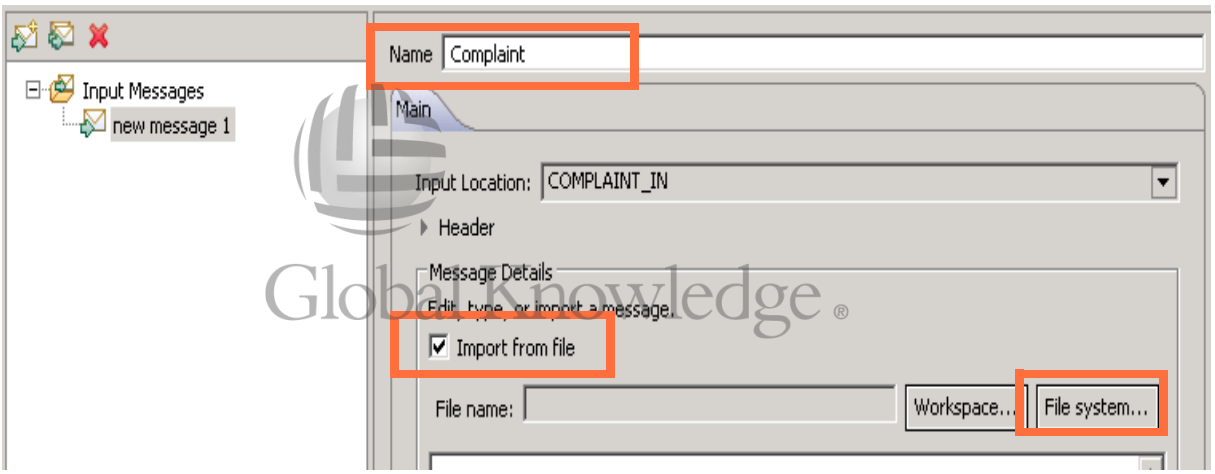
__ a. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

__ b. In the **Send Message** window, click the **New Message** icon.



__ c. For the **Name**, type: **`Complaint`**

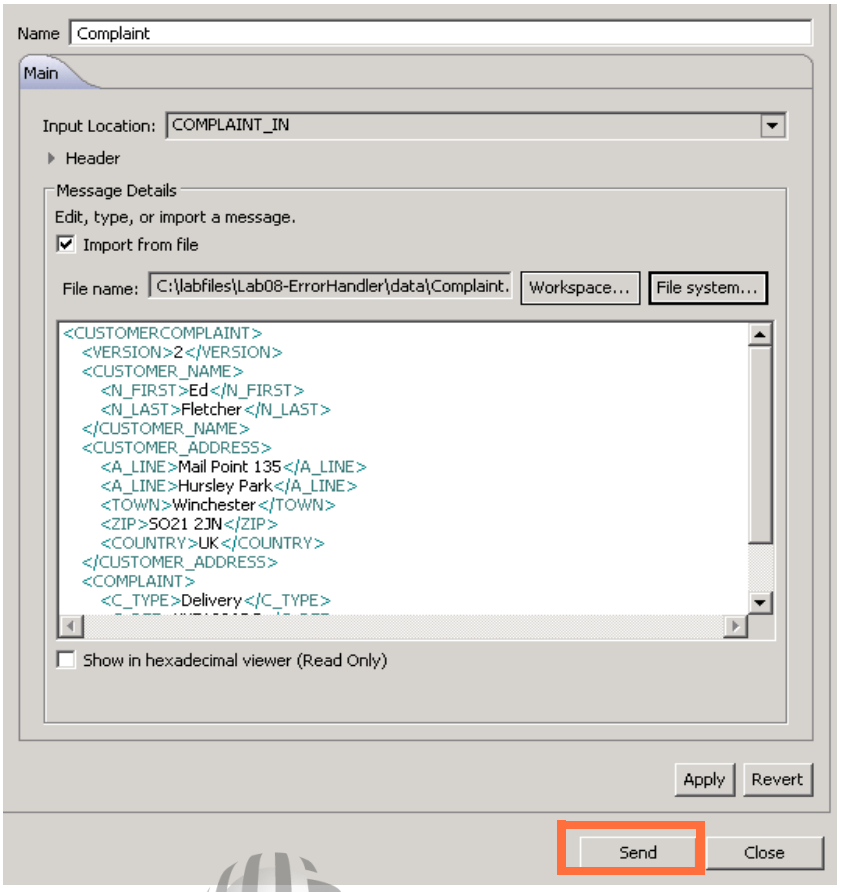
__ d. Select the **Import from file** check box and then click **File system**.



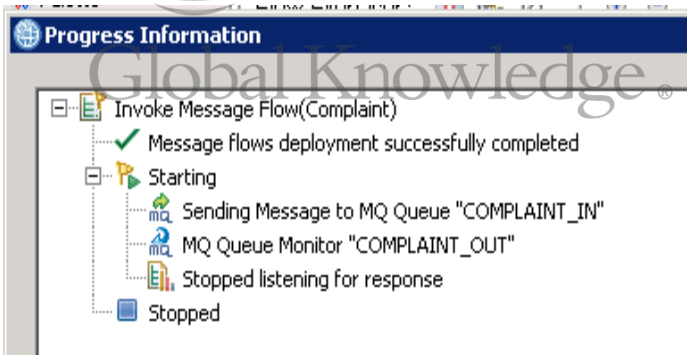
__ e. Go to the `C:\labfiles\Lab08-ErrorHandler\data` directory, click the `Complaint.xml` file, and then click **Open**.

The file is imported into the Flow exerciser. In this file, the VERSION element is set to **2**, so the message should go to the COMPLAINT_OUT queue.

__ f. Click **Send**.

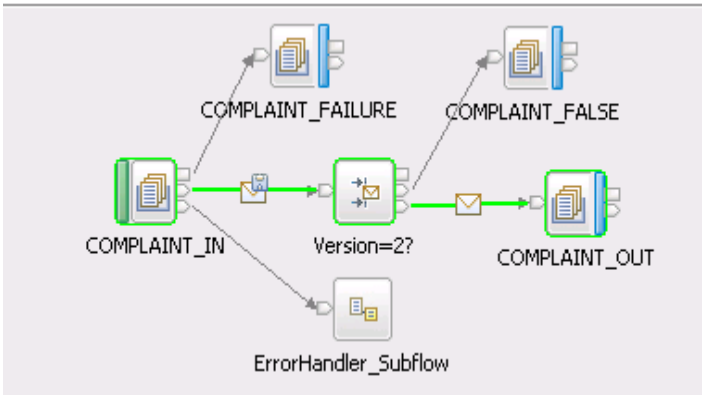


__ g. The **Progress Information** window shows that the message is sent to input queue COMPLAIN_IN. It also shows the destination, which is the COMPLAINT_OUT queue.



When the **Progress Information** window displays **Stopped**, the test is complete. Click **Close**.

__ h. The message path is highlighted on the message flow.



As expected, the message is routed to the COMPLAINT_OUT queue because the VERSION element is set to **2**.

Information

If you view the queues by using IBM MQ Explorer, you see that the **Current queue depth** for the output queue is empty. The Flow exerciser does not put the message to the output queue. It shows the message path only. If you want to view the messages on the queues, you can use the IBM MQ `amqsput` sample program or the RFHtil SupportPac to test the flow by putting a message to the COMPLAINT_IN queue. You can use the Flow exerciser to view the message path.

For example, to use the `amqsput` sample program to verify that the `Complaint.xml` message is put on the COMPLAINT_IN queue on the IIBQM queue manager, type the following command in a command window:

```
amqsput COMPLAINT_IN IIBQM < C:\labfiles\Lab08-ErrorHandler\data\Complaint.xml
```


Test 2: Calling the error handler

In this test, you modify the test environment to create a runtime error that causes the message to go to the ErrorHandler_Subflow. In this test, the message should arrive on the ERROR_OUT queue with the exception information and the original message.

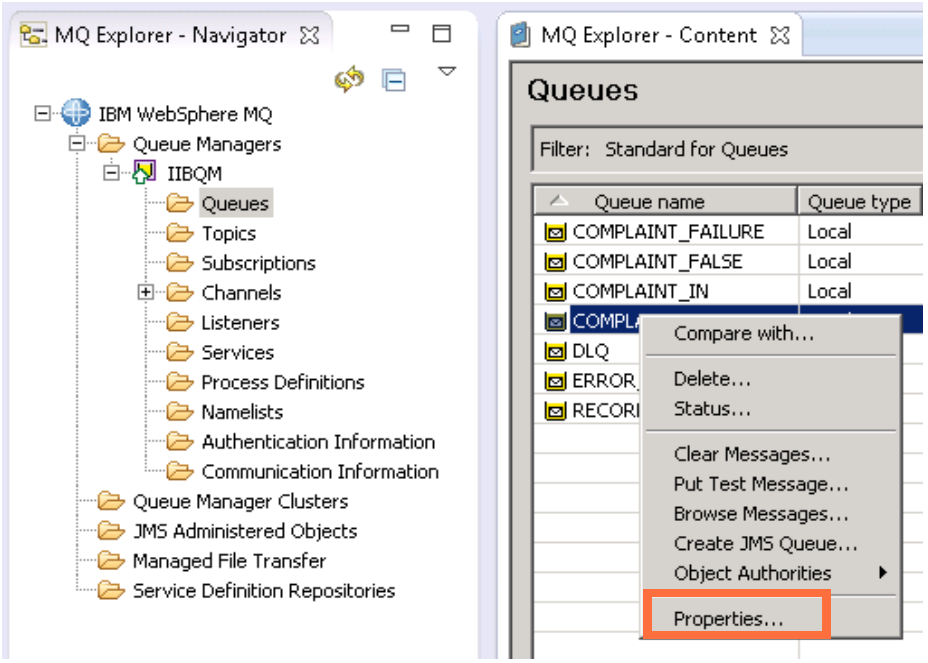
__ 1. To cause a runtime error, you set the COMPLAINT_OUT queue to the “Put inhibited” status so that the message cannot be written to the output queue.

__ a. Open IBM MQ Explorer if it is not already open.

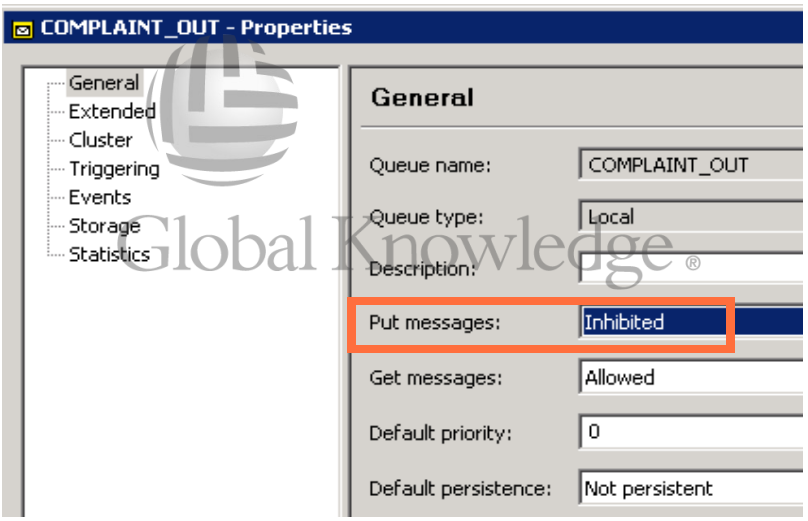
__ b. In the **MQ Explorer - Navigator** view, expand **Queue Managers > IIBQM**.

__ c. Click **Queues** to display the **Queues** content view.

__ d. Right-click COMPLAINT_OUT and then click **Properties**. The queue **Properties** window is displayed.

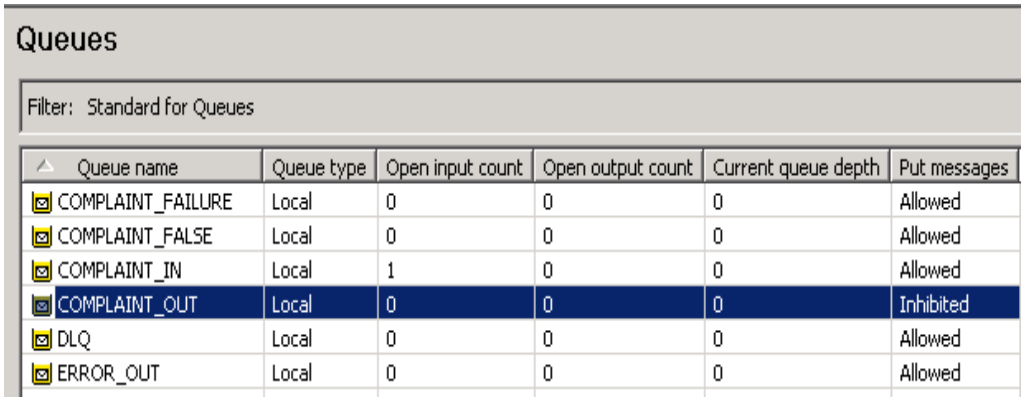


e. On the **General** tab change the setting for **Put messages** to **Inhibited** and then click **Apply**.



__ f. Click **OK** to close the queue properties window.

__ g. In the list of queues, for COMPLAINT_OUT, verify that **Put messages** option is now set to Inhibited.

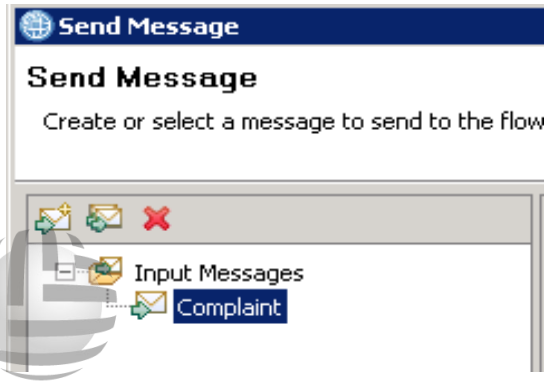


Queue name	Queue type	Open input count	Open output count	Current queue depth	Put messages
COMPLAINT_FAILURE	Local	0	0	0	Allowed
COMPLAINT_FALSE	Local	0	0	0	Allowed
COMPLAINT_IN	Local	1	0	0	Allowed
COMPLAINT_OUT	Local	0	0	0	Inhibited
DLQ	Local	0	0	0	Allowed
ERROR_OUT	Local	0	0	0	Allowed

__ 2. Send the test message.

__ a. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

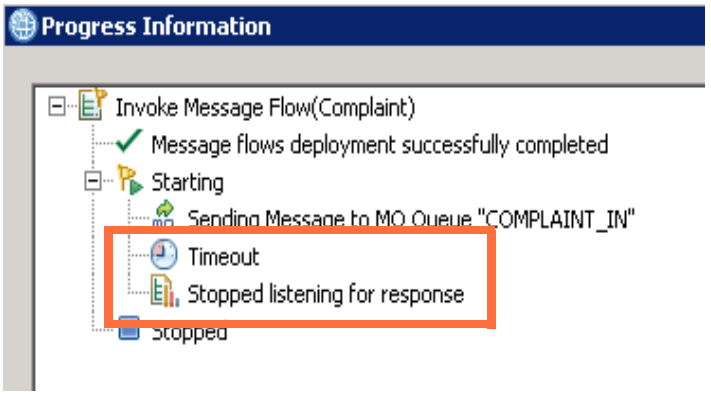
__ b. On the **Send Message** window, click **Complaint** under the **Input Messages** folder.



__ c. Click **Send**.

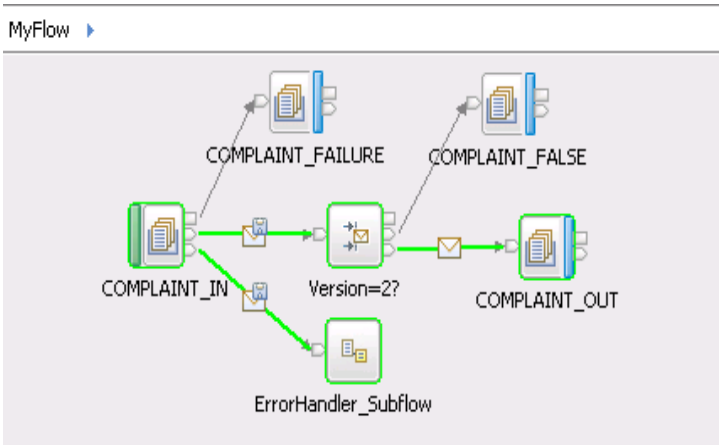
__ 3. This time the **Progress Information** shows the status.

Wait for the Timeout event and the notification that the Flow exerciser stopped listening for a response. Click **Close** to view the path.



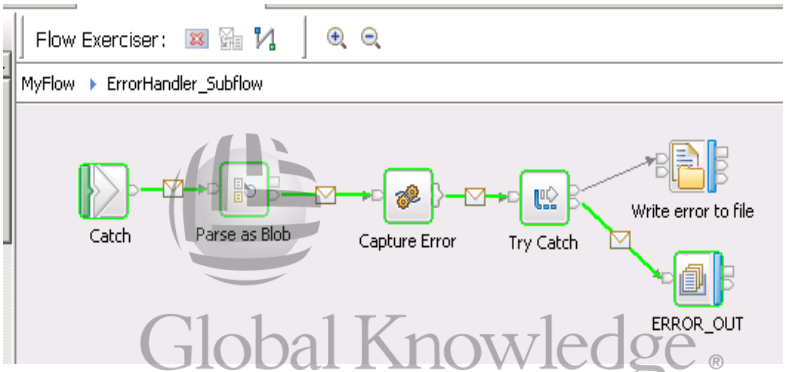
__ 4. View the flow path.

Based on the path on main flow you can see that an attempt is made to send the message to the COMPLAINT_OUT queue but now the path to ErrorHandler_subflow is also highlighted.



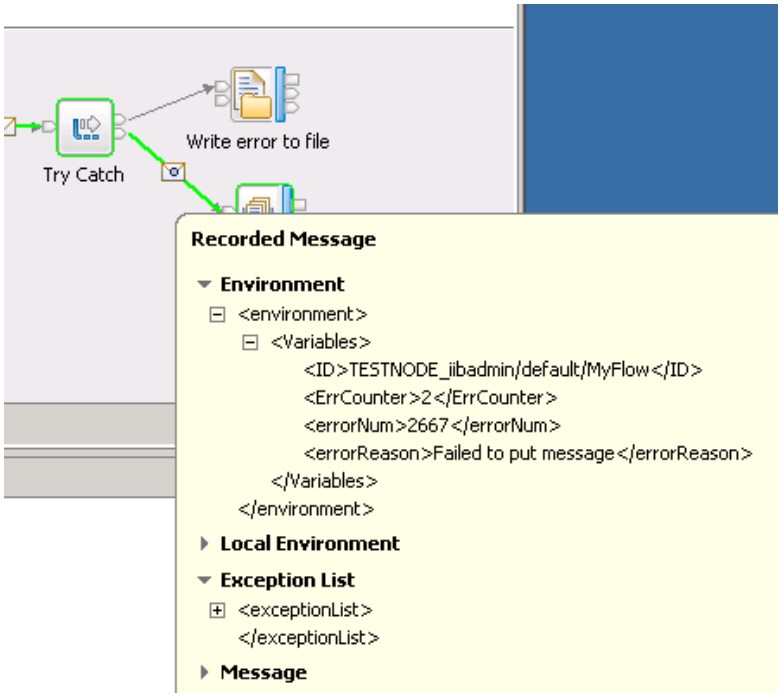
The message rolls back to the COMPLAINT_IN node because the **Failure** terminal on the COMPLAINT_OUT node is not wired. The message is then sent out the COMPLAINT_IN node **Catch** terminal because it is wired.

__ 5. Double-click the **ErrorHandler_Subflow** node in the main flow to display the subflow path.



In the ErrorHandler_Subflow view, you see that message is sent down the **Try** path of the TryCatch node to the ERROR_OUT queue.

__ 6. Click the message icon on the path between the Try Catch node and the ERROR_OUT node.



The Environment tree contains the error information that the **Capture Error** Compute node creates. These lines of ESQL code in the Compute node create the <ID>TESTNODE_iibadmin/default/MyFlow<ID> element:

```
SET ID = BrokerName || '/' ||
    ExecutionGroupLabel || '/' ||
    MessageFlowLabel;
SET Environment.Variables.ID = ID;
```


The contents of the <ID> element identify the integration node, integration server, and message flow that generated the exception.

These next lines of ESQL code get the exception information from the ExceptionList and create the <ErrCounter>, <errorNum>, and <errorReason> elements in the Environment tree:

```
CALL getLastExceptionDetail(InputExceptionList, mNumber, mText);

-- Track the number of messages that error and add this counter
-- to fields tracked in RFH2 USR folder
BEGIN ATOMIC
  SET s_Counter = s_Counter + 1;
  set OutputRoot.XMLNSC.ExceptionDump.ErrCounter = s_Counter;
  SET OutputRoot.XMLNSC.ExceptionDump.ErrList = InputExceptionList;
  set Environment.Variables.ErrCounter = s_Counter;
  Set Environment.Variables.errorNum = mNumber;
  Set Environment.Variables.errorReason = mText;
```

__ 7. Close the message in the Flow exerciser.

__ 8. In the CaptureError Compute node, the ESQL code writes the exception information to the **MQRFH2.usr** folder.

Use RFHUtil to verify contents of the **MQRFH2.usr** folder.

__ a. Start RFHutil by double-clicking the shortcut icon on the desktop.

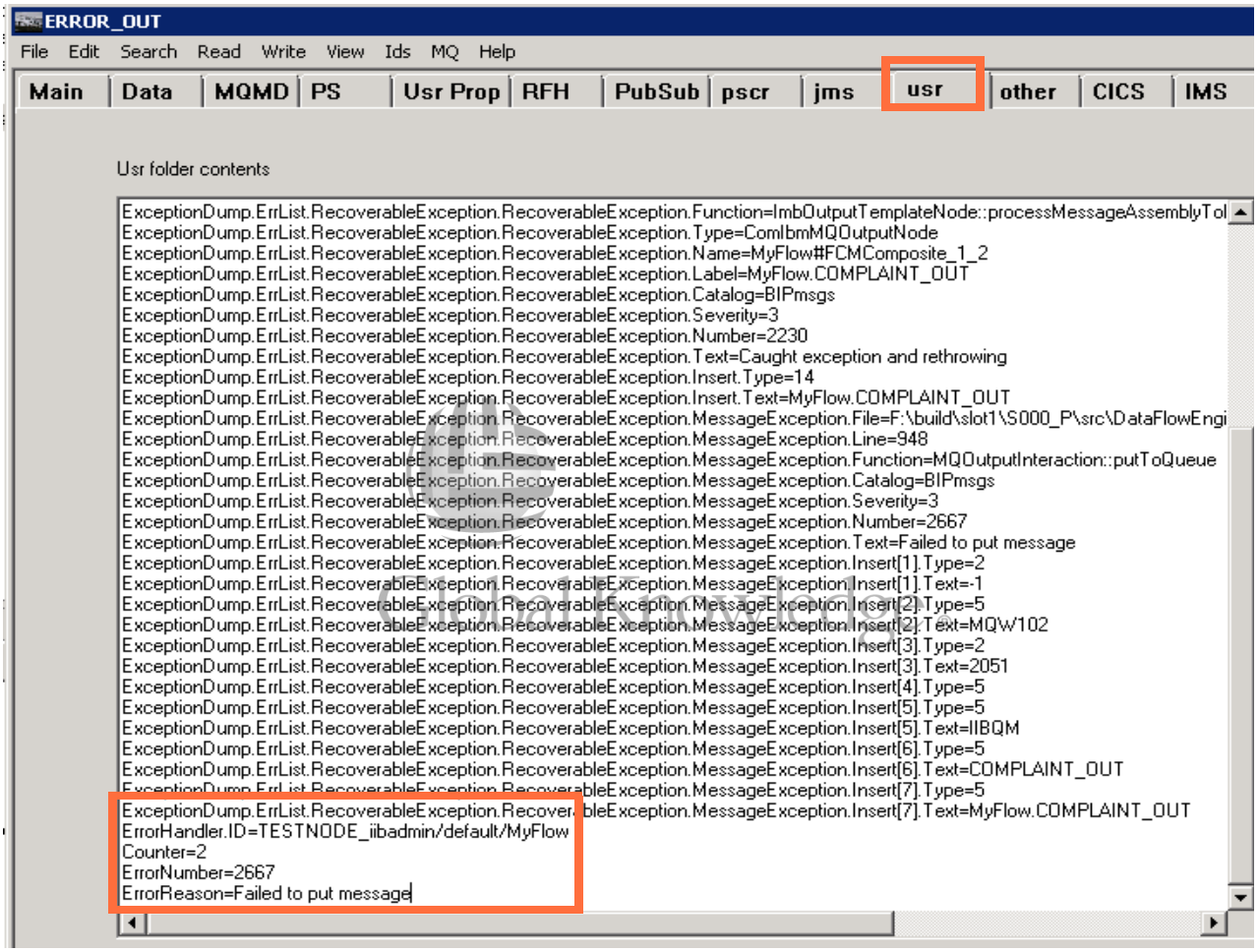
__ b. On **Main** tab, set **Queue manager name** to **IIBQM** and **Queue name** to **ERROR_OUT**.

__ c. Click **Browse Q**.

__ d. Click the **Data** tab to view the original message.

__ e. Click the **usr** tab to display the contents of **MQFRH2.usr** folder.

__ f. Scroll to bottom of the **Usr folder contents** pane to see the error summary that the **Capture Error** Compute node creates.



__ g. Close RFHutil.

Test 3: Testing the TryCatch node in the subflow

In this test, you change the test environment to cause the **Try** path of the Try Catch node to fail. This test shows how the ErrorHandler_Subflow responds to nested errors.

In this test, you inhibit PUT operations to both the COMPLAINT_OUT queue and the ERROR_OUT queue.

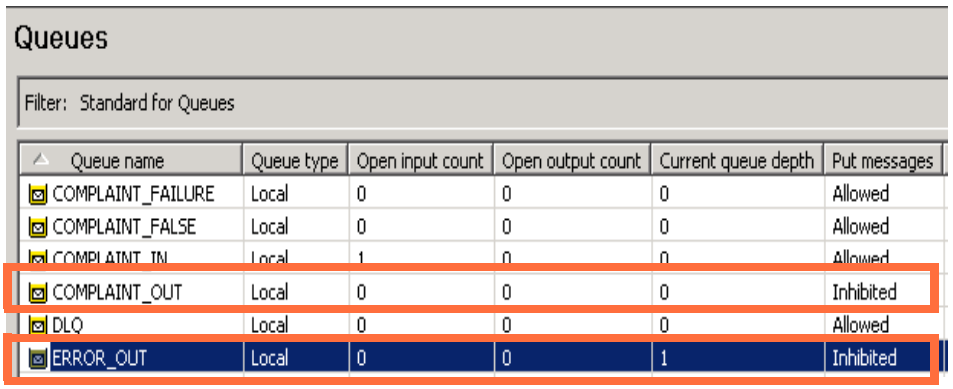
__ 1. Set the ERROR_OUT queue to inhibit PUT operations.

__ a. In IBM MQ Explorer, right-click the ERROR_OUT queue in the **Queues** view, and then click **Properties**.

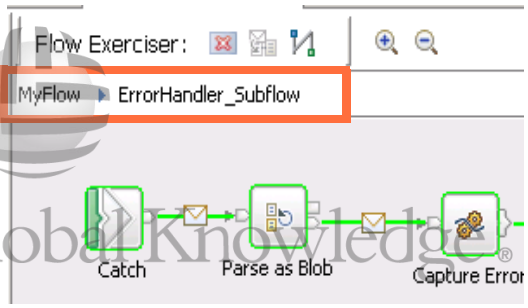
__ b. On the **General** tab, set the **Put messages** property to **Inhibited**.

__ c. Click **Apply** and then click **OK**.

__ 2. In the list of queues, both ERROR_OUT and COMPLAINT_OUT should now show that the **Put messages** property is now **Inhibited**.



__ 3. In the Flow exerciser, return to main flow by clicking **MyFlow** in Flow exerciser navigator.



__ 4. Send the test message again.

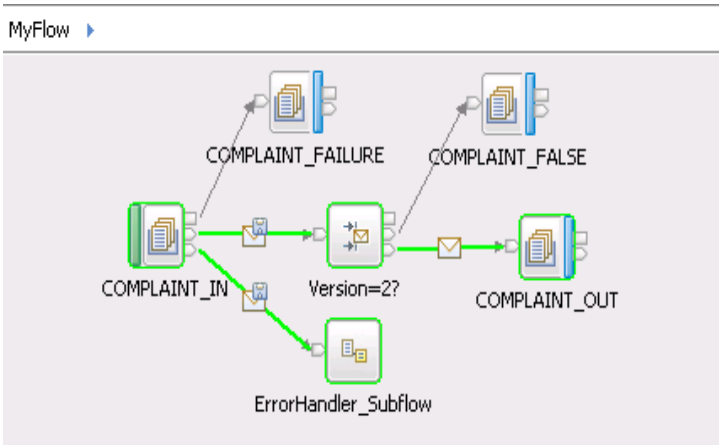
__ a. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

__ b. On the **Send Message** window, click **Complaint** under the **Input Messages** folder.

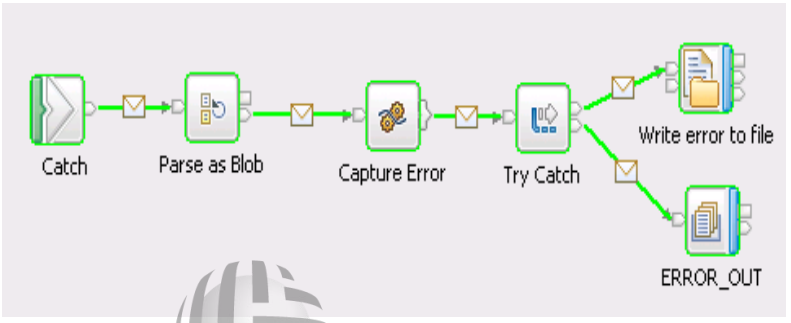
__ c. Click **Send**.

__ 5. The Progress Information window shows the status. Wait for a timeout event and notification that the Flow exerciser stopped listening for a response. Click **Close** to view the path.

__ 6. Similar to Test 2, you can see that an attempt is made to send the message to the COMPLAINT_OUT queue. The **Catch** path from the COMPLAINT_IN node to the ErrorHandler_subflow is also highlighted.



__ 7. Double-click the **ErrorHandler_Subflow** node in the main flow to display the subflow path.

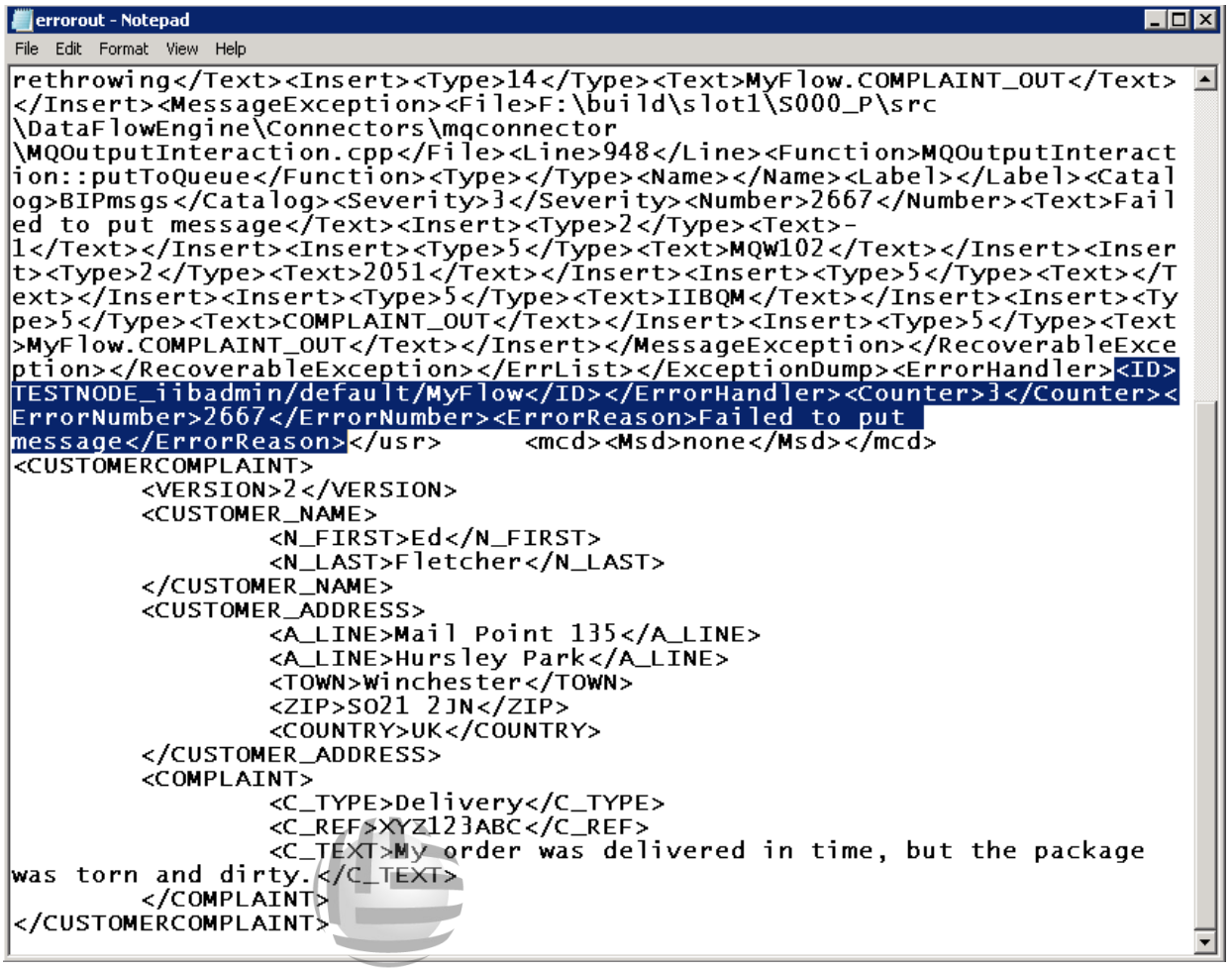


This time, when the TryCatch node in the ErrorHandler_Subflow attempts to write the error message to ERROR_OUT, it fails. So, as shown in the highlighted path, the message is sent down the **Catch** path of the Try Catch node to the **Write error to file** FileOutput node.

__ 8. Verify that the exception information is written to the output file: `C:\labfiles\errorout.txt`

__ 9. Open the file in Notepad to examine the contents.

The file contains the identifier information that the Compute node adds to the exception information and the original message.



Exercise clean-up

__ 1. In the Message Flow editor, click the Flow exerciser icon to stop the Flow exerciser and return to edit mode.

__ 2. Close all the Message Flow editor tabs.

__ 3. In the **Integration Nodes** view, right-click the **default** integration server and click **Stop Recording**.

__ 4. In the **Integration Nodes** view, right-click the **default** integration server and click **Delete > All Flows and Resources**.

__ 5. In IBM MQ Explorer, change the **Put messages** property on the COMPLAINT_OUT and ERROR_OUT queues to **Allowed**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you created the error handler subflow that uses the ResetContentDescriptor node and the Try Catch node.

In the second part of this exercise, you added the error handler subflow to the main flow.

In the third part of this exercise, you used the IBM Integration Toolkit Flow exerciser to test the message flow and the error handler subflow.

After completing this exercise, you should be able to:

- Implement a generic error handling routine in the form of a subflow
- Use a ResetContentDescriptor node to force the message to be reparsed according to the parser domain that is specified in the node properties
- Use the TryCatch node to provide a special handler for exception processing

Exercise 9. Referencing a database in a map

What this exercise is about

In this exercise, you use a Database node in a message flow to store a message in a database. You also import a COBOL Copybook and XML schema to create a data model, and use the Graphical Data Mapping editor to transform the message.

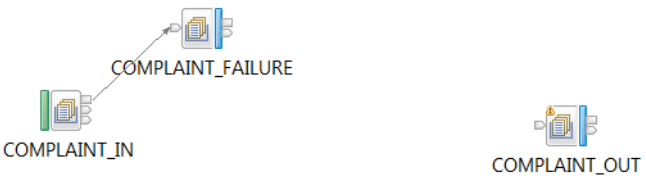
What you should be able to do

After completing this exercise, you should be able to:

- Create a shared library that contains data models that describe the input and output data
- Import a COBOL Copybook to create a DFDL schema file
- Reference a shared library in a message flow application
- Discover database definitions
- Add a Database node to a message flow
- Create the logic to store a message in a database
- Use the Graphical Data Mapping editor to map message elements
- Reference an external database when mapping message elements

Introduction

In this exercise, you extend a message flow that processes a complaint message. The message flow starts with an MQ Input node that gets the message from the COMPLAINT_IN queue. It ends with an MQ Output node that puts the message to a COMPLAINT_OUT node.



The input message is a COBOL Copybook file. An example of the file is shown here.

```
2Ed        Fletcher  Mail Point 135      Hursley Park
WinchesterSO21 2JN  UKDelivery  XYZ123ABC I placed an order on
15-11-99, well in time for Christmas and I still have not had a
```

```
delivery schedule sent to me.  Please cancel the order and refund me
NOW.
```

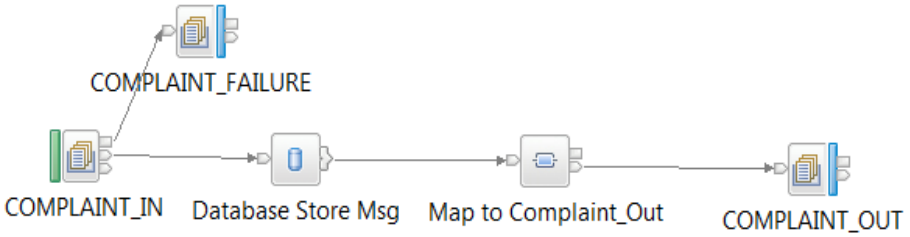
The output message is an XML message that the Complaint_Out.xsd schema defines. An example of the output message is shown here.

```
<Complaint_Out>
   <Version>2</Version>
   <Customer>
   <Name>
      <First>Ed</First>
      <Last>Fletcher</Last>
   </Name>
   <Address>
      <Line>Mail Point 135</Line>
      <Line>Hursley Park</Line>
      <Town>Winchester</Town>
      <Zip>SO21 2JN</Zip>
      <Country>UK</Country>
   </Address>
   <Complaint>
      <Type>Delivery</Type>
      <Reference>XYZ123ABC</Reference>
      <Text>I placed an order on 15-11-99, well in time for Christmas
and I still have not had a delivery schedule sent to me.  Please cancel
the order and refund me NOW.</Text>
   </Complaint>
   <Admin>
      <Reference>COMda1b71b2</Reference>
      <Manager>
         <FirstName>SALLY A.</FirstName>
         <LastName>Kwan</LastName>
      <Phone>4738</Phone>
      </Manager>
      <Text>Please action</Text>
   </Admin>
</Complaint_Out>
```

In the first part of this exercise, you create a shared library and then create the data models for the input and output messages in the shared library.

In the next parts of the exercise, you complete the message flow so that when the message arrives in the flow, it is stored in the MSGSTORE database for auditing purposes. The message flow also enhances the output message with information about the manager of the department that is to handle the complaint. This data is selected from the EMPLOYEE table in the

SAMPLE database. Finally, the output message is written to the COMPLAINT_OUT queue.



Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit, IBM MQ V8, and IBM DB2
- A local IBM MQ queue manager that is named IIBQM with the following local queues: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_OUT, and DLQ
- Lab files in the `C:\labfiles\Lab09-DBMap` directory
- User `iibadmin` that is a member of the “mqm” group
- Users `iibadmin` and `Administrator` in the DB2ADMS and DB2USERS groups
- The EMPLOYEE table in the SAMPLE database and the COMPLAIN table in the MSGSTORE database

Note

The DB2 databases, tables, and ODBC data sources that are required for this exercise are already created on the lab exercise image.

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab09`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab09`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, verify that the integration node **TESTNODE_iibadmin** is started.

Start it if it is stopped.

__ 4. This exercise requires an IBM MQ queue manager.

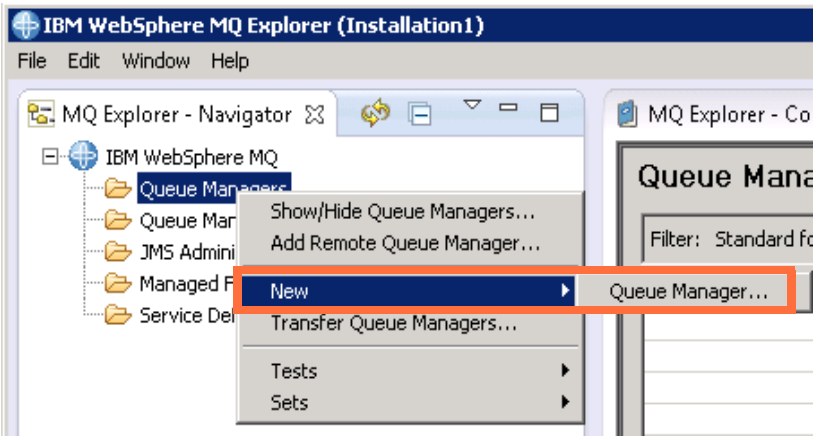
If you created a queue manager that is named **IIBQM** in a previous exercise, proceed to Step 5.

If you do not have a queue manager that is named **IIBQM**, create it now by following these instructions.

__ a. Start IBM MQ Explorer.

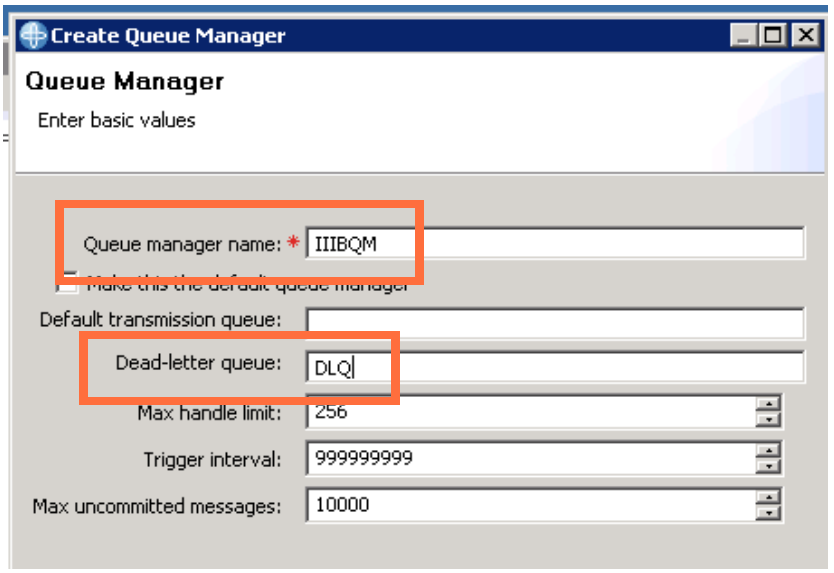
From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

__ b. In the **MQ Explorer Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ c. For the **Queue Manager** name, type: **`IIBQM`**

__ d. For the **Dead-letter queue**, type: **DLQ**



__ e. Click **Finish**.

After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 5. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

In a command window, type:

```
runmqsc IIBQM < C:\labfiles\Lab09-DBMap\CreateQueues.mqsc
```

__ 6. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the IBM MQ Explorer Navigator view, expand the **IIBQM** folder.

__ b. Click **Queues** under the queue manager in the **MQ Explorer - Navigator** view to display the **Queues** view.

__ c. Verify that the following queues are listed in the **Queues** view: COMPLAINT_FAILURE, COMPLAINT_IN, COMPLAINT_OUT, and DLQ.

Note

You might have other queues for this queue manager from a previous exercise. You do not need to delete the unused queues.

__ 7. In the IBM Integration Toolkit, import the starting application from the `DB_StartingApp_PI.zip` file in `C:\labfiles\Lab09-DBMap` directory.

Select the **RouteComplaint** folder.

The **RouteComplaint** application contains a partial message flow that is named **DB.msgflow**.

Part 1: Create a shared library that contains the data models

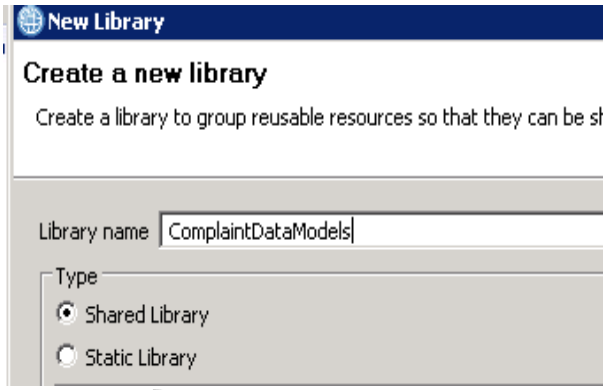
In this part of the exercise, you create a shared library and then create the data models for the COBOL Copybook that defines the input file and the XML schema that defines the output file. You then reference the shared library in the RouteComplaint application.

__ 1. Create a shared library that is named **ComplaintDataModels**.

__ a. In the IBM Integration Toolkit, click **File > New > Library**.

__ b. Ensure that **Shared Library** is selected.

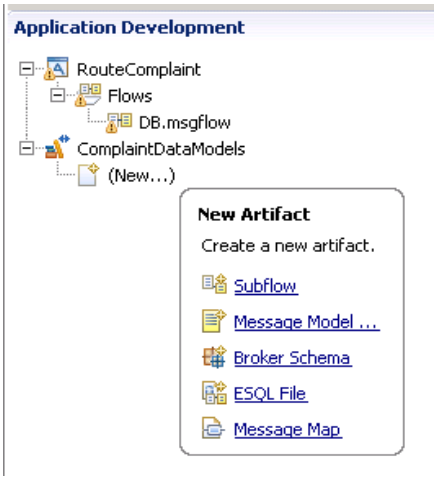
__ c. For the **Library Name**, type: **ComplaintDataModels**



__ d. Click **Finish**.

__ 2. Import the COBOL copybook to create a DFDL schema from the COMPLAINT_IN.cpy COBOL Copybook file in the C:\labfiles\Lab09-DBMap\data directory.

__ a. Under the **ComplaintDataModels** library folder in the **Application Development** view, click **New > Message Model** to start the New Message Model wizard.



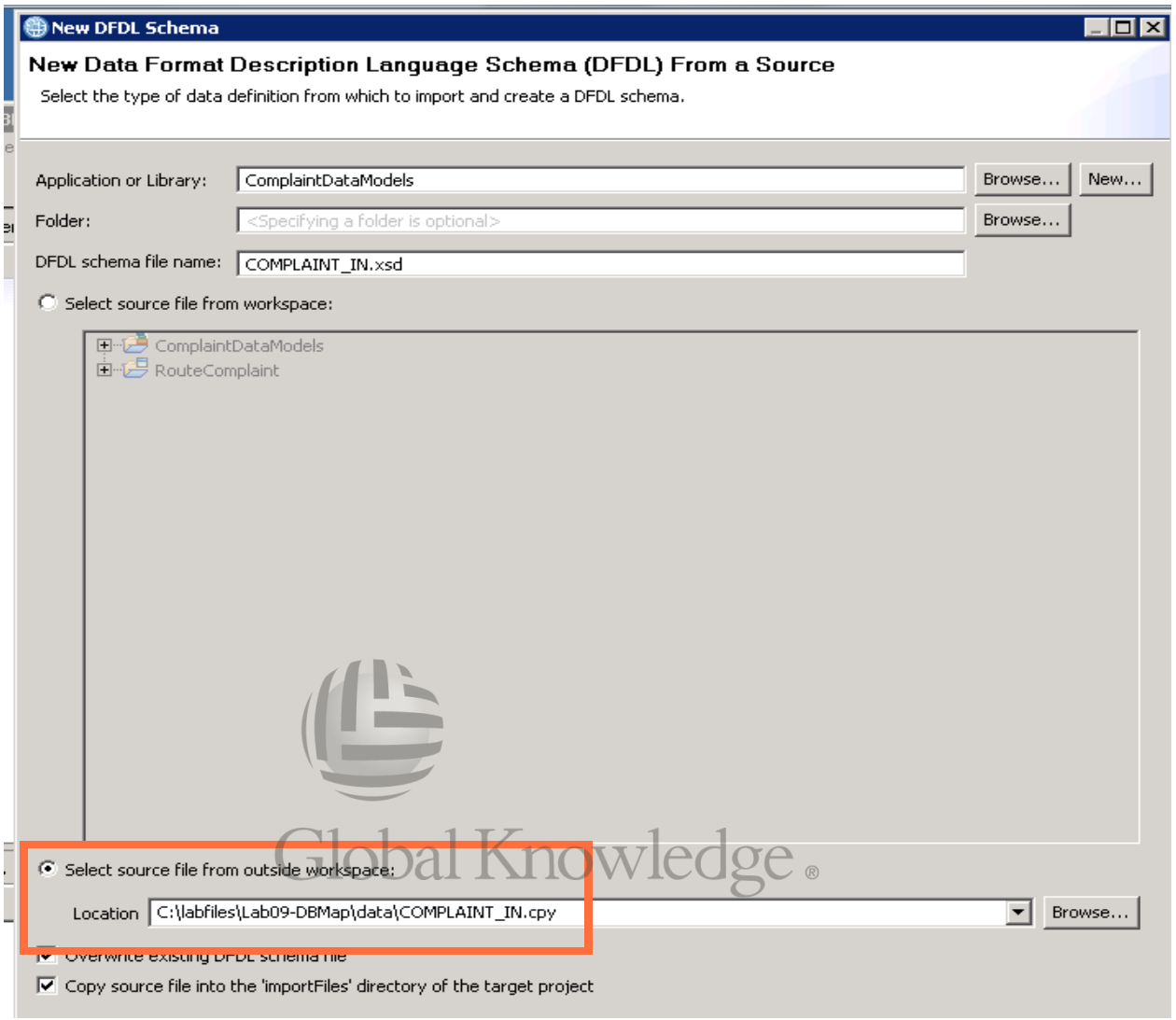
__ b. Click **COBOL** under the **Text and binary** section and then click **Next**.

__ c. Click **Create a DFDL schema file by importing a COBOL Copybook or program** and then click **Next**.

__ d. Click **Select source from outside workspace**.

__ e. To the right of the **Location** field, click **Browse**.

__ f. Browse to `C:\labfiles\Lab09-DBMap\data` directory, click **COMPLAINT_IN.cpy**, and then click **Open**.

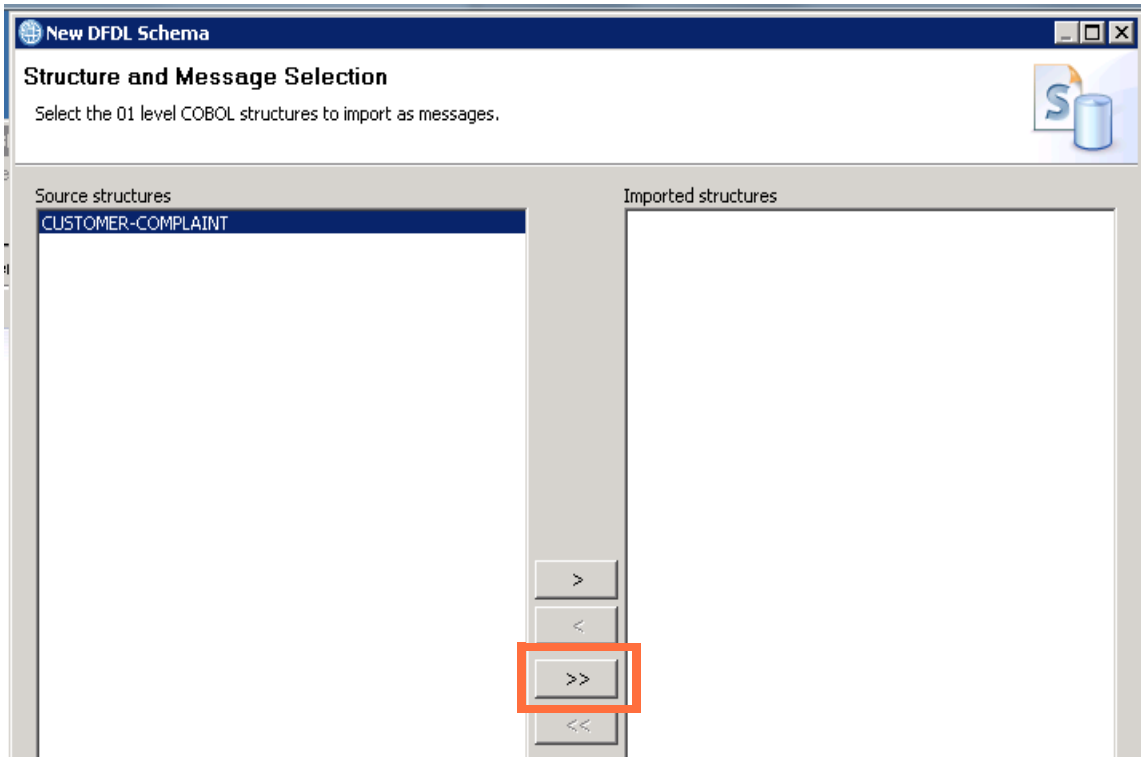


__ g. Click **Next**. The COBOL Copybook is read and analyzed.

__ h. You must select which source 01 level structures to import, and whether to create a message from them. For this exercise, the copybook contains only one 01 level structure.

On the **Structure and Message Selection** page, verify CUSTOMER_COMPLAINT is shown in the **Source structures** pane.

__ i. Click CUSTOMER_COMPLAINT under **Source structures** and then click **>>** to move CUSTOMER_COMPLAINt under **Imported structures**.



__ j. Click **Finish**.

The DFDL schema **COMPLAINT_IN.xsd** is generated and is opened in the DFDL schema editor.

__ 3. Review the **COMPLAINT_IN.xsd** schema so that you understand its structure and elements.

Close the schema editor when you are finished reviewing the schema.

__ 4. Create a message model in the **ComplaintDataModels** library from the XML schema `COMPLAINT_OUT.xsd` that is in the `C:\labfiles\Lab09-DBMap\data` directory.

__ a. In the **Application Development** view, right-click **Schema Definitions** under the **ComplaintDataModels** library folder and then click **New > Message Model**.

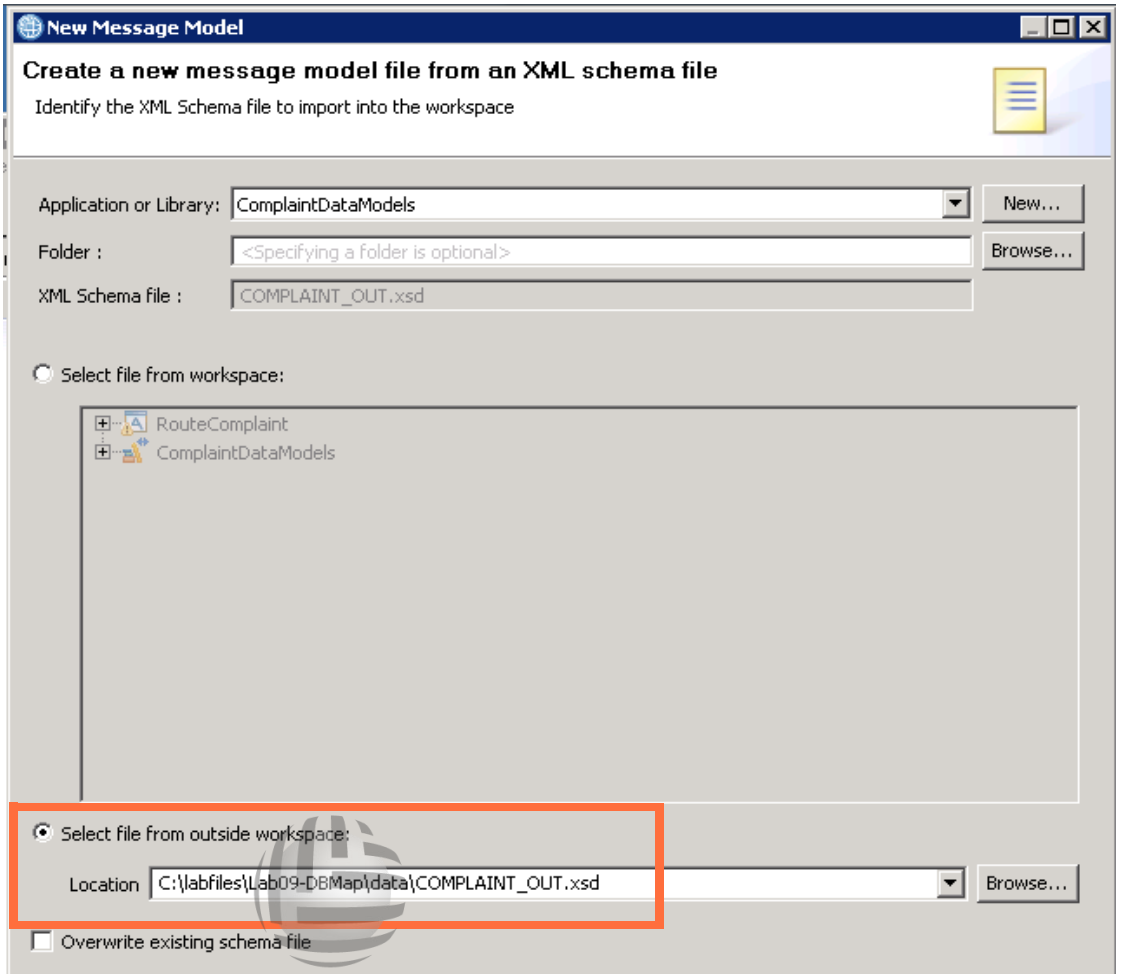
The New Message Model window is displayed.

__ b. Under the **XML** section, click **Other XML** and then click **Next**.

__ c. On the **Other XML** page, click **I already have an XML schema file for my data** and then click **Next**.

__ d. Click **Select file from outside workspace** and then click **Browse**.

__ e. Browse to `C:\labfiles\Lab09-DBMap\data` directory, click **COMPLAINT_OUT.xsd**, and then click **Open**.



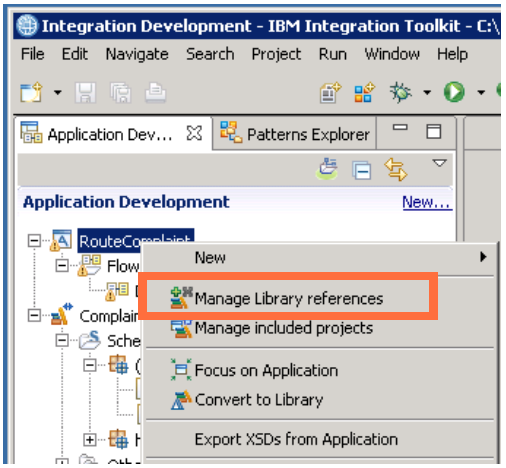
__ f. Click **Finish**. The XML schema is read and analyzed.

__ 5. Review the newly created message definition **COMPLAINT_OUT.xsd** in the **Outline** view. The COMPLAINT_OUT.xsd file also opens in the schema editor.

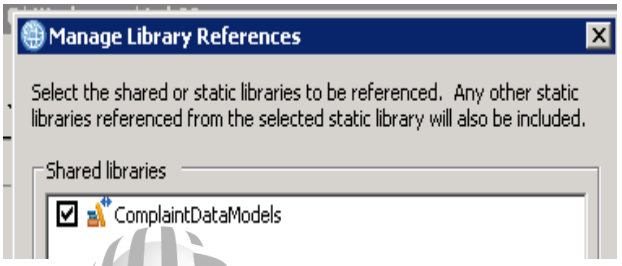
Close the schema editor when you are finished reviewing the schema.

__ 6. Reference the **ComplaintDataModels** shared library in the **RouteApplication** application.

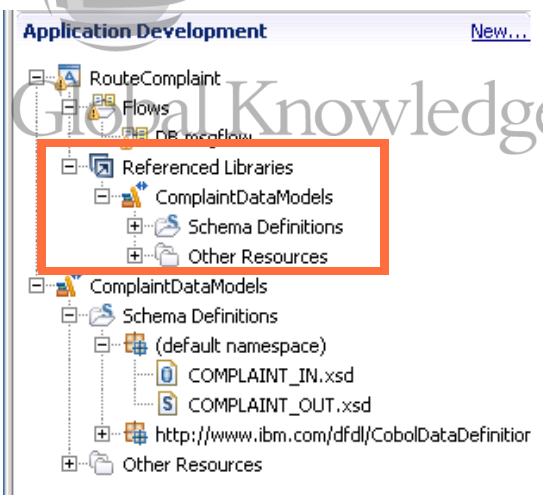
__ a. In the Application Development view, right-click **RouteComplaint** and then click **Manage library references**.



__ b. Click **ComplaintDataModels** and then click **OK**



__ c. A library reference is added to the **RouteComplaint** application.



Part 2: Discover the database definition

In this part of the exercise, you create the configurable services that define the JDBC connections to the DB2 databases for this exercise. You then define the connections to the SAMPLE and MSGSTORE databases in the IBM Integration Toolkit.

Note

The DB2 database tables are already created on the lab exercise image.

__ 1. Create the JDBC configurable services that define the JDBC connections to the DB2 SAMPLE and MSGSTORE databases.

__ a. Open a Command Prompt window.

__ b. Change directories to the `C:\labfiles\Lab09-DBmap` directory. Type:

```
cd C:\labfiles\Lab09-DBMap
```

__ c. Run the **ConfigureJDBC** command to create the JDBC configurable service for the SAMPLE database. Type:

```
ConfigureJDBC TESTNODE_iibadmin SAMPLE
```

__ d. Run the **ConfigureJDBC** command to create the JDBC configurable service for the MSGSTORE database. Type:

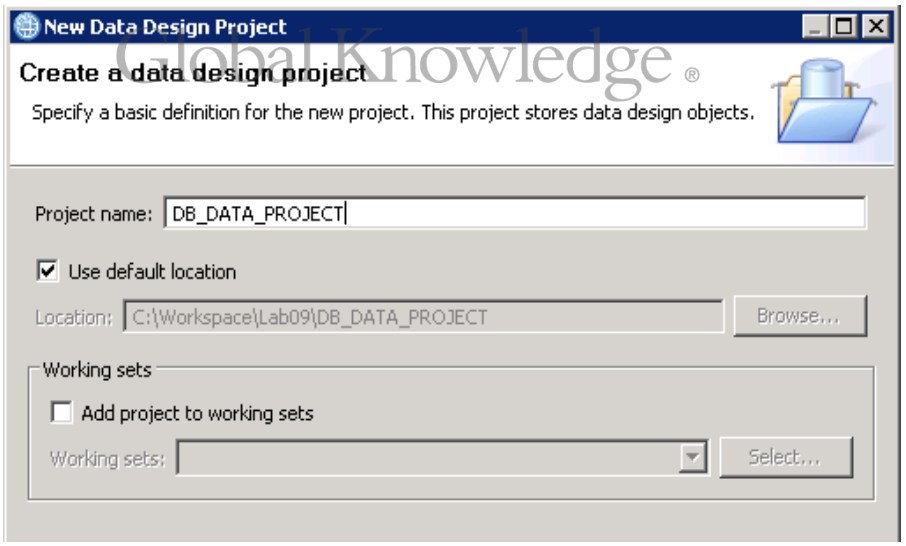
```
ConfigureJDBC TESTNODE_iibadmin MSGSTORE
```

__ 2. In the Integration Toolkit, discover the database definition for the DB2 database SAMPLE, in the ADMINISTRATOR schema, and store it in the DB_DATA_PROJECT.

__ a. Click **File > New > Database Definition**. The **New Database Definition File** window opens.

__ b. To the right of the **Data design project** field, click **New** to create a data design project.

__ c. For the **Project name**, type `DB_DATA_PROJECT`



__ d. Click **Finish**.

__ e. On the **Create a data design project** page, click **Next**.

__ f. On the **Select Connection** page, click **New**.

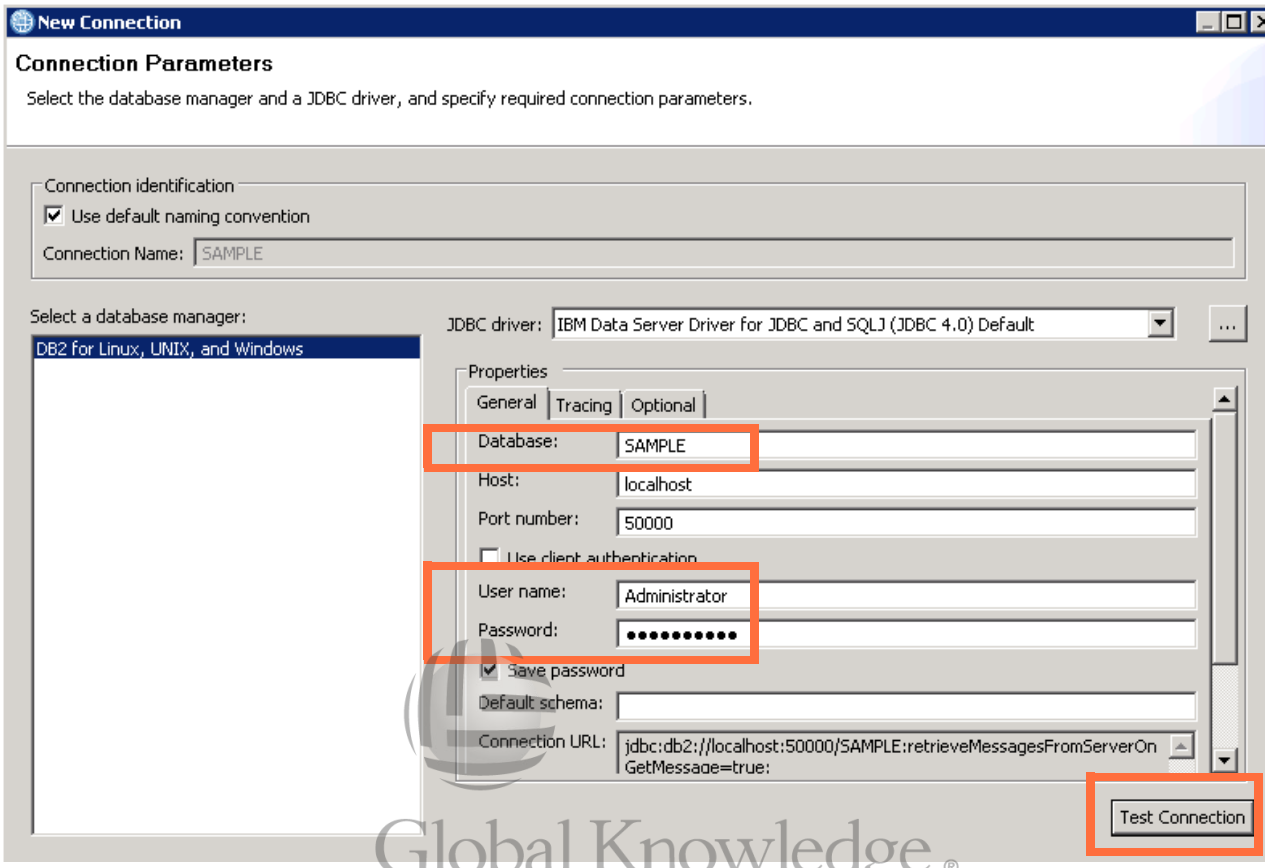
__ g. In the **Properties** pane, for **Database**, type: **SAMPLE** (if it is not already entered)

__ h. For **User name**, type: **Administrator**

__ i. For **Password**, type: **web1sphere**

__ j. Click **Save password**.

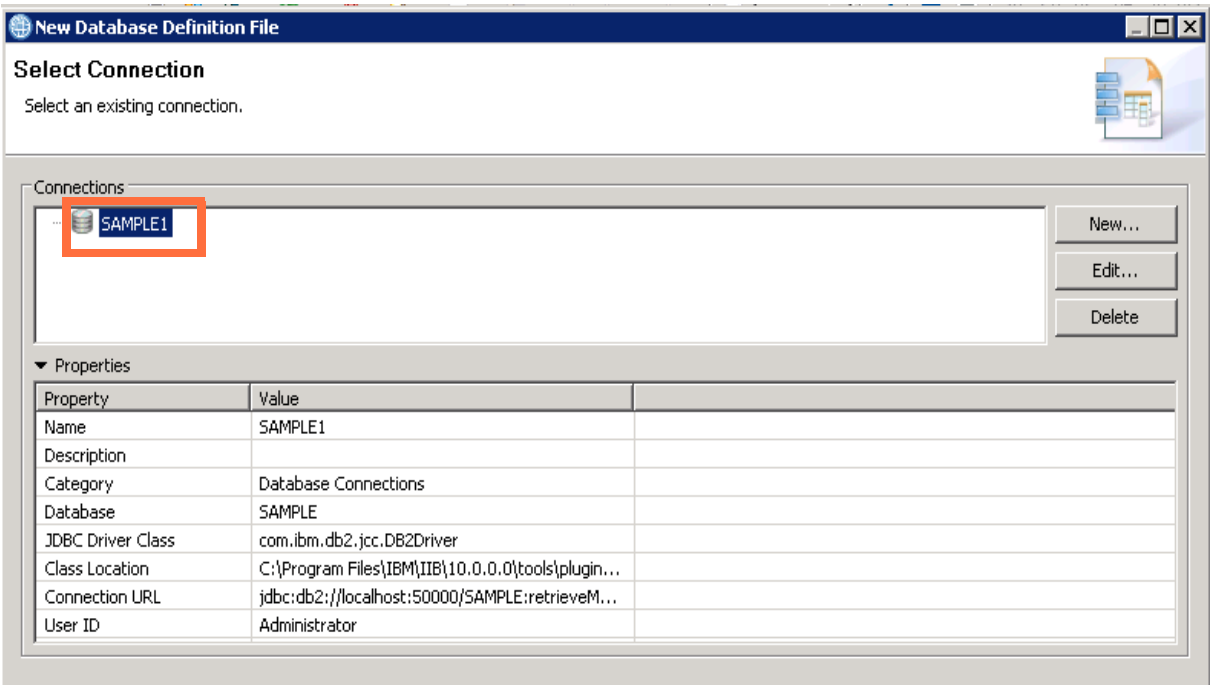
__ k. Click **Test Connection** to test JDBC connection is tested.



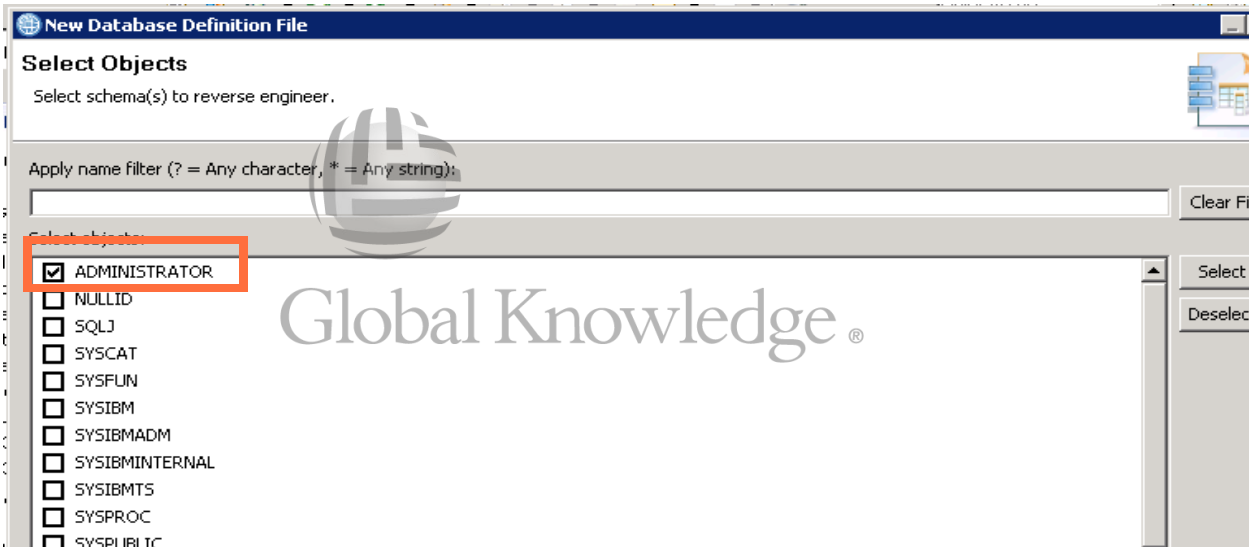
__ l. A “Connection succeeded” message should be displayed. Click **OK**.

__ m. Click **Finish**. The **New Database Definition File** window displays the connection information for the SAMPLE database.

__ n. Click **SAMPLE1**, and then click **Next**.



__ o. Select **ADMINISTRATOR** and then click **Finish**.



__ 3. The model is created. The database connection opens in the Physical Model data editor.

Review the model, but do not change anything. Close the editor when you are finished reviewing the model.

__ 4. Discover the database definition for the DB2 database MSGSTORE in the ADMINISTRATOR schema, and store it in DB_DATA_PROJECT.

__ a. Click **File > New > Database Definition**. The New Database Definition File menu is shown.

__ b. Select **DB_DATA_PROJECT** for the **Project name** and then click **Next**. The **Select Connection** window opens.

__ c. Click **New**.

__ d. In the **Properties** pane, for **Database**, type: **MSGSTORE**

__ e. For **User name**, type: **Administrator**

__ f. For **Password**, type: **web1sphere**

__ g. Click **Save password**.

__ h. Click **Test Connection**. The JDBC connection is tested.

__ i. A “Connection succeeded” message is shown. Click **OK**.

__ j. Click **Finish**. The Select Connection menu reappears.

__ k. In the **Connections** pane, click **MSGSTORE**, and then click Next. The Select Schema window opens.

__ l. Click **ADMINISTRATOR**, and then click **Finish**. The model is created.

__ 5. The database connection opens in the Physical Model data editor.

Review the model, but do not change anything. Close the editor when you are finished reviewing the model.

Information

Database definition files are associated with the Data Project Explorer view and the Database Explorer view. Tools are available in these views for working with your databases.

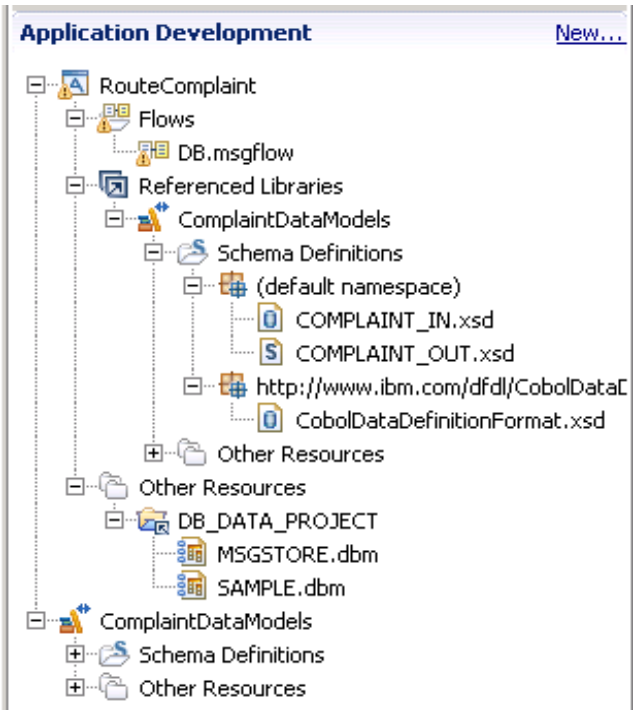
Database definition files are not automatically updated. If you change the database, you must re-create the database definition files.

__ 6. The message flow **DB.msgflow** needs access to the database definitions that are stored in DB_DATA_PROJECT. Add a reference to the DB_DATA_PROJECT.

__ a. Right-click the **RouteComplaint** application in the **Application Development** view and then click **Manage included projects**.

__ b. Click **DB_DATA_PROJECT** and then click **OK**.

__ c. The DB_DATA_PROJECT folder is now included in the **RouteComplaint** application.



Part 3: Complete the message flow

In this part of the exercise, you complete the message flow by adding a Database node and a Mapping node. You also configure the Database node to store the message in the COMPLAIN table in the MSGSTORE database.

__ 1. In the **Application Development** view, expand **RouteComplaint > Flows**, and then double-click the **DB.msgflow** file to open it in the Message Flow editor.

As you can see, some nodes are missing from the message flow.

__ 2. Add a Database node from the **Database** drawer and rename it to: **Database Store Msg**.

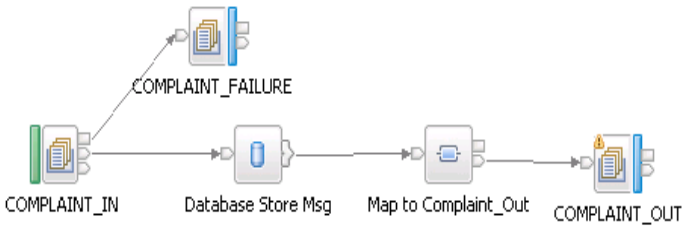
__ 3. Add a Mapping node from the **Transformation** drawer and rename it to: **Map to Complaint_Out**.

__ 4. Connect the new nodes.

__ a. Wire the **COMPLAINT_IN** node **Out** terminal to the **Database Store Msg** node **In** terminal.

__ b. Wire the **Database Store Msg** node **Out** terminal to the **Map to Complaint_Out** node **In** terminal.

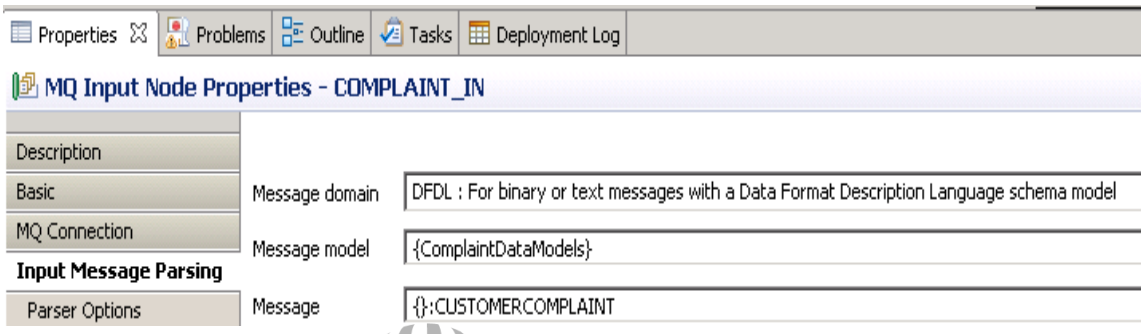
__ c. Wire the **Map to Complaint_Out** node **Out** terminal to the **COMPLAINT_OUT** node **In** terminal.



__ 5. Configure the COMPLAINT_IN node **Input Message Parsing** properties to use the DFDL model that you imported in Part 1 of this exercise.

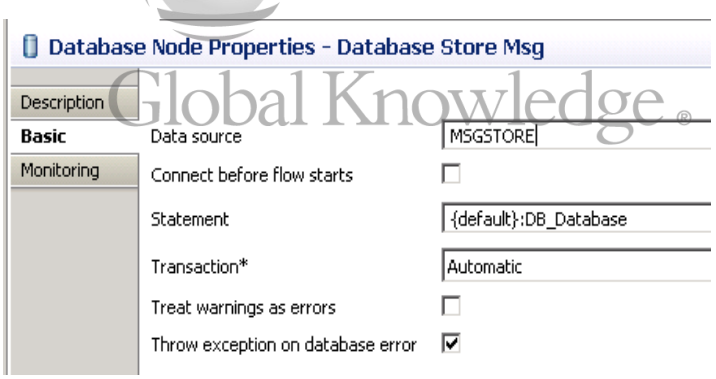
__ a. For the **Message domain,** select **DFDL**.

__ b. For the **Message**, click **Browse** and then click **CUSTOMERCOMPLAINT**.



__ 6. Configure the **Database Store Msg** node properties.

On the **Basic** properties tab, for **Data source** type: **`MSGSTORE`**



__ 7. Provide the ESQL for the **Database Store Msg** node.

__ a. Double-click the **Database Store Msg** node. The ESQL editor opens in a new module within the existing ESQL file.

__ b. The ESQL code for this node inserts a row into the COMPLAIN table. The COMPLAIN table consists of the MESSAGE, MSGID, and RECEIVED columns, the complete message bit stream, and the Message ID and time stamp for identification.

Copy the ESQL from `C:\labfiles\Lab09-DBMap\Cut&Paste.txt` file and paste it into the ESQL after the BEGIN statement.

The completed module should contain the following code:

```
CREATE DATABASE MODULE DB_Database
    CREATE FUNCTION Main() RETURNS BOOLEAN
      BEGIN
          INSERT INTO Database.ADMINISTRATOR.COMPLAIN(MSGID, RECEIVED, MESSAGE)
         VALUES (Root.MQMD.MsgId,CURRENT_TIMESTAMP,ASBITSTREAM(Root));
    RETURN TRUE;
     END;
END MODULE;
```

__ c. Save the file and close the ESQL editor.

Note

Warning messages in the **Problems** view about ambiguous database table references are OK; the references are resolved at run time.

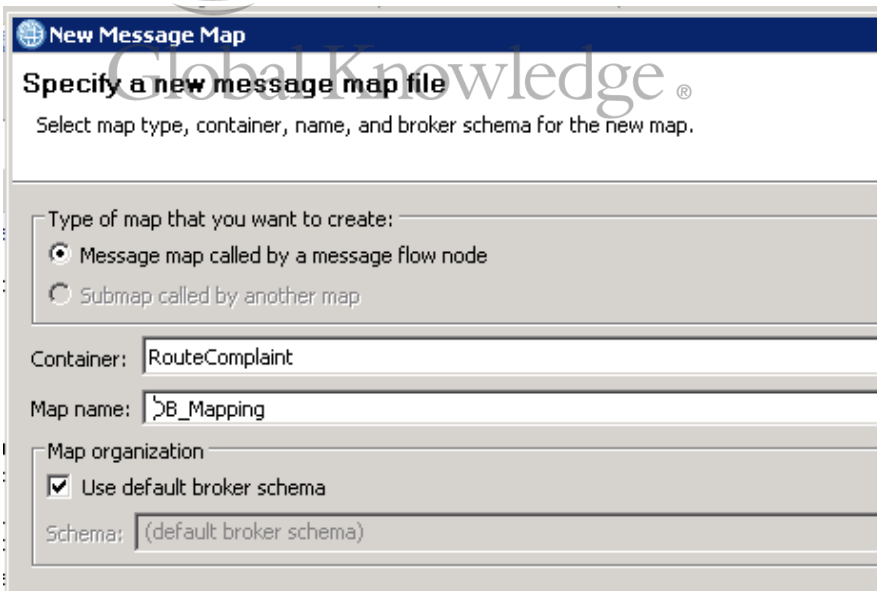
Part 4: Create the mappings

In this part of the exercise, you use the Graphical Data Mapping editor to map the input message to the output message.

__ 1. Create the map and define the map source and target.

__ a. Double-click the **Map to Complaint_Out** Mapping node in the message flow. The **New Message Map** window opens.

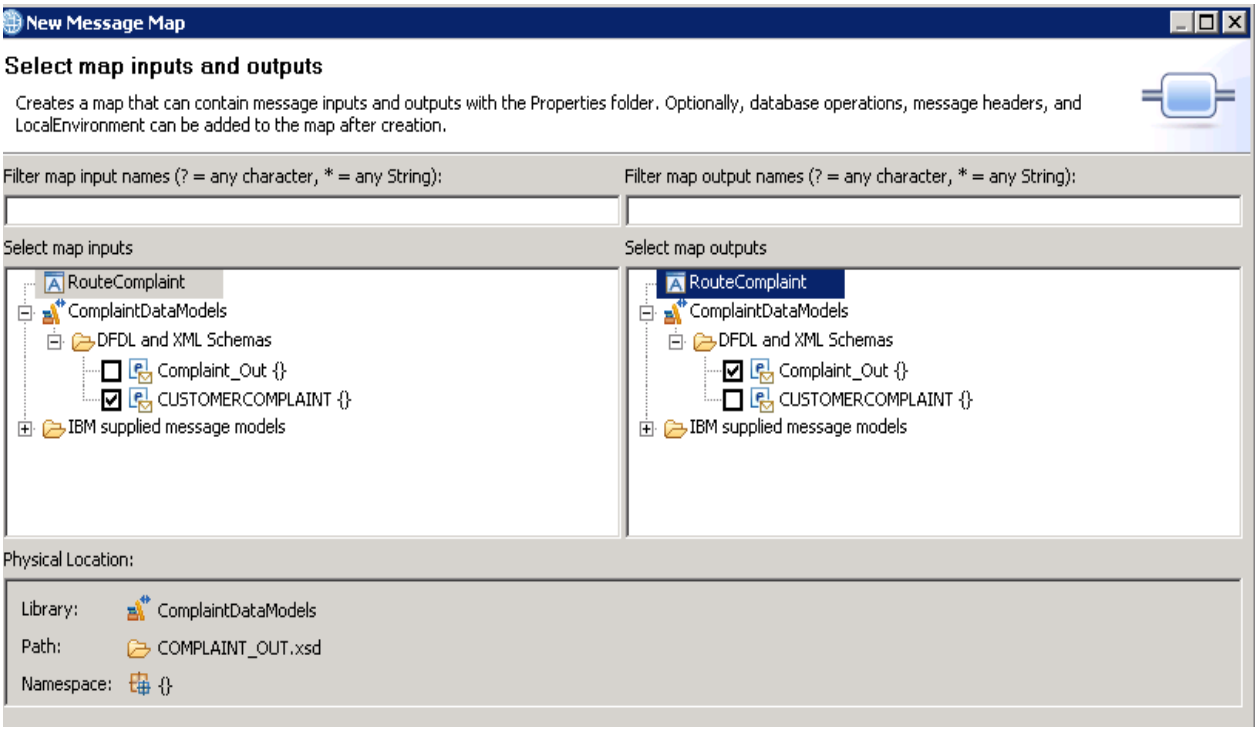
__ b. Accept the default values on this page for **Container** and **Map name** and click **Next**.



The **“**Select map inputs and outputs” window opens.

__ c. Under **Select map inputs**, expand **ComplaintDataModels > DFDL and XML schemas** and then click **CUSTOMERCOMPLAINT { }**.

__ d. Under **Select map outputs**, expand **ComplaintDataModels > DFDL and XML schemas**, and then select **Complaint_Out { }**.



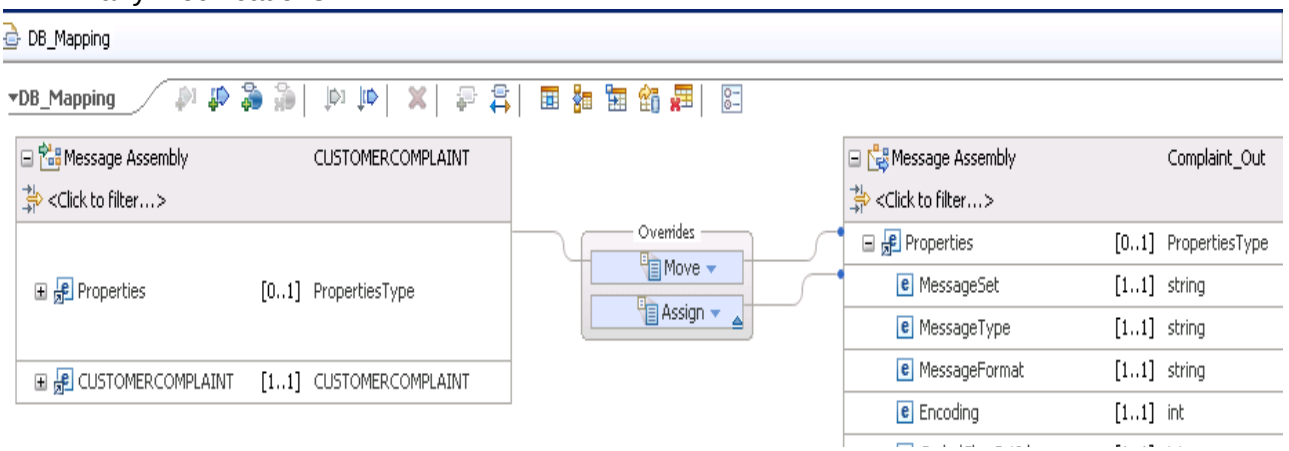
__ e. Click **Finish**. The Mapping editor opens.

__ 2. You must define how the source elements map to the target elements.

When you populate the message map, the **Properties** folder for the source and target are displayed in the message map.

View the map **Properties** transforms.

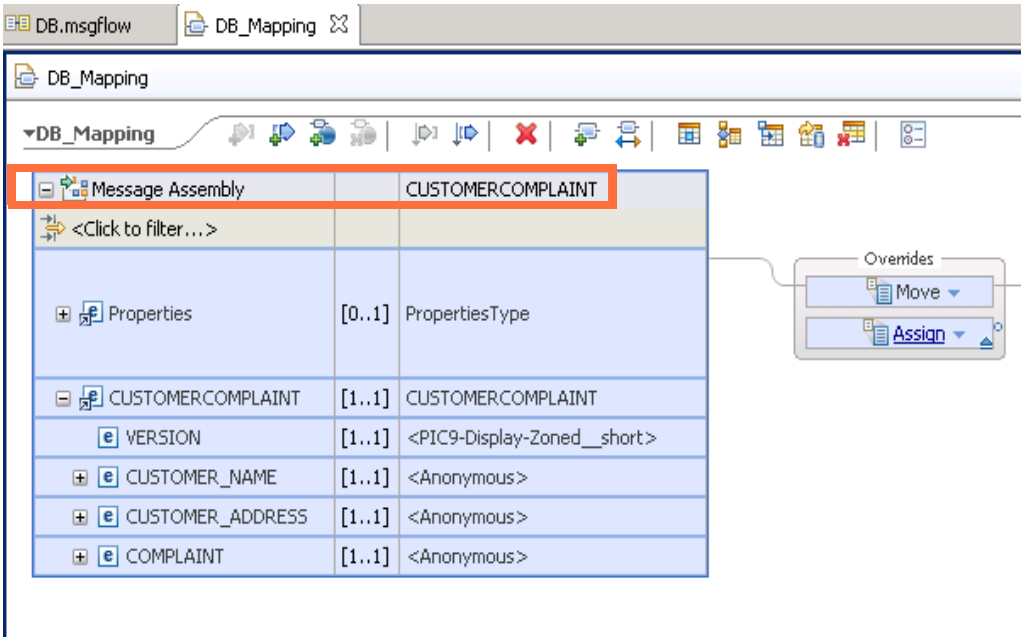
Observe that a **Move** transform is already defined between the source and target **Properties**. The **Move** transform maps the source message directly to the target without any modifications.



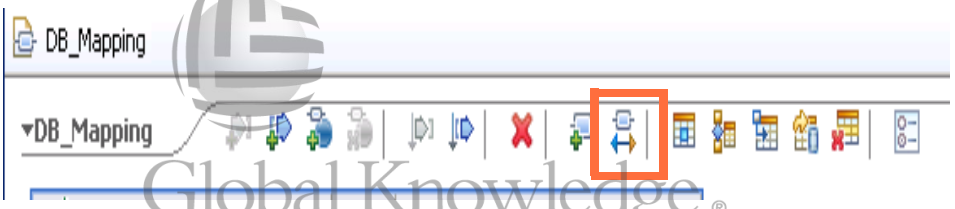
The **Assign** transform, sets the **MessageSet** on the output to the library name **ComplaintDataModels**.

__ 3. Map the first part of the message body. Since many of the field names in the source and target are similar, use the auto map feature to quickly map the source structure to the target structure.

__ a. Click the CUSTOMERCOMPLAINT message assembly in the source structure to select the entire message. The message turns blue to indicate that it is selected.

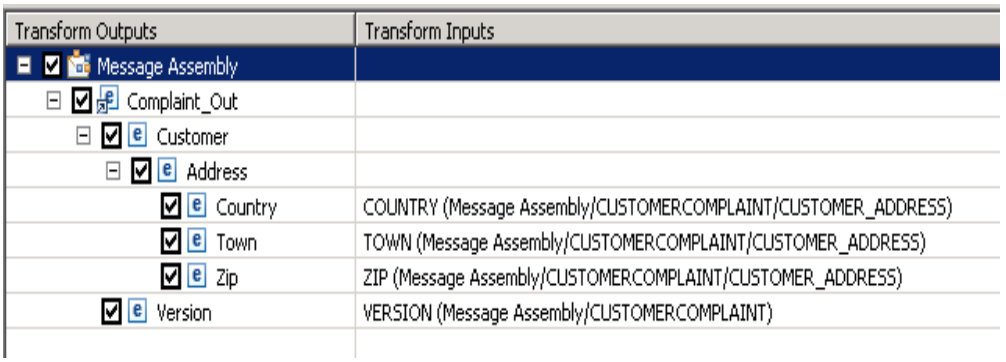


__ b. Click the **Auto map input to output** icon in the Mapping editor toolbar.



The AutoMap properties window opens.

__ c. Without changing any of the parameters, click **Next**. A preview of the mapped fields is displayed.

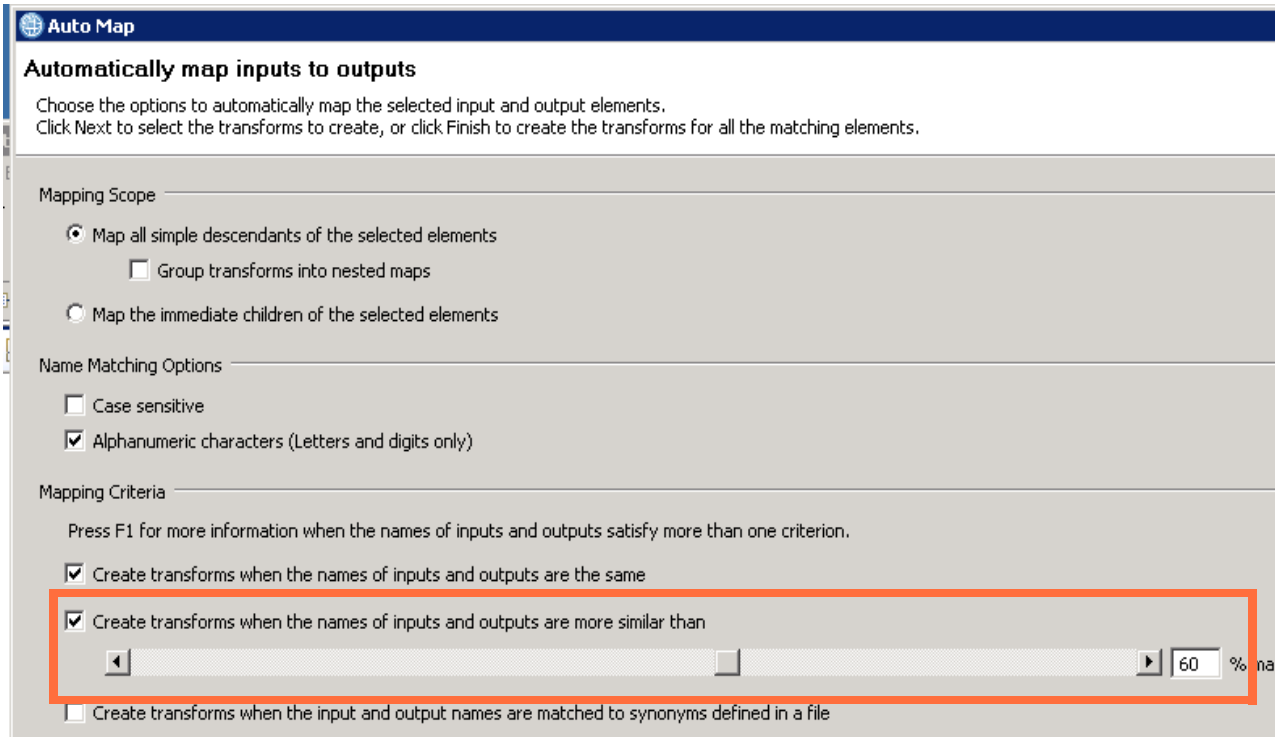


__ d. You see that only the fields that exactly matched are mapped. You can control this “strictness” in the matching so that names do not need to match exactly.

Click **Back**.

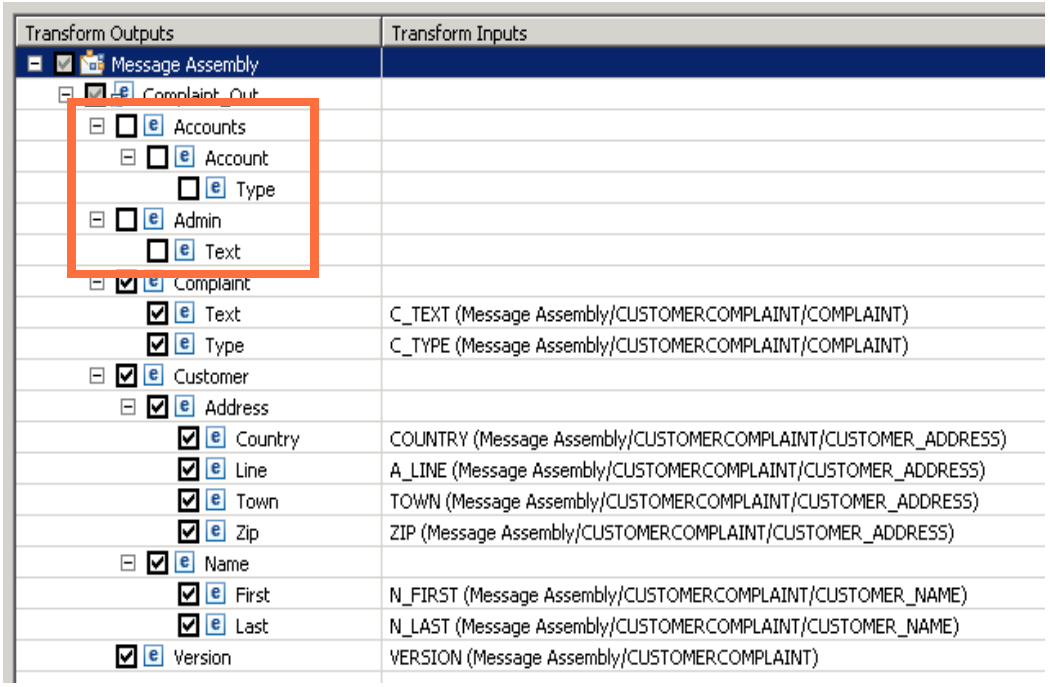
__ e. Under **Mapping Criteria**, click the **Create transforms when the names of input and outputs are more similar than** option.

__ f. Leave the default of **60%** selected.



__ g. Click **Next**. Now the preview shows that more matches were found, based on “looser” matching of the source and target fields.

__ h. Clear the check boxes for the assignments of Accounts/Account/Type and Admin/Text Data from the EMPLOYEE table in the SAMPLE database populates the Admin fields in the next part of this exercise.



__ i. Click **Finish**. The automatic mapping runs.

__ 4. Review the results of the auto map.

Hint

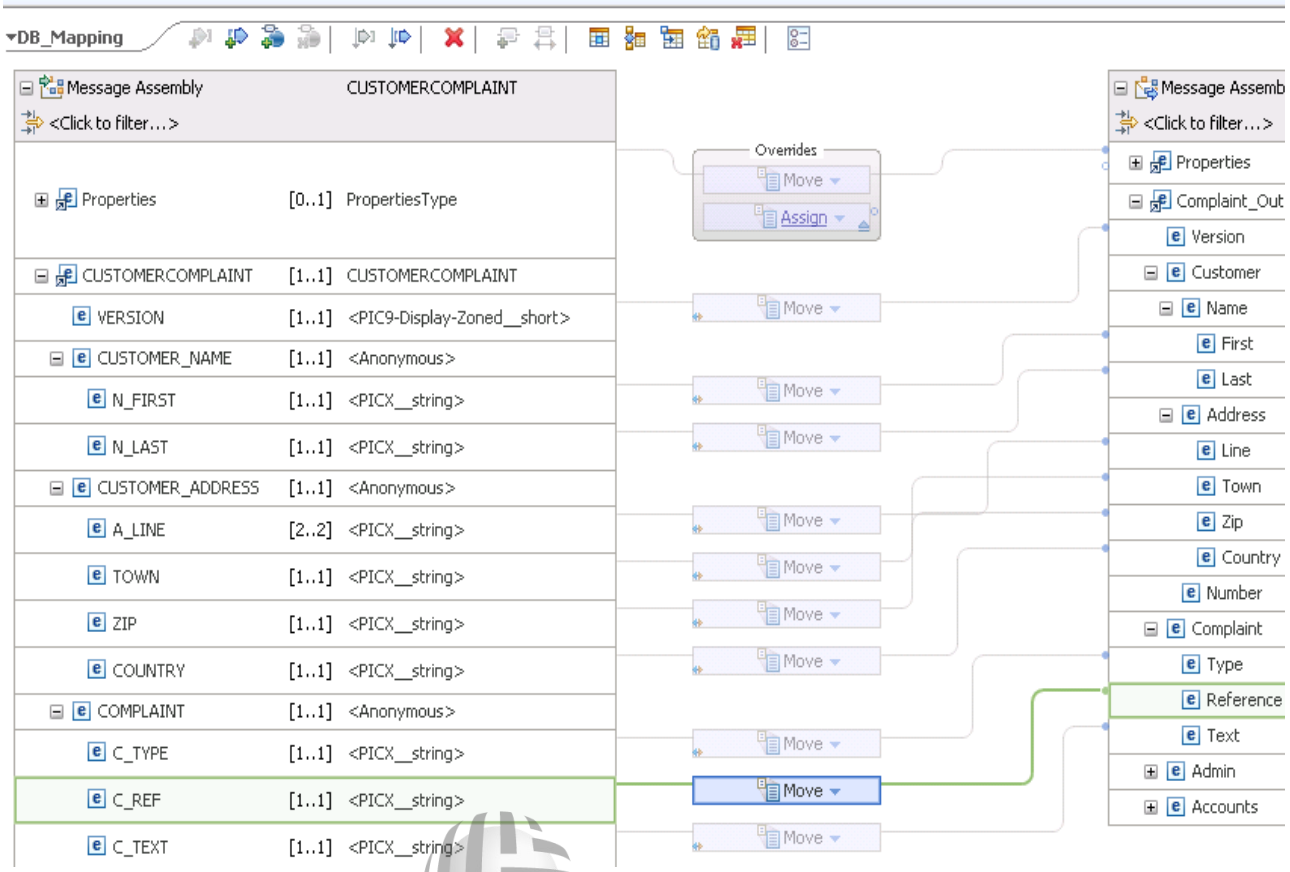
Double-click the **DB_Mapping** tab to expand the Graphical Data Mapping editor view. Double-click the tab again to return the view to the default size.

__ 5. The COMPLAINT/C_REF field in the input message did not get mapped to the Complaint/Reference field on the output message because the names were too dissimilar.

__ a. Manually map the fields by hovering the mouse cursor over COMPLAINT/C_REF in the source message. An orange wiring handle is shown.

__ b. Drag the handle to the Complaint/Reference field in the target message. The editor makes a connection and creates a **Move** transform by default.

__ c. Verify that all the fields in the source message are mapped to a field in the output message.



__ 6. Save the message map (Ctrl+S).

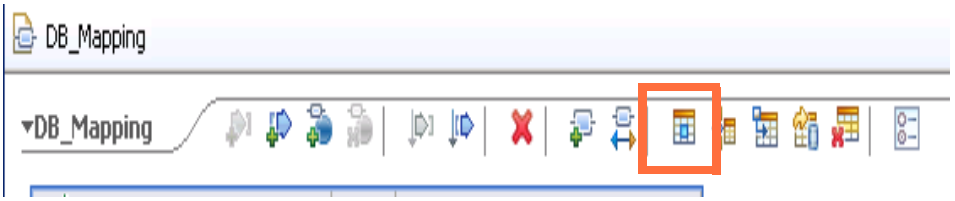
Part 5: Reference a database in a map

The **Manager** fields in the output message are selected from the EMPLOYEE table in the SAMPLE database.

EMPNO	FIRSTNME	MIDINITL	LASTNME	WORKDEPT	PHONENO	JOB
EMP012	Willie	F	Makeit	E01	4547	MANAGER
EMP025	Justin	Q	Public	E01	4524	EMPLOYEE
EMP106	Betty	M	Bacon	C01	4891	MANAGER
EMP077	Colin	J	Watson	C01	4835	EMPLOYEE
EMP301	Rebecca	L	Sunset	C01	4090	EMPLOYEE

In this part of the exercise, you set up a database retrieval to return data to populate an output field.

__ 1. In the Mapping editor toolbar, click the **Select rows from a database** icon.



The **New Database Select** window opens.

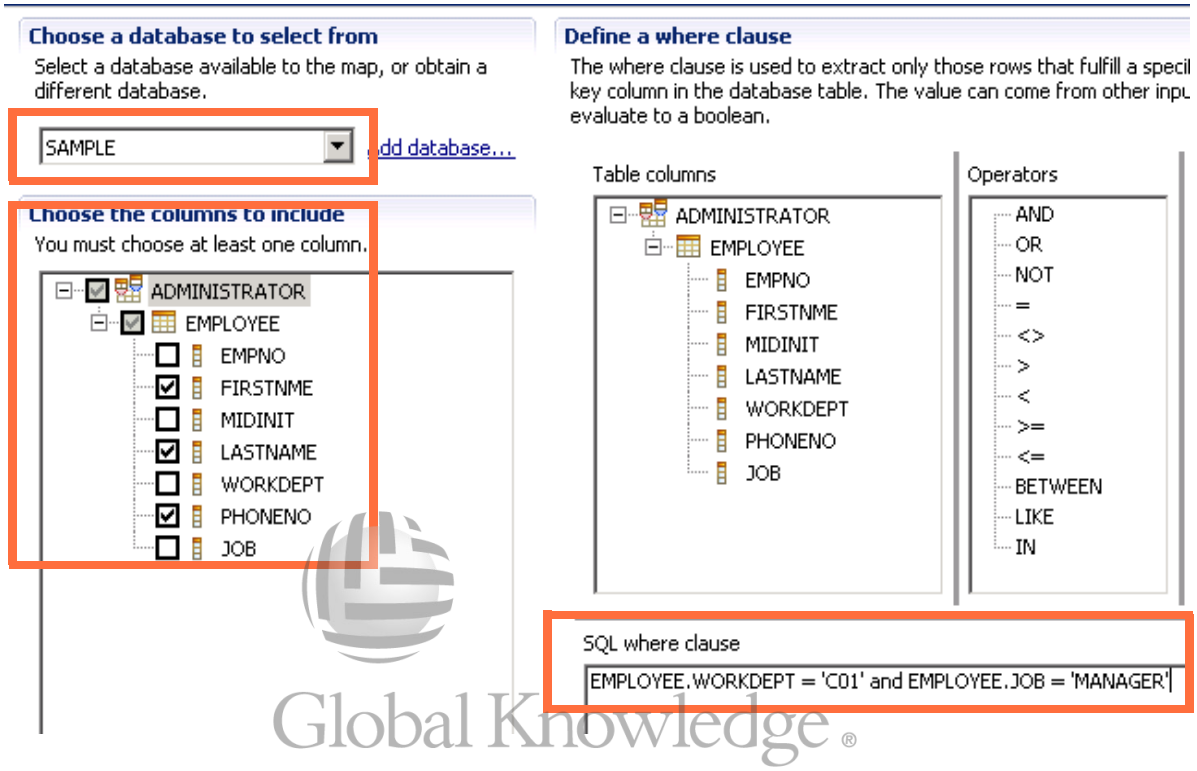
__ 2. Under **Choose a database to select from**, select **SAMPLE**.

__ 3. Under **Choose the columns to include**, expand **ADMINISTRATOR > EMPLOYEE**, and then click **FIRSTNME**, **LASTNAME**, and **PHONENO**.

__ 4. In the **SQL where clause** field, under the **Define a where clause** section, type:

```
EMPLOYEE.WORKDEPT = 'C01' AND EMPLOYEE.JOB = 'MANAGER'
```

This statement selects only rows from the database for Department C01 (which corresponds to Complaint type of "Delivery"), where the employee type is "MANAGER".



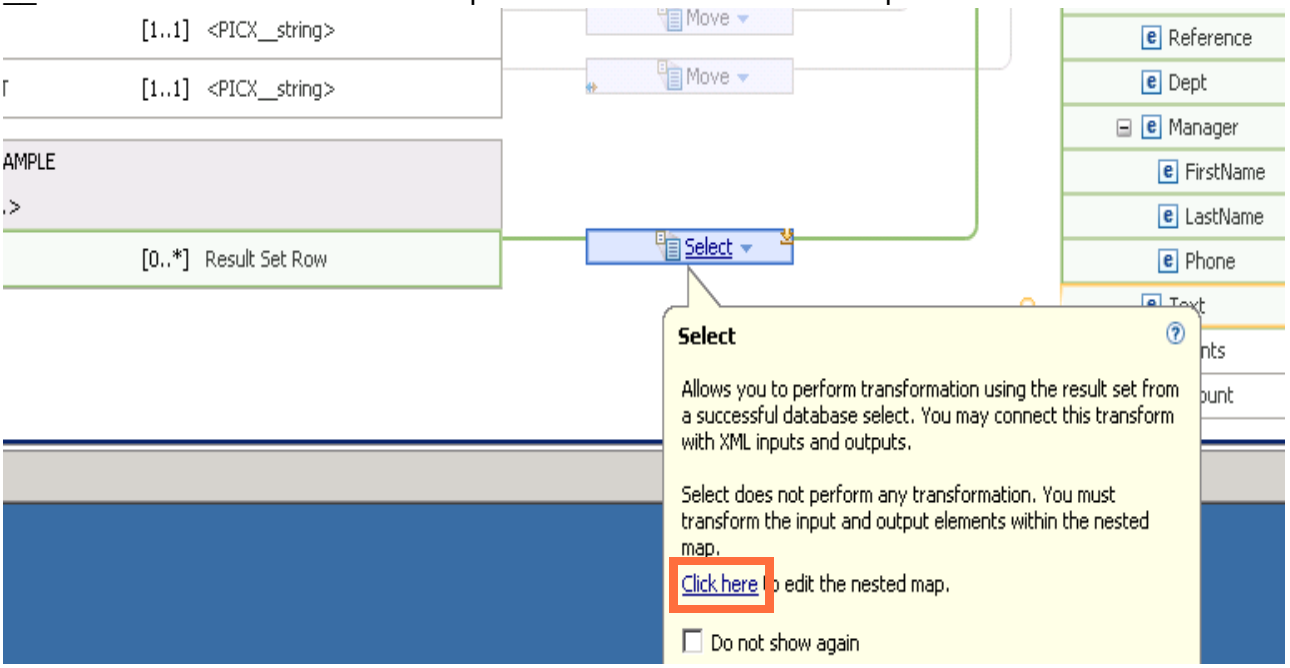
__ 5. Click **OK**. The Mapping editor pane returns.

__ 6. Scroll down under the source message. You should see that a **Select from SAMPLE** input is added to the map source.

__ 7. Map the fields that are returned from the database SELECT statement. Connect the **Select** transform to **Complaint_Out.Admin**.

You are mapping a structure to a structure, so a local map is created. A message is displayed with instructions on how to proceed with the local mapping of fields.

__ 8. Click the **Click here** link to open the editor for the nested map.



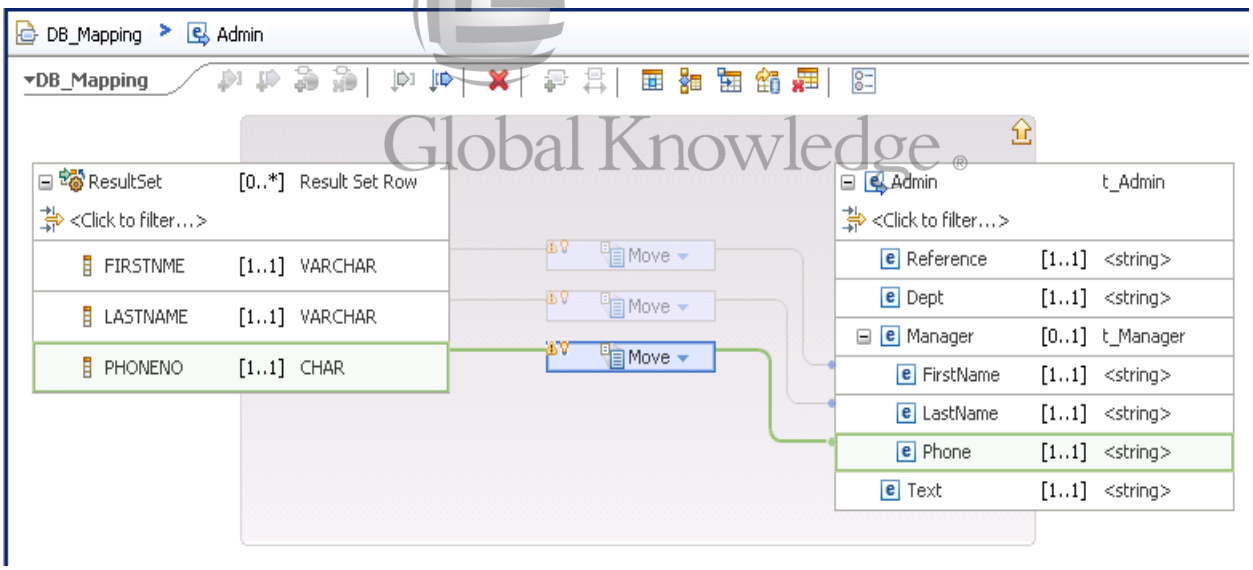
__ 9. In the output, expand **Manager** under **Admin**.

__ 10. Connect these **ResultSet** fields in the input to the **Manager** fields in the output

__ a. Connect **FIRSTNME** to **FirstName**.

__ b. Connect **LASTNAME** to **LastName**.

__ c. Connect **PHONENO** to **Phone**.



__ 11. Save the map (Ctrl+S), and close the mapping editor.

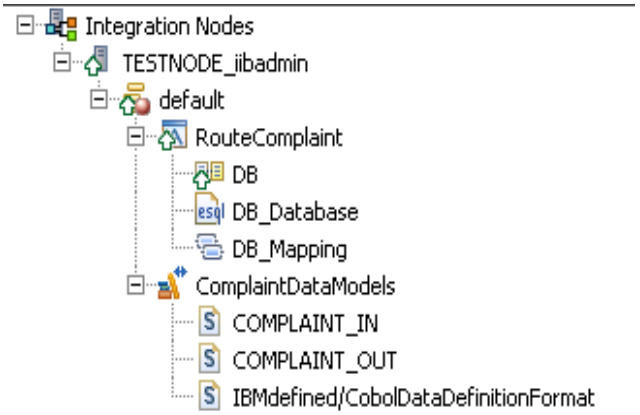
__ 12. The **Problems** view might contain cardinality and ambiguous database table reference warnings and some *Unresolvable message field reference* warning messages. These references are resolved at run time and can be ignored.

__ 13. Save the message flow (Ctrl+S).

Part 6: Test with the application

In this part of the exercise, you test the message flow and verify the contents of the output message by using the IBM Integration Toolkit Flow exerciser. You also use the DB2 command line processor to verify that the message flow added a row to the COMPLAIN table in the MSGSTORE database.

__ 1. In the Message Flow editor, click the **Start Flow exerciser** icon to build and deploy that application to the **default** integration server on the TESTNODE_iibadmin integration node.



__ 2. The message flow input node expects a COBOL message. Import the file **`Complaint_2d_cwf.txt`** from the **`C:\labfiles\Lab09-DBMap\data\`** directory.

__ a. Click the **Send a message to the flow** icon in the Flow exerciser toolbar.

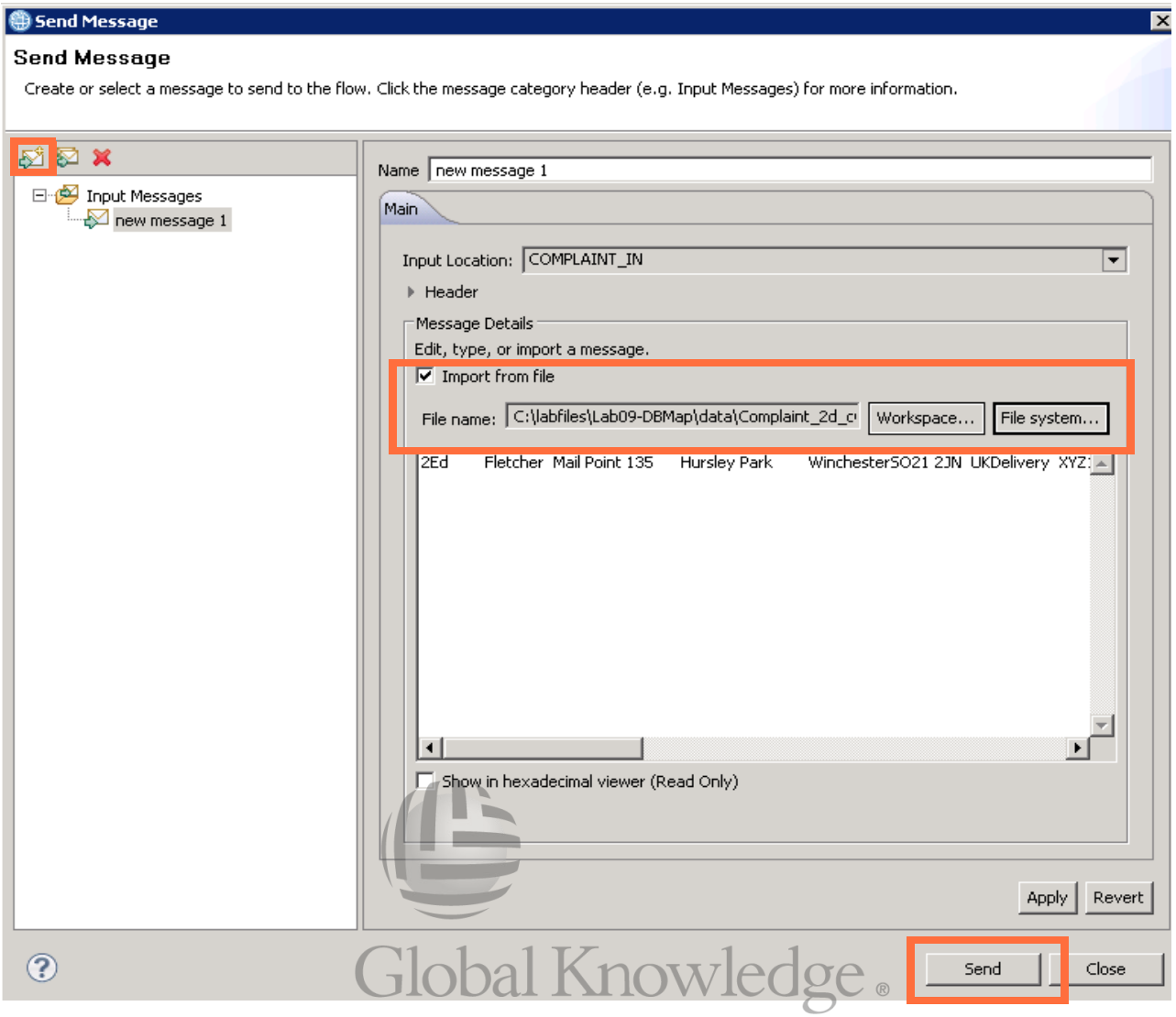
__ b. Click the **New Message** icon.

__ c. Click **Import from file**.

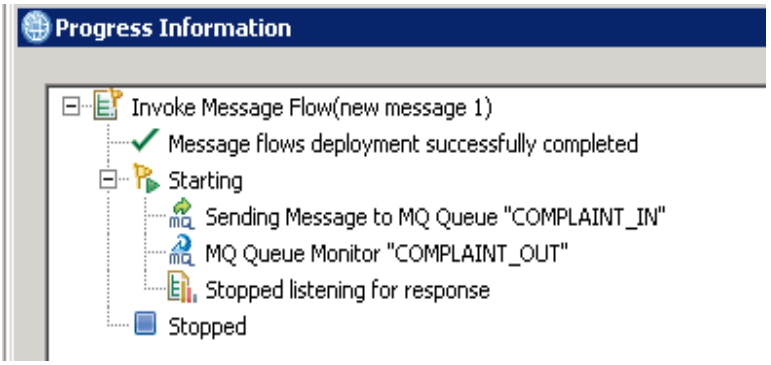
__ d. Click **File system**.

__ e. Go to the **`C:\labfiles\Lab09-DBMap\data\`** directory, click **`Complaint_2d_cwf.txt`**, and then click **Open**.

__ f. Click **Send**.



__ 3. The **Progress Information** windows shows the status of the message flow. Verify that the message flow ran successfully.

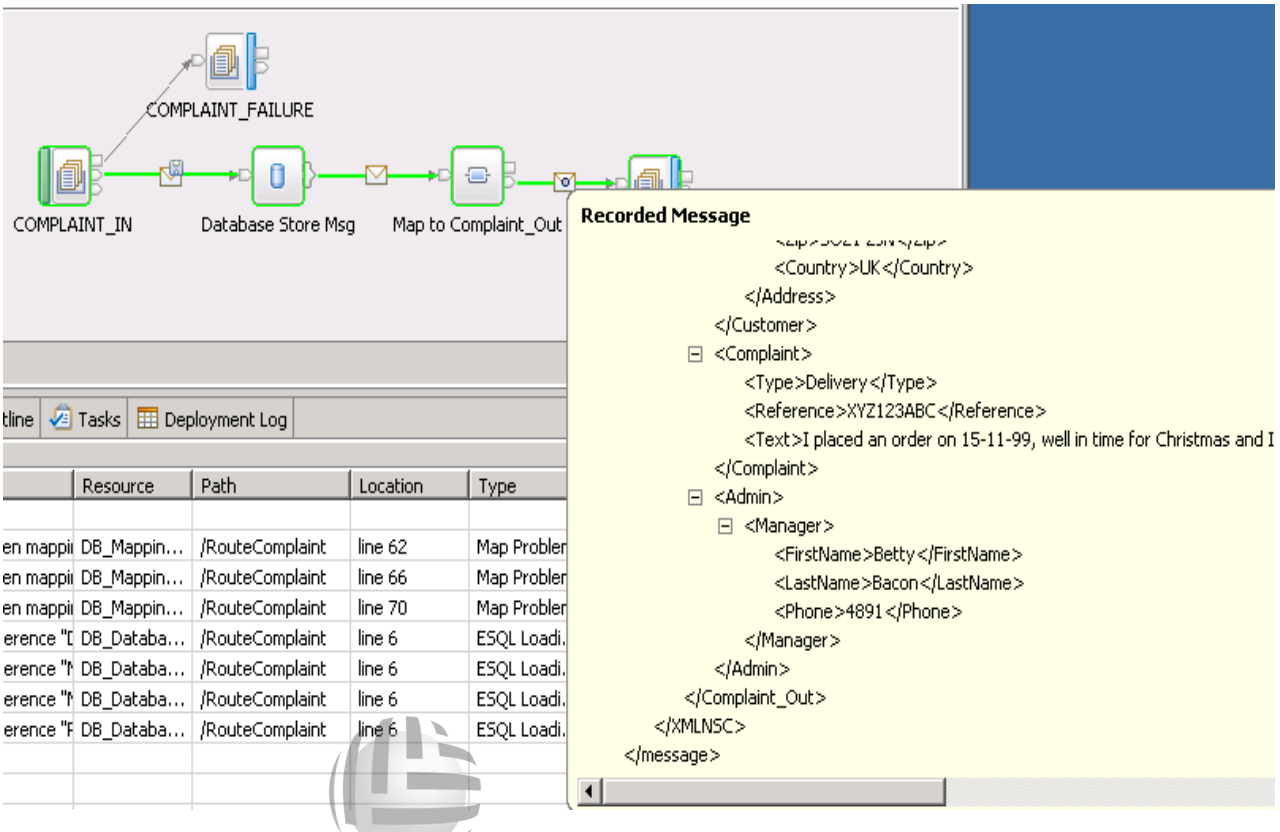


__ 4. Click **Close** on the **Progress Information** window when you see the **Stopped** event.

__ 5. The Flow exerciser path should show that the message was sent to the COMPLAINT_OUT node.

Click the message icon between the **Map to Complaint_Out** node and the **COMPLAINT_OUT** node.

Verify that the output message is correct and that it includes the **Manager** information from the EMPLOYEE table in the SAMPLE database.



__ 6. The ESQL code in the Database node stored the message in a database table. Use the DB2 command line processor to verify that an entry is created in the COMPLAIN table in the MSGSTORE database.

__ a. From the Windows **Start** menu, click **All Programs > IBM DB2 > DB2COPY1 (Default) > Command Line Processor**.

__ b. Type the following commands to connect to the MSGSTORE database and display the contents of the COMPLAIN table in the ADMINISTRATOR schema:

```
connect to MSGSTORE
select * from ADMINISTRATOR.COMPLAIN
```

__ c. Verify that the database contains at least one row and that the date in the time stamp is correct.

- The entry in the MSGID column should start with the queue manager name `IIBQM` followed by a bitstream.
- The RECEIVED column should contain a time stamp with the current date.
- The MESSAGE column should start with the `MD` characters followed by a binary bitstream.

__ d. End the connection to the MSGSTORE database. Type: `terminate`

__ e. Close the DB2 command line processor window.

__ 7. If necessary, correct the message flow, and retest. Use the problem determination tools that you learned in this course to determine the problem.

Exercise clean-up

__ 1. In the Message Flow editor, click the Flow exerciser icon to stop the Flow exerciser and return to edit mode.

__ 2. Close all the Message Flow editor tabs.

__ 3. In the **Integration Nodes** view, right-click the **default** integration server and click **Stop Recording**.

__ 4. In the **Integration Nodes** view, right-click the **default** integration server and click **Delete > All Flows and Resources**.

End of exercise

Exercise review and wrap-up

In this part of the exercise, you created a shared library and then created the data models for the COBOL Copybook that defines the input file and the XML schema that defines the output file. You then referenced the shared library in an application.

In the second part of the exercise, you created the configurable services that defined the JDBC connections to the DB2 databases. You then defined the connections to the SAMPLE and MSGSTORE databases in the IBM Integration Toolkit.

In the third part of the exercise, you completed the message flow by adding a Database node and a Mapping node. You also configured the Database node to store the message in the COMPLAIN table in the MSGSTORE database.

In the fourth part of the exercise, you used the Graphical Data Mapping editor to map the input message to the output message.

In the fifth part of the exercise, you set up a database retrieval in the Graphical Data Mapping editor to return data to populate an output field.

In the sixth part of the exercise, you tested the message flow and verified the contents of the output message by using the IBM Integration Toolkit Flow exerciser. You also used the DB2 command line processor to verify that the message flow added a row to the COMPLAIN table in the MSGSTORE database.

Having completed this exercise, you should be able to:

- Create a shared library that contains data models that describe the input and output data
- Import a COBOL Copybook to create a DFDL schema file
- Reference a shared library in a message flow application
- Discover database definitions
- Add a Database node to a message flow
- Create the logic to store a message in a database
- Use the Graphical Data Mapping editor to map message elements
- Reference an external database when mapping message elements

Exercise 10.Transforming data by using the Compute and JavaCompute nodes

What this exercise is about

In this exercise, you create a message flow application that uses ESQL and a Compute node or Java and a JavaCompute node to transform message content.

What you should be able to do

After completing this exercise, you should be able to:

- Use a Compute node or JavaCompute node in a message flow application to transform a message

Introduction

In this exercise, you create a simple messaging framework for the processing customer complaint messages. In this application, a user completes a web-based complaint form, which arrives on an IBM MQ queue as an incoming XML message, such as the example shown here.

```
<CUSTOMERCOMPLAINT>
   <VERSION>1</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <COUNTRY>UK</COUNTRY>
   </CUSTOMER_ADDRESS>
   <COMPLAINT>
      <C_TYPE>Delivery</C_TYPE>
      <C_REF>XYZ123ABC</C_REF>
      <C_TEXT>My order was delivered in time, but the package is torn.</C_TEXT>
   </COMPLAINT>
</CUSTOMERCOMPLAINT>
```


In this exercise, you create a message flow that reads the XML message from an IBM MQ queue that is named COMPLAINT_IN. An XML schema file, which you import into the application, defines the input file.

After the message is read, a Compute node or JavaCompute node in the message flow generates a unique complaint ID and determines the department that is to process the message. The complaint type (C_TYPE) in the input message determines the department.

Finally, the message flow writes the XML message to the IBM MQ queue that is named COMPLAINT_OUT. The sample output message includes the input message, followed by the complaint ID and the department.

```
<CUSTOMERCOMPLAINT>
   <VERSION>1</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <COUNTRY>UK</COUNTRY>
   </CUSTOMER_ADDRESS>
   <COMPLAINT>
      <C_TYPE>Delivery</C_TYPE>
      <C_REF>XYZ123ABC</C_REF>
      <C_TEXT>My order was delivered in time, but the package was torn.</C_TEXT>
   </COMPLAINT>
   <ADMIN>
      <REFERENCE>COMda1b71b2-f7fd-49e5-addb-ac9062f490c2</REFERENCE>
      <DEPT>B01</DEPT>
   </ADMIN>
</CUSTOMERCOMPLAINT>
```


Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit and IBM MQ V8
- The files that are required for this exercise are in the `C:\labfiles\Lab10-Compute` directory
- The COMPLAINT_IN and COMPLAINT_OUT local queues exist on an IBM MQ queue manager that is named IIBQM

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab10`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab10`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, verify that the integration node **TESTNODE_iibadmin** is started.

Start it if it is stopped.

__ 4. This exercise requires an IBM MQ queue manager.

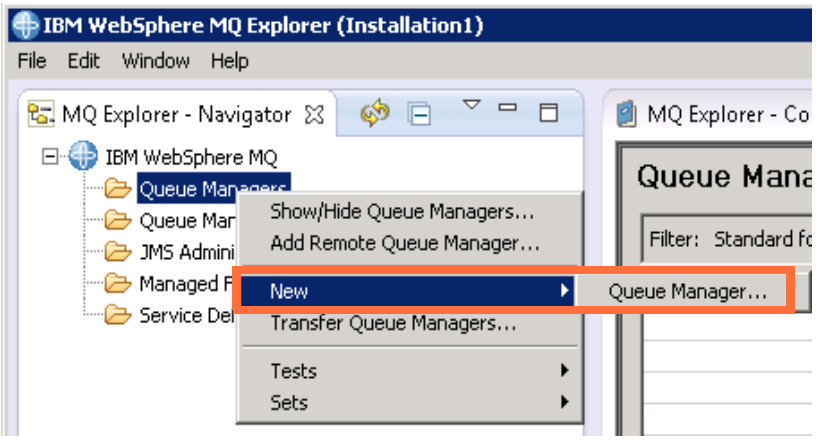
If you created the queue manager that is named **IIBQM** in a previous exercise, you can use that queue manager in this exercise. Proceed to Step 5.

If you did not create a queue manager, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ` by following these instructions.

__ a. Start IBM MQ Explorer.

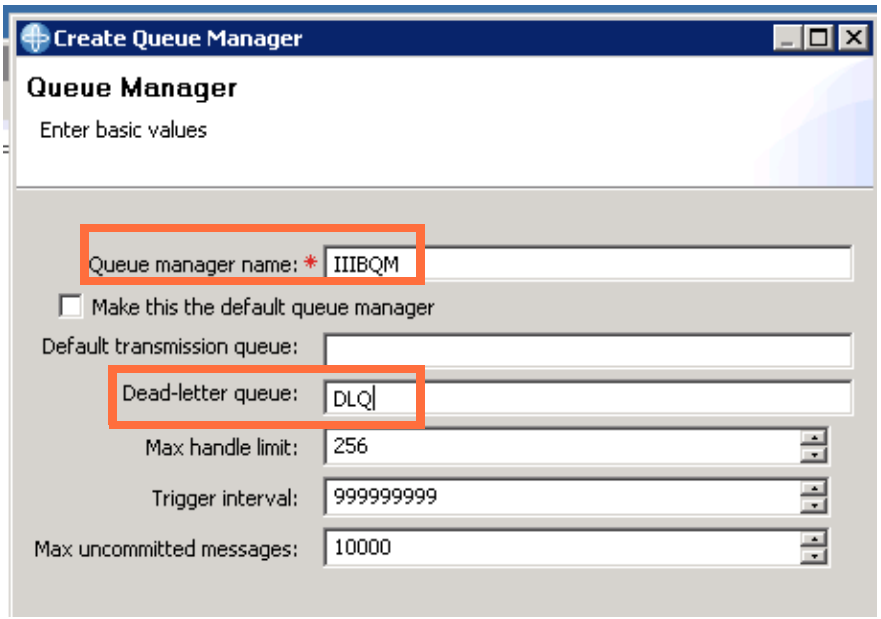
From the Windows **Start** menu, click **All Programs > IBM WebSphere MQ > WebSphere MQ (Installation 1)** or double-click the **WebSphere MQ Explorer (Installation1)** icon on the desktop.

__ b. In the **MQ Explorer Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



__ c. For the **Queue Manager** name, type: **`IIBQM`**

__ d. For the **Dead-letter queue**, type: **DLQ**



__ e. Click **Finish**.

After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

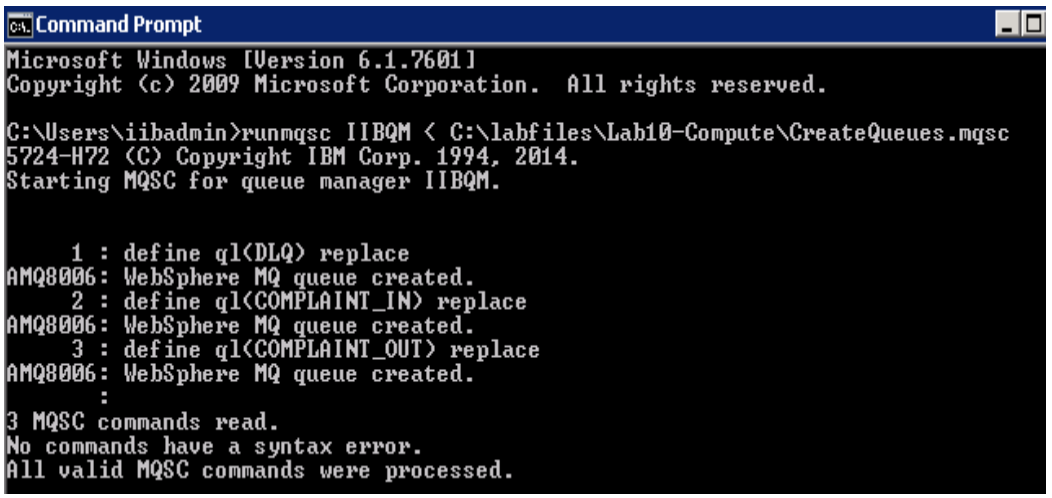
The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 5. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

In the command prompt window, type:

```
runmqsc IIBQM < C:\labfiles\Lab10-Compute\CreateQueues.mqsc
```

The command results should indicate that queues were created successfully.

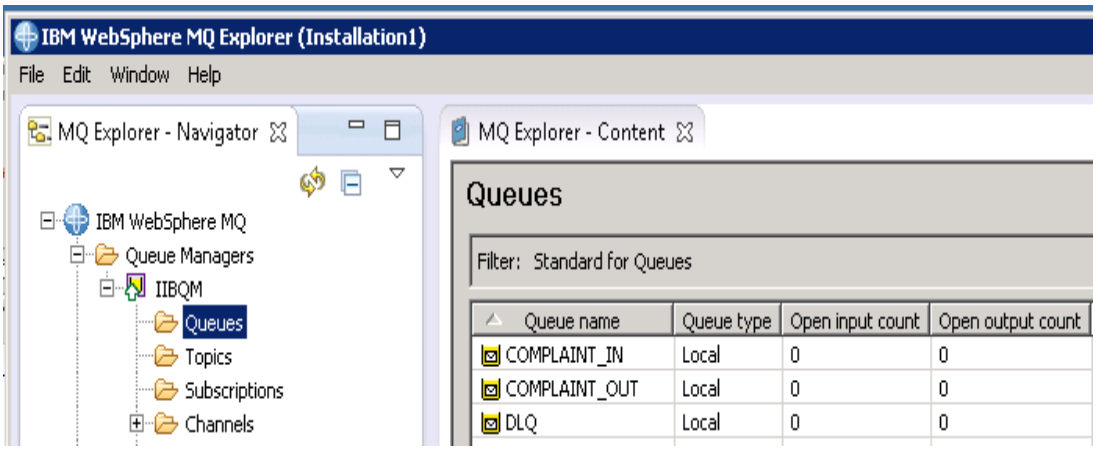


__ 6. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the IBM MQ Explorer Navigator view, expand the **IIBQM** folder.

__ b. Click **Queues** under the queue manager in the **MQ Explorer - Navigator** view to display the **Queues** view.

__ c. Verify that the following queues are listed in the **Queues** view: DLQ, COMPLAINT_IN, COMPLAINT_OUT



Note

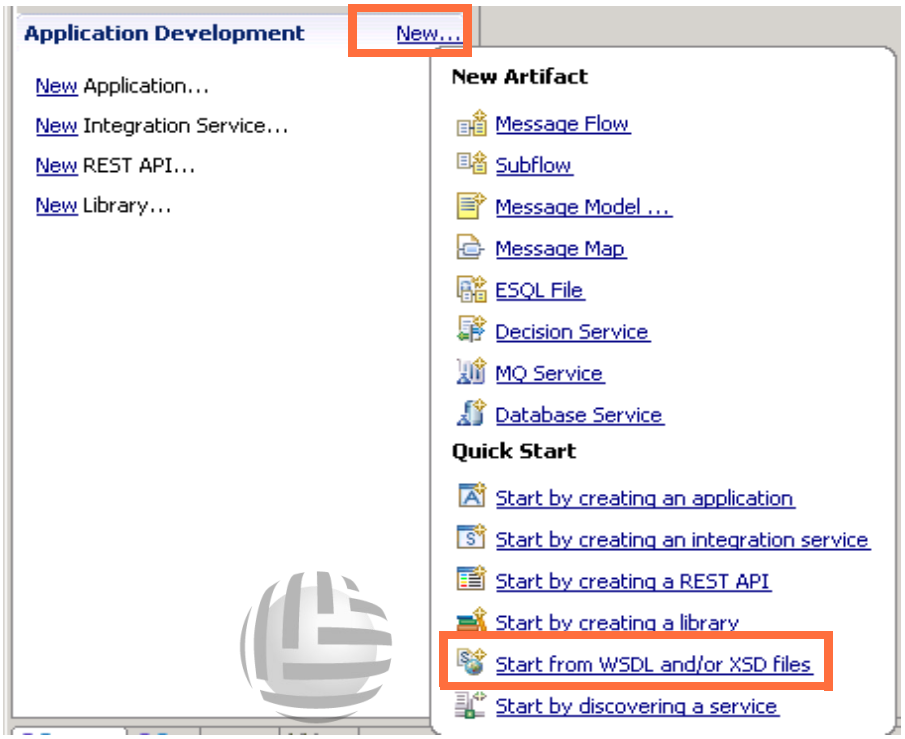
You might have other queues for this queue manager from previous exercises. You do not need to delete the unused queues.

Part 1: Create the message flow

In this part of the exercise, you import an XML schema definition (XSD) file that is named **`Complaint.xsd`** that describes the input file. You also add processing nodes to the message flow.

__ 1. Create a message flow application that is named **RouteComplaint** by starting from an XSD file.

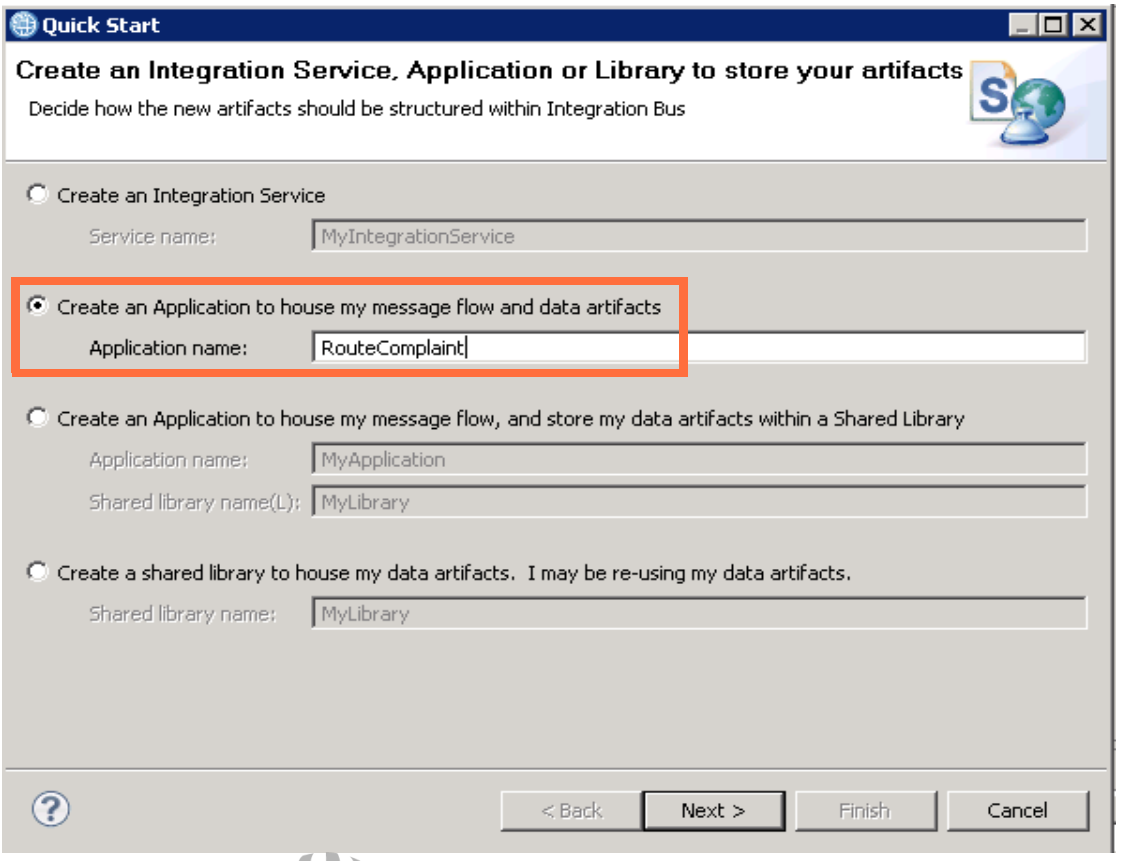
__ a. In the IBM Integration Toolkit Application Development view, click **New > Start from WSDL and/or XSD files**.



The **Quick Start** window opens.

__ b. Click **Create an Application to house my message flow and data artifacts**.

__ c. For the **Application name**, type: **`RouteComplaint`**



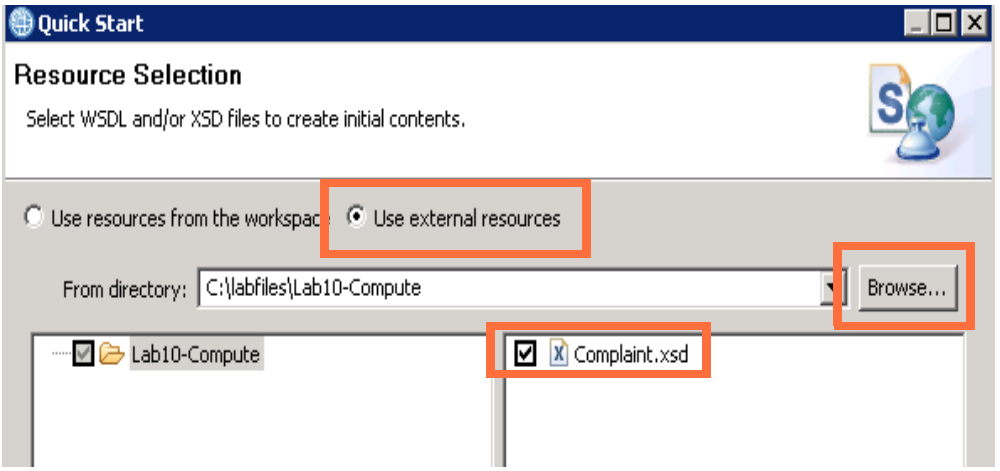
__ d. Click **Next**. The **Resource Selection** window opens.

__ e. Click **Use external resources**.

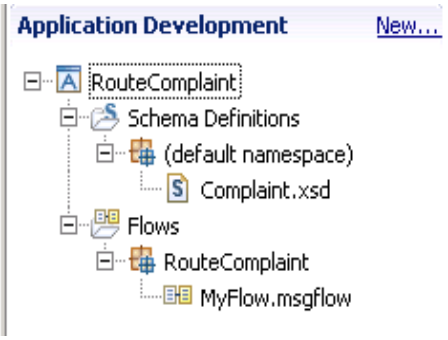
__ f. Click **Browse** to the right of **From directory**.

__ g. Browse to the **`C:\labfiles\Lab10-Compute`** directory and then click **OK**.

__ h. Select the **Complaint.xsd** check box.



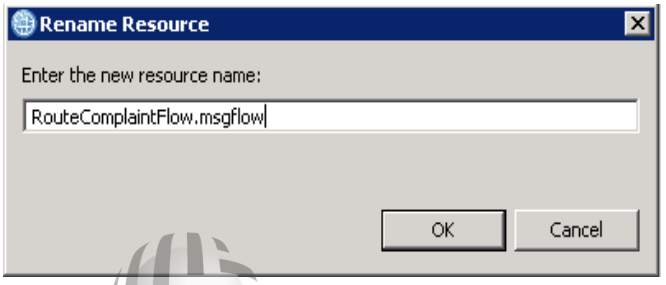
__ i. Click **Finish**. The schema definition is imported and the application is created in the **Application Development** view.



__ 2. The **RouteComplaint** application contains a message flow file that is named **MyFlow.msgflow**. Rename the message flow file to **RouteComplaintFlow.msgflow**.

__ a. Right-click the **MyFlow.msgflow** file in the **Application Development** view and then click **Rename**.

__ b. For the new message flow name, type: **`RouteComplaintFlow.msgflow`**



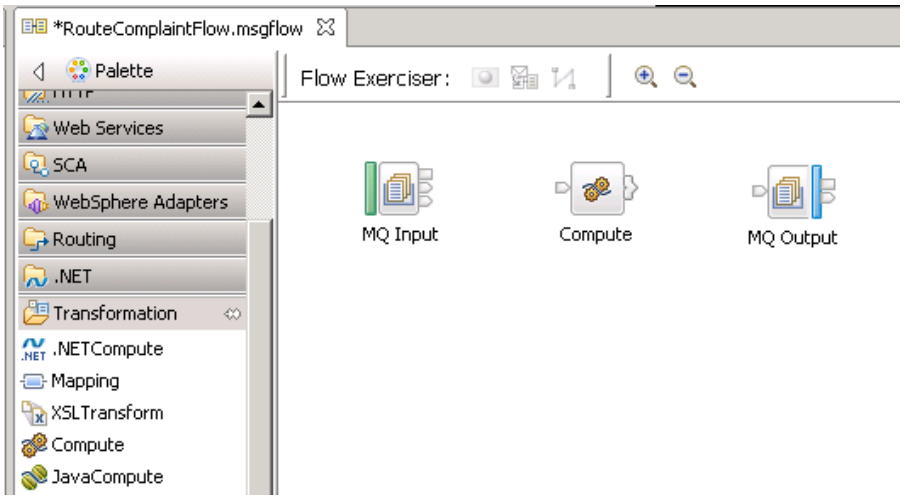
__ c. Click **OK**.

__ 3. Open the message flow file in the Message Flow editor.

__ 4. Add the following nodes to the Message Flow editor canvas.

- From the **WebSphere MQ** drawer, add one MQInput node
- From the **Transformation** drawer, add one Compute node if you want to use ESQL in this exercise, or add one JavaCompute node if you want to use Java in this exercise

- From the **WebSphere MQ** drawer, add one MQOutput node



Note

Select either the Compute node or the JavaCompute node. Do not select both. If time is available, you can implement the other node after you complete the exercise.

__ 5. Rename the nodes to more descriptive names.

To rename a node, click it, and then switch to the **Properties** tab. Change the **Node name** field on the **Description** tab.

Alternatively, you can click the name in the Message Flow editor canvas and type a new name.

__ a. Rename the MQInput node to: `COMPLAINT_IN`

__ b. Rename the MQOutput node to: `COMPLAINT_OUT`

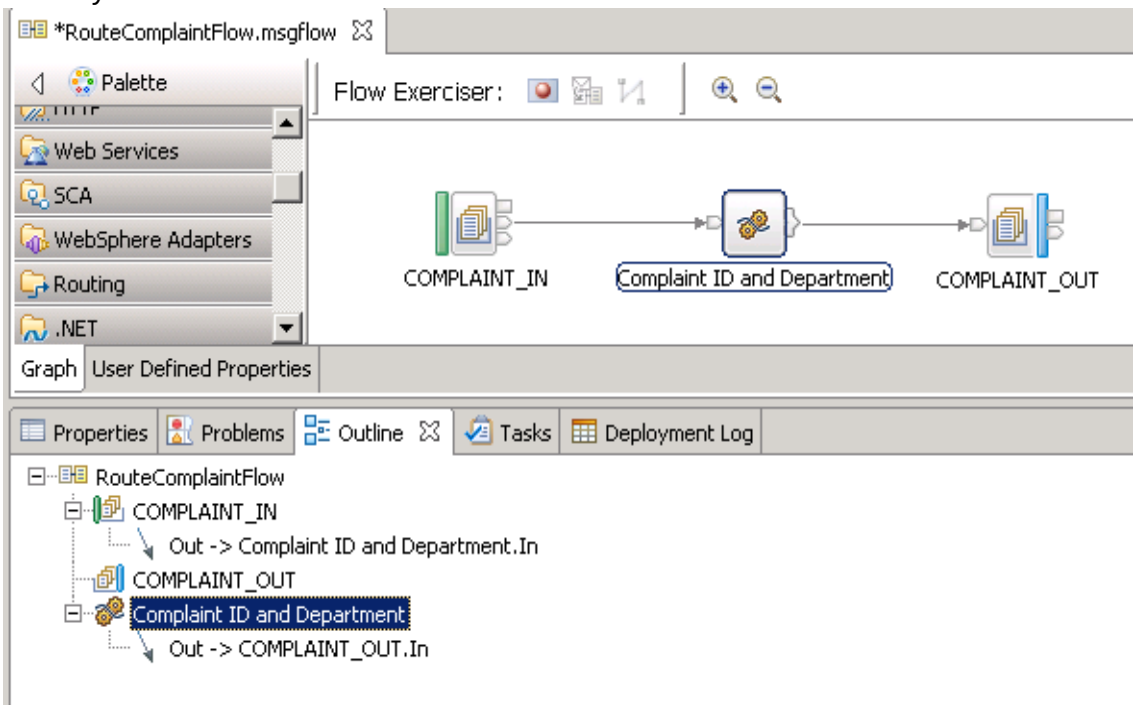
__ c. Rename the Compute or JavaCompute node to: `Complaint ID and Department`

__ 6. Connect the nodes.

__ a. Wire the COMPLAINT_IN node **Out** terminal to the Complaint ID and Department node **In** terminal.

__ b. Wire the Complaint ID and Department **Out** terminal to the COMPLAINT_OUT node **In** terminal.

__ 7. Verify the terminal connections in the **Outline** view.



__ 8. Configure the properties for MQInput node that is named COMPLAINT_IN.

__ a. On the **Basic** tab, set the **Queue name** property to: **`COMPLAINT_IN`**

__ b. On the **MQ Connections** tab, set the **Destination queue manager name** property to: **`IIBQM`**

__ c. On the **Input Message Parsing** tab, set the **Message Domain** property to: **`XMLNSC`**

__ 9. Configure the properties for MQOutput node that is named COMPLAINT_OUT.

__ a. On the **Basic** tab, set the **Queue name** property to: **`COMPLAINT_OUT`**

__ b. On the **MQ Connections** tab, set the **Destination queue manager name** property to: **`IIBQM`**

__ 10. Save the message flow file. You should see that the compute node contains an error.

If you look in the **Problems** view, you see that this error is caused by missing code for the compute node. In the next part of the exercise, you create the code for the compute node.

Part 2: Configure the Compute or JavaCompute node

In this part of the exercise, you configure the **Complaint ID and Department** node. The code that you provide for the node adds the following logic:

- Copy the input message to the output message.
- Create the `OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE` element in the output message.

  The value for this element is computed by concatenating the string `COM` with a unique identifier that a function generates. This function returns universally unique identifiers

(UUIDs) as CHARACTER values. It acts as a type of random generator of character strings.

- Create the `OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT` element in the output message. The value for this element is based on the content of `InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE`, as follows:
  - If the complaint is about an order (`C_TYPE = "Order"`), then the department `B01` is assigned to handle the complaint.
  - If the complaint is about a delivery (`C_TYPE = "Delivery"`), then the department `C01` is assigned to handle the complaint.
  - All other complaint types are assigned to department `E01`.

Important

The code is provided for you in a text file in the `C:\labfiles\Lab10-Compute` directory.

You implement the transformation code by using a Compute node or a JavaCompute node. Complete the steps based on the node that you added to your flow.

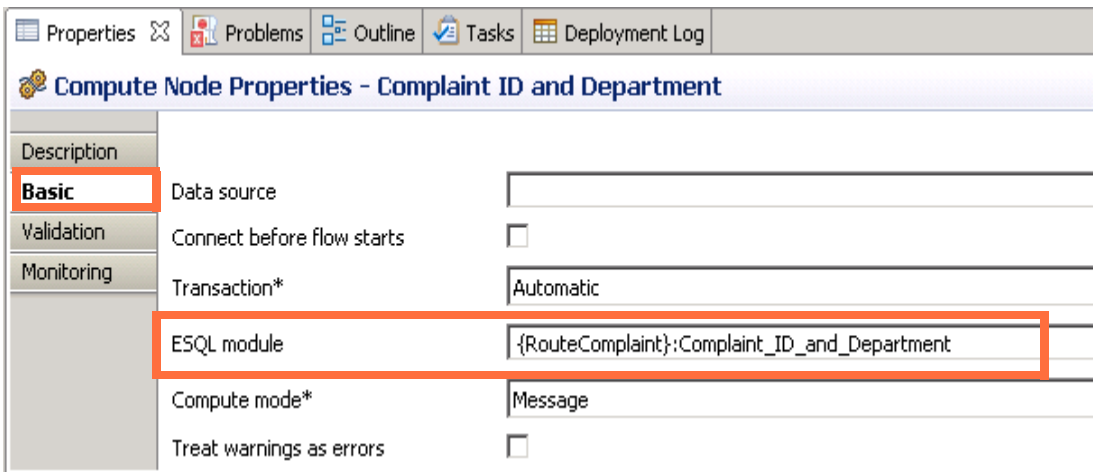
If time remains after you implement and test the message flow with one of the compute nodes, you can remove the node and substitute it with the other compute node.

Option 1: Write a transformation by using a Compute node and ESQL

If you are using a Compute node in your message flow, the next step is to create the ESQL to transform the message.

__ 1. On the Compute node that is named Complaint ID and Department, set the **ESQL module** property on the **Basic** tab.

For the **ESQL module** property, type: **`{RouteComplaint}:Complaint_ID_and_Department`**



__ 2. Double-click the Complaint ID and Department Compute node in the Message Flow editor to open the ESQL code template.

In the code template, the first statement `BROKER SCHEMA RouteComplaint` identifies the application. It is taken from the first part of the ESQL module name that you entered in the node property for the **ESQL module**.

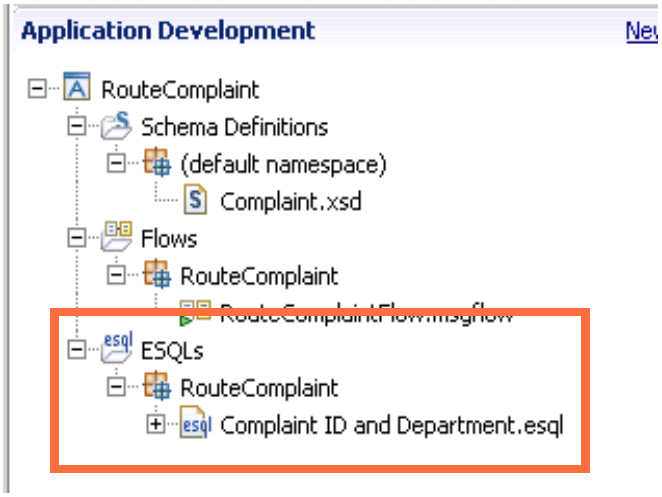
The second statement, `CREATE COMPUTE MODULE Complaint_ID_and_Department` is the Compute module name and must match the name that you entered in the ESQL module property in Step 1.

__ 3. Copy the ESQL from the `ESQL_Cut&Paste.txt` file in the `C:\labfiles\Lab10-Compute` directory and replace the existing code.

The contents of the ESQL module should match the code that is shown here.

```
BROKER SCHEMA RouteComplaint

CREATE COMPUTE MODULE Complaint_ID_and_Department
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN

      SET OutputRoot = InputRoot;

/* create the complaint reference ID by using the UUIDASCHAR function */
SET OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE = 'COM' ||
UUIDASCHAR;

/* assign the department to handle the complaint based on complaint type */
      SET OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT =
        CASE InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE
           WHEN 'Order' THEN 'B01'
           WHEN 'Delivery' THEN 'C01'
           ELSE 'E01'
        END;  /* case */

   RETURN TRUE;  /* propagate message to Out terminal */

    END;  /* create function */
END MODULE;
```

__ 4. Save the ESQL file. The source for the ESQL module is saved with the application under the **ESQLs** folder.



__ 5. Close the ESQL editor.

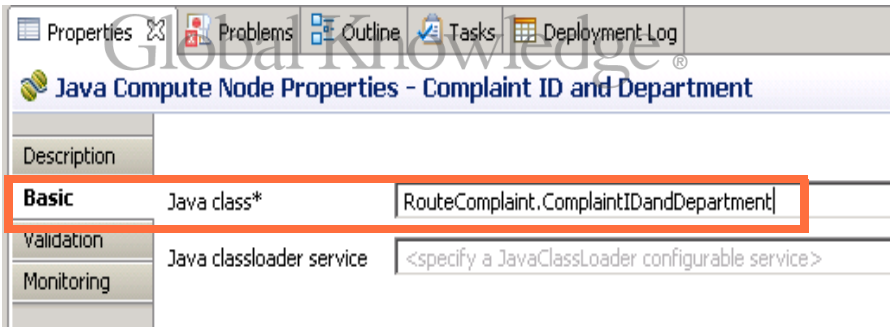
__ 6. Save the message flow.

__ 7. Verify that the **Problems** view is empty.

Option 2: Write a transformation by using a JavaCompute node and Java

If you are using a JavaCompute node in your message flow, the next step is to create the Java code to transform the message.

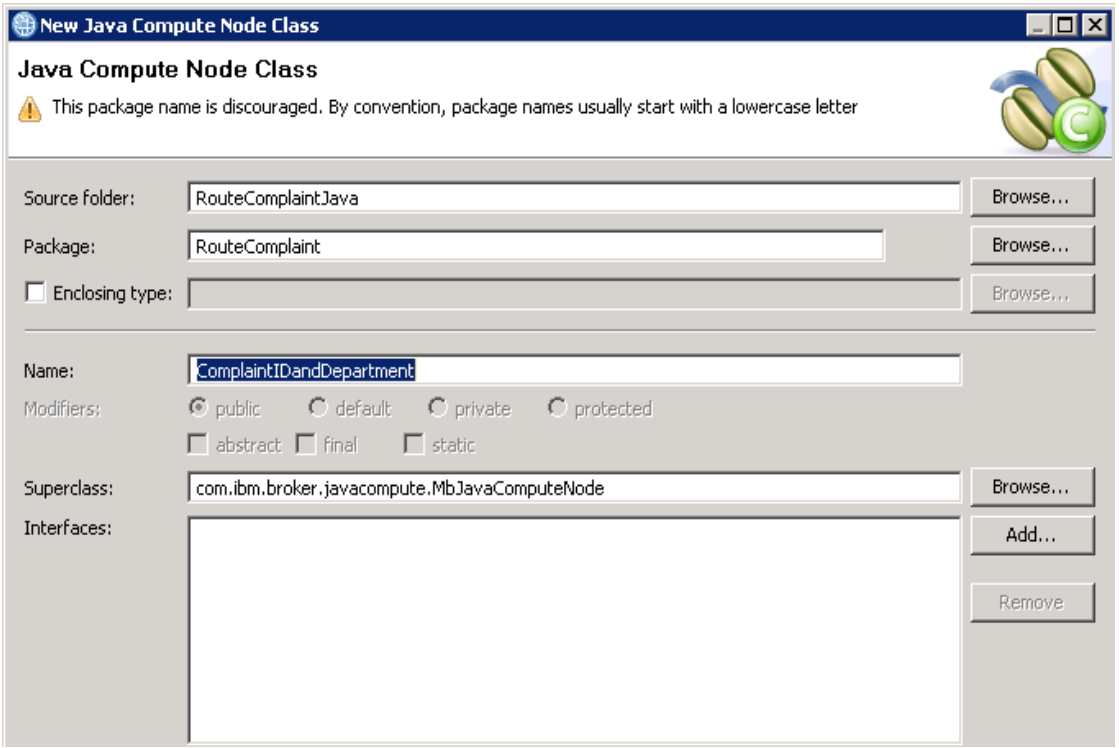
__ 1. In the JavaCompute node that is named Complaint ID and Department, set the **Java class** property to `RouteComplaint.ComplaintIDandDepartment` on the **Basic** tab.



__ 2. Create a JavaCompute node project where you store the classes that are used in the JavaCompute nodes.

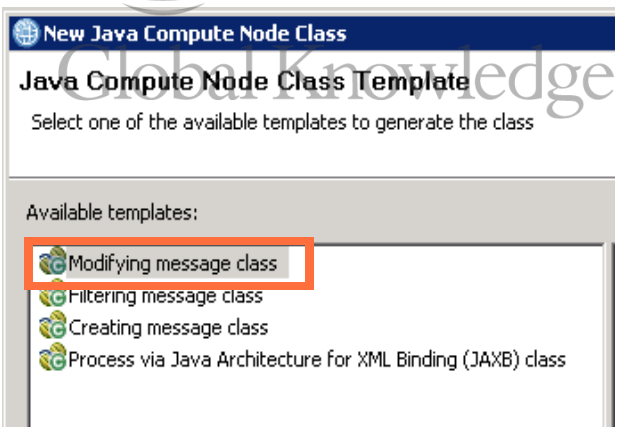
__ a. In the Message Flow editor, double-click the Complaint ID and Department node. The **New Java Compute Node Class** wizard is displayed.

By default, the **Package** is named **RouteComplaint** and the class is named **ComplaintIDandDepartment** (the names that you specified for the **Java class** property for the node).



__ b. Click **Next**. The Java Compute Node Class Template window opens.

__ 3. In this exercise you modify the message in this JavaCompute node, select **Modifying message class**, and then click **Finish**.



The Java editor opens with the code template for modifying a message.

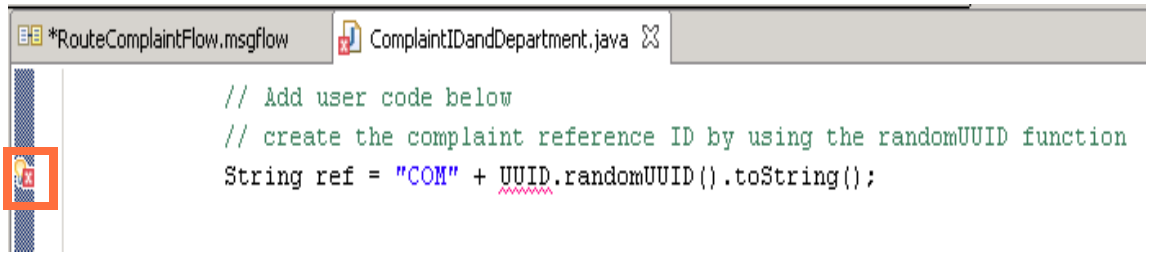
__ 4. Copy the Java code from the `Java_Cut&Paste.txt` file in the `C:\labfiles\Lab10-Compute` directory.

__ 5. Paste the Java code to the Java between the `// Add user code below` and `// End of user code` lines.

The code that you insert into the file is shown here:

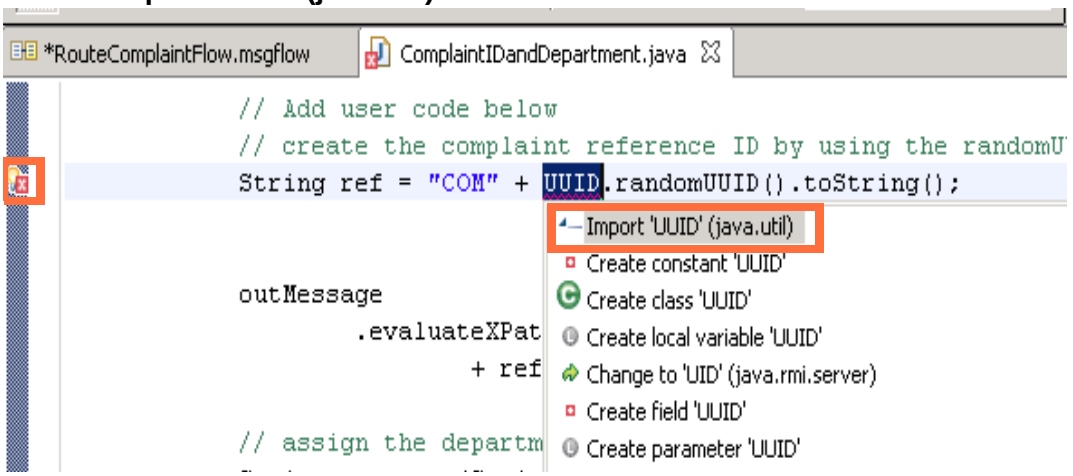
```
// Add user code below
   String ref = "COM" + UUID.randomUUID().toString();

          outMessage
                .evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?REFERENCE
[set-value('"+ ref + "')]");

          String ctype = (String) outMessage

.evaluateXPath("string(CUSTOMERCOMPLAINT/COMPLAINT/C_TYPE)");

          if (ctype.equalsIgnoreCase("Order"))
             outMessage

.evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?DEPT[set-value('B01')]");

          else if (ctype.equalsIgnoreCase("Delivery"))
             outMessage

.evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?DEPT[set-value('C01')]");

          else
             outMessage

.evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?DEPT[set-value('E01')]");

// End of user code
```

__ 6. When you paste the Java code into the editor, a red error decorator is shown in the left margin of the editor. If you hover the cursor over the decorator, you see the message *UUID cannot be resolved*.



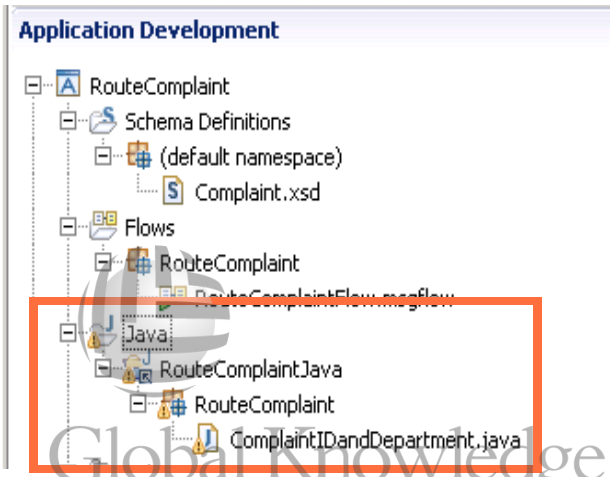
__ a. To fix this error, click the red error decorator in the margin. A quick fix menu opens.

__ b. Click **Import 'UUID' (java.util)**.



The error decorator disappears.

__ 7. Save the changes in the Java editor. The source for the Java file is saved with the application under the **Java** folder.



__ 8. Close the Java editor.

__ 9. Save the message flow.

__ 10. Verify that there are no errors in the **Problems** view.

Part 3: Test the message flow

In this part of the exercise, you use the IBM Integration Toolkit Flow exerciser to test the message flow.

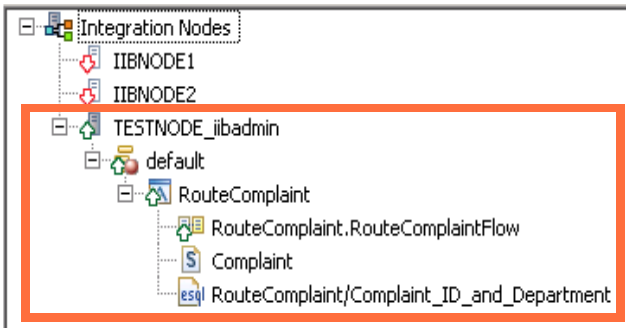
The `C:\labfiles\Lab10-Compute\data` directory contains three XML files.

- When you test the flow with `Complaint_Order.xml`, the output message should contain the string `<DEPT>B01</DEPT>` because the complaint is about an order (`C_TYPE = "Order"`).
- When you test the flow with file `Complaint_Delivery.xml`, the output message should contain the string `<DEPT>C01</DEPT>` because the complaint is about a delivery (`C_TYPE = "Delivery"`).

- When you test the flow with `Complaint_Misc.xml`, the output message should contain the string `<DEPT>E01</DEPT>` because the complaint is not about an order or a delivery.

__ 1. In the IBM Integration Toolkit Message Flow editor, start the Flow exerciser to create the BAR file, deploy the application to the **default** integration server on the TESTNODE_iibadmin integration node, and start recording.

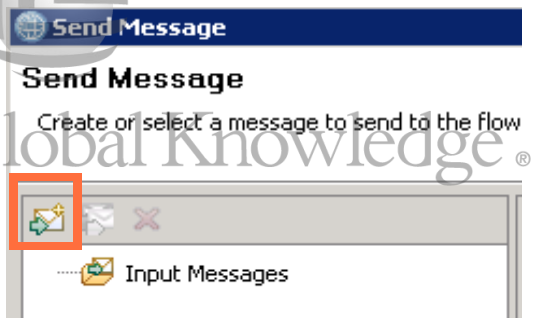
__ 2. Using the **Deployment Log** view and the **Integration Nodes** view, verify that the message flow application deployed successfully.



__ 3. Test the message flow with the `Complaint_Order.xml` file by importing the file from the file system and sending the message with the Flow exerciser.

__ a. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

__ b. In the **Send Message** window, click the **New Message** icon.

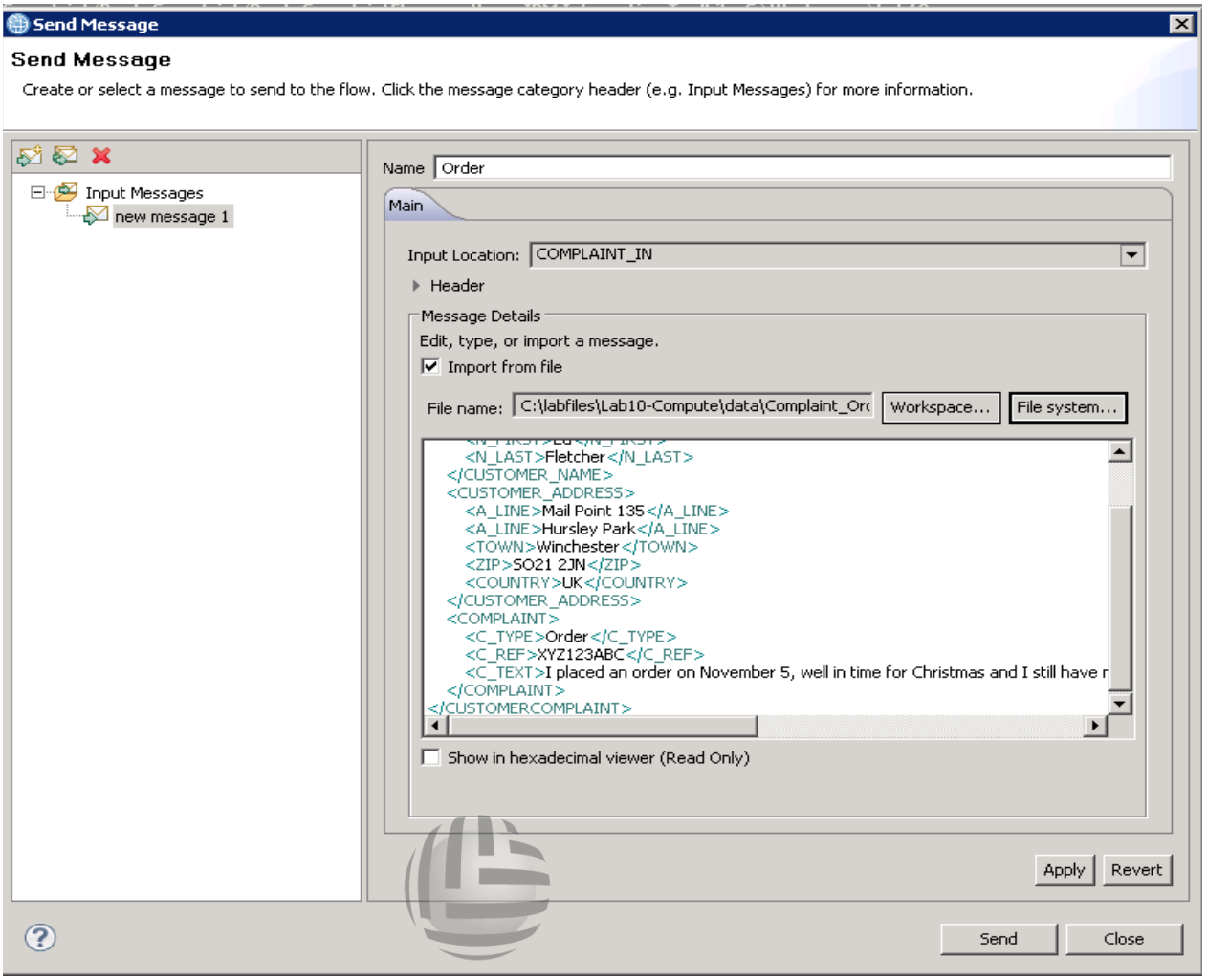


__ c. For the **Name**, type: **`Order`**

__ d. Click **Import from file** and then click **File system**.

__ e. Go to the `C:\labfiles\Lab10-Compute\data` directory, click the `Complaint_Order.xml` file, and then click **Open**.

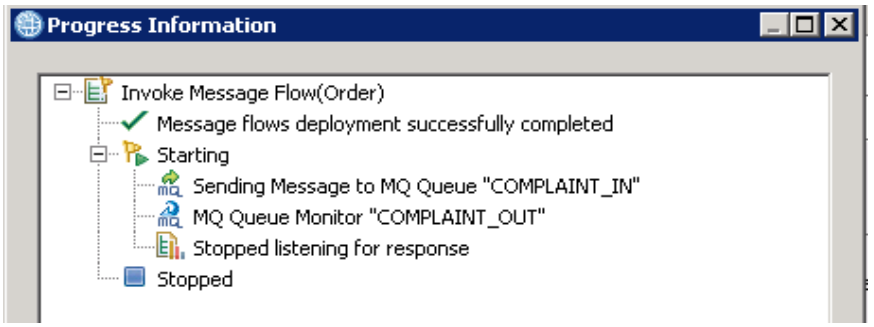
The file is imported into the Flow exerciser.



__ f. Click **Send**.

__ g. The **Progress Information** window shows that the message is sent to input queue COMPLAINT_IN. It also shows the destination, which is the COMPLAINT_OUT queue.

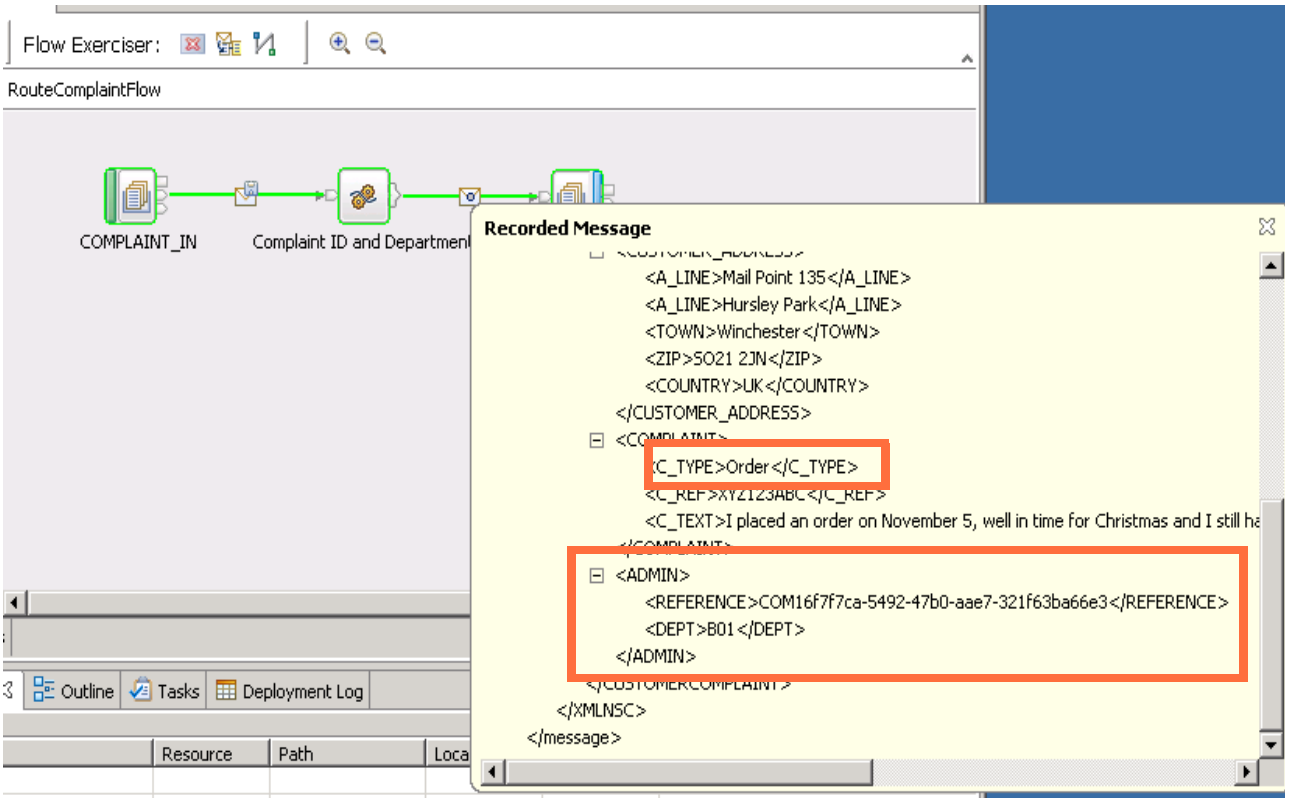
When the **Progress Information** window displays **Stopped**, the test is complete. Click **Close**.



__ h. The message path is highlighted on the message flow.

Click the message icon after the Complaint ID and Department node to display the output message.

Verify that the <ADMIN> elements are appended to the message and that the value of <DEPT> is correct.



__ i. Close the message window.

Information

If you view the queues by using IBM MQ Explorer, you see that the **Current queue depth** for the output queue is empty. The Flow exerciser does not put the message to the output queue. It shows the message path only. If you want to view the messages on the queues, you can use the IBM MQ Rfhutil SupportPac IH03 to test the flow by putting a message to the COMPLAINT_IN queue.

__ 4. Following the procedure in Step 3, test the message flow with `Complaint_Delivery.xml` file.

When you test the flow with file `Complaint_Delivery.xml`, the output message should contain the string `<DEPT>C01</DEPT>` because the complaint is about a delivery (`C_TYPE = "Delivery"`).

__ 5. Following the procedure in Step 3, test the flow with the `Complaint_Misc.xml` file.

When you test the flow with `Complaint_Misc.xml`, the output message should contain the string `<DEPT>E01</DEPT.>` because the complaint is not about an order or a delivery.

__ 6. Click the Flow exerciser icon to return the flow to edit mode.

Information

In a classroom environment, the time that is allotted for this exercise assumes that you complete the message flow by using either the Compute node with ESQL or the JavaCompute with Java. If time allows or you are working in a self-paced environment, you can add the other type of Compute node to the message flow by following these steps:

1. Add the other type of Compute node to the message flow and connect it the **Out** terminal of the COMPLAINT_IN MQInput node.
2. Add the code to the Compute or JavaCompute node by following the instructions in Part 2 of the exercise.
3. Using IBM MQ Explorer, create a local queue that is named COMPLAINT_OUT2.
4. Add an MQOutput node to the message flow that references the COMPLAINT_OUT2 queue.
5. On the **Basic** properties tab of the new MQOutput node, set the **Queue name** property to: **`COMPLAINT_OUT2`**
6. On the **MQ Connections** properties tab of the new MQOutput node, set the **Destination queue manager name** property to: **`IIBQM`**
7. Connect the new Compute or JavaCompute node to the new MQOutput node.
8. Save the message flow.
9. Deploy and test the message flow by following the instructions in Part 3 of this exercise and use the Flow exerciser to verify the output message.

Exercise clean-up

__ 1. Close the message flow in the Message Flow editor.

__ 2. In the IBM Integration Toolkit **Integration Nodes** view, right-click the **default** integration server on **TESTNODE_iibadmin** and then click **Stop recording**.

__ 3. In the IBM Integration Toolkit **Integration Nodes** view, delete all flows and resources from **default** integration server on **TESTNODE_iibadmin**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you imported an XML schema definition that describes the input file. You also added processing nodes to the message flow. In the second part of this exercise, you configured the Complaint ID and Department node by using a Compute node and ESQL or a JavaCompute node and Java. In the third part of this exercise, you used the IBM Integration Toolkit Flow exerciser to test the message flow. Having completed this exercise, you should be able to use a Compute node or JavaCompute node in a message flow application to transform a message

Exercise 11. Creating a runtime-aware message flow

What this exercise is about

Message flows can be made even more powerful and flexible if they can interact with the environment in which they are operating. In this exercise, you modify an existing message flow for its runtime environment.

What you should be able to do

After completing this exercise, you should be able to:

- Add a user-defined property to a message flow
- Promote subflow properties to the main flow
- Define custom keywords
- Set configurable properties in the BAR file
- View BAR file properties at run time

Introduction

Message flows can be made even more powerful and flexible if they can interact with the environment in which they are operating. A message flow that can determine whether it is running in a production environment might process differently than the same flow that is running in a development or test environment. Similarly, because of the reusability characteristics of a subflow, you might have multiple versions of it running in a particular environment. In that situation, it is useful to know what version is running. You can use user-defined variables to provide embedded version information. In the first part of this exercise, you add user-defined properties to a message flow.

You can use promoted properties to change how a message flow operates at the BAR file level, rather than at the message flow level. You can promote a message flow node property to the message flow level to simplify the maintenance of the message flow and its nodes. You can also use promoted properties to provide common values for multiple message flow nodes in the flow by converging promoted properties. In the second part of this exercise, you promote subflow properties to the main flow.

You can define custom keywords to provide more information about a message flow, such as the message flow author and a version. The IBM Integration Toolkit interprets custom keywords as information to display in the

Properties view. In the third part of this exercise, you create keywords for the subflow that are displayed in the deployed message flow.

In the fourth part of this exercise, you build and deploy a BAR file manually so that you can examine the promoted properties and keywords.

Requirements

- A lab environment with the IBM Integration Bus V10 Integration Toolkit and IBM MQ V8
- A local IBM MQ queue manager that is named IIBQM with the following local queues: DLQ, COBOL_IN, and XML_OUT
- Lab files in the `C:\labfiles\Lab11-UDP` directory
- IBM MQ SupportPac IH03 (RFHUtil) in the `C:\Software\ih03` directory
- User `iibadmin` that is a member of the “mqm” group

Exercise instructions

Exercise preparation

__ 1. Start the IBM Integration Toolkit if it is not already running.

__ 2. To prevent any conflicts with any existing message flow applications, switch to a new workspace that is named `C:\Workspace\Lab11`.

__ a. In the Integration Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, type: **`C:\Workspace\Lab11`**

__ c. Click **OK**. After a brief pause, the IBM Integration Toolkit restarts in the new workspace.

__ d. Close the **Welcome** window to go to the **Application Development** perspective.

__ 3. Import the solution project interchange file.

__ a. Click **File > Import**.

__ b. Click **IBM Integration > Project Interchange** and then click **Next**.

__ c. Click **Browse** and browse to `C:\labfiles\Lab11-UDP` directory.

__ d. Select the `RouteComplaintWithSubflow_PI.zip` file and then click **Open**.

__ e. Select **RouteComplaint** and then click **Finish**.

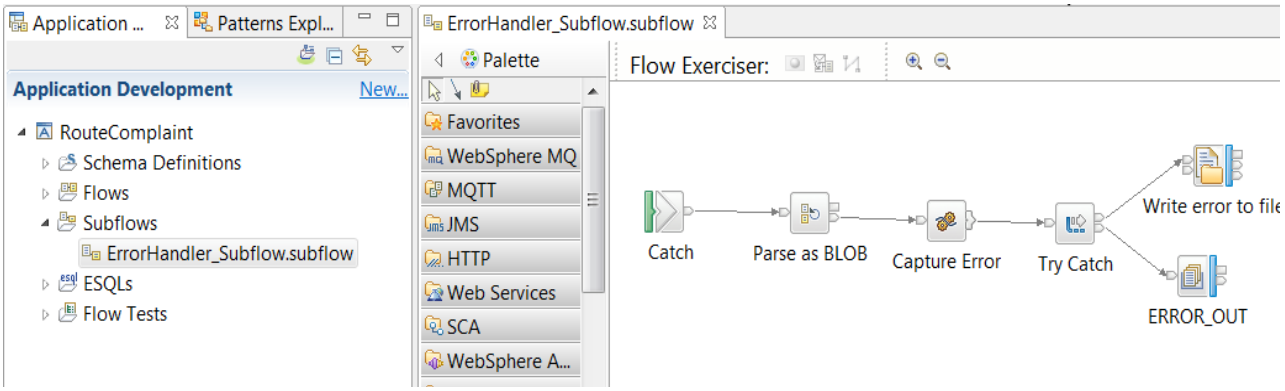
The project interchange file is opened in the **Application Development** view.

Part 1: Add user-defined property to subflow

A user-defined property (UDP) is a property that is defined when you construct a message flow by using the Message Flow editor. ESQL or Java programs inside message flow nodes can use a user-defined property.

__ 1. In the **Application Development** view, expand **RouteComplaint > Subflows**.

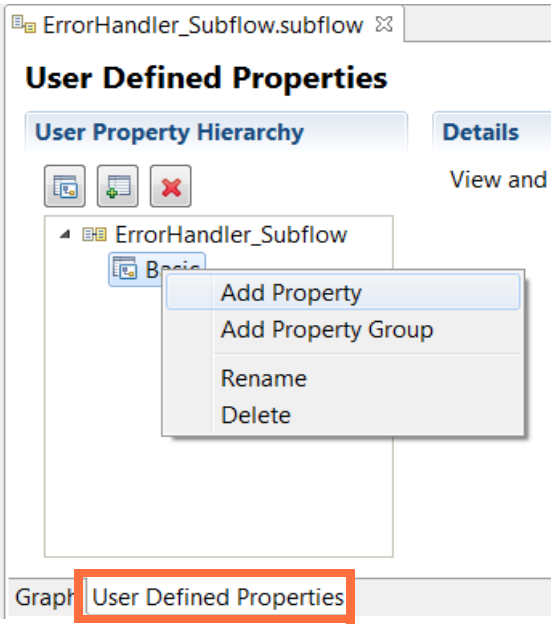
__ 2. Double-click **ErrorHandler_Subflow** to open it in the Message Flow editor.



__ 3. Add a user-defined property that is named **Subflow_Version** to the subflow.

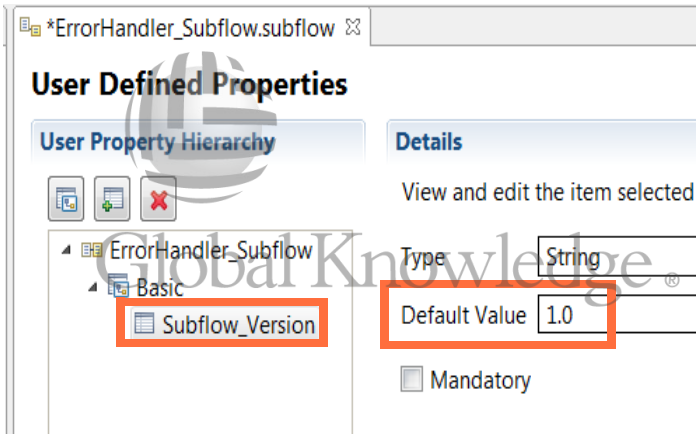
__ a. Click the **User Defined Properties** tab at the bottom of the Message Flow editor view.

__ b. In the **User Property Hierarchy** section, right-click **Basic** under **Error_Handler_Subflow**, and then click **Add Property**.



__ c. Replace the default property name `Property1` by typing: `Subflow_Version`

__ d. Under the **Details** section, change the **Default Value** to: 1.0



__ e. Save the subflow (Ctrl+S).

Part 2: Promote subflow properties to the main flow

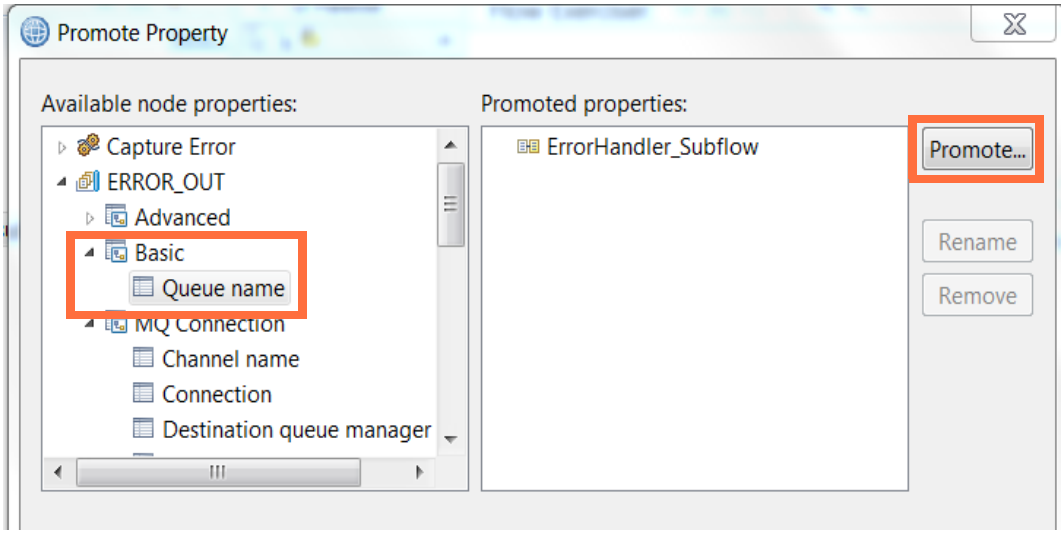
In this part of the exercise, you promote the **Queue name** and **MQ Connection** properties of the ERROR_OUT MQOutput node.

__ 1. Promote the **Queue name** property of the ERROR_OUT MQ Output node in the ErrorHandler_Subflow.

__ a. Click the **Graph** tab at the bottom of the Message Flow editor window to display the message flow diagram.

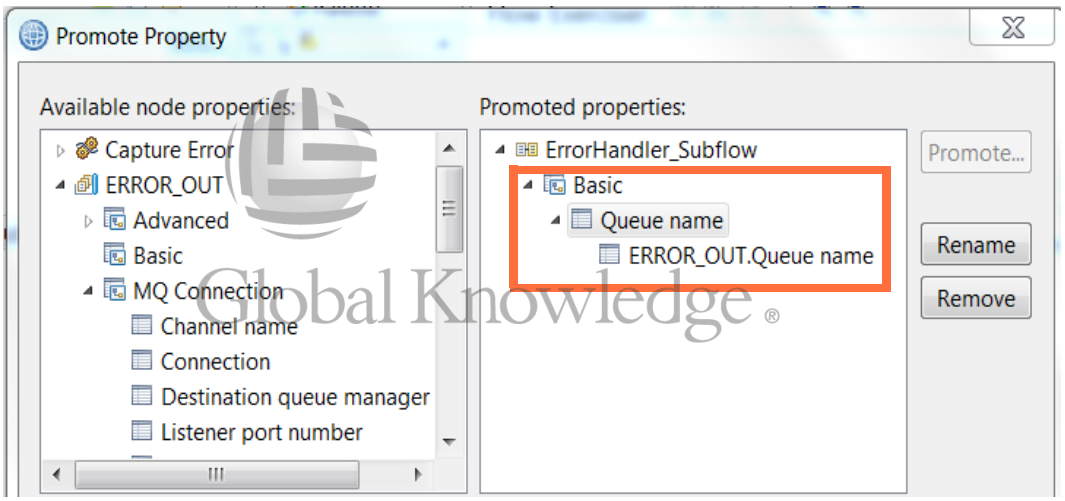
__ b. Right-click the ERROR_OUT MQ Output node, and then click **Promote Property** from the menu.

__ c. In the **Available node properties** window, click **Basic > Queue name** and then click **Promote**.



__ d. On the **Target Selection** window, click **ErrorHandler_Subflow**, and then click **OK**.

The **Queue name** property is shown in the **Promoted properties** window under the group **Basic**. If you expand the property, you see the property name, prefixed by the node label (ERROR_OUT).

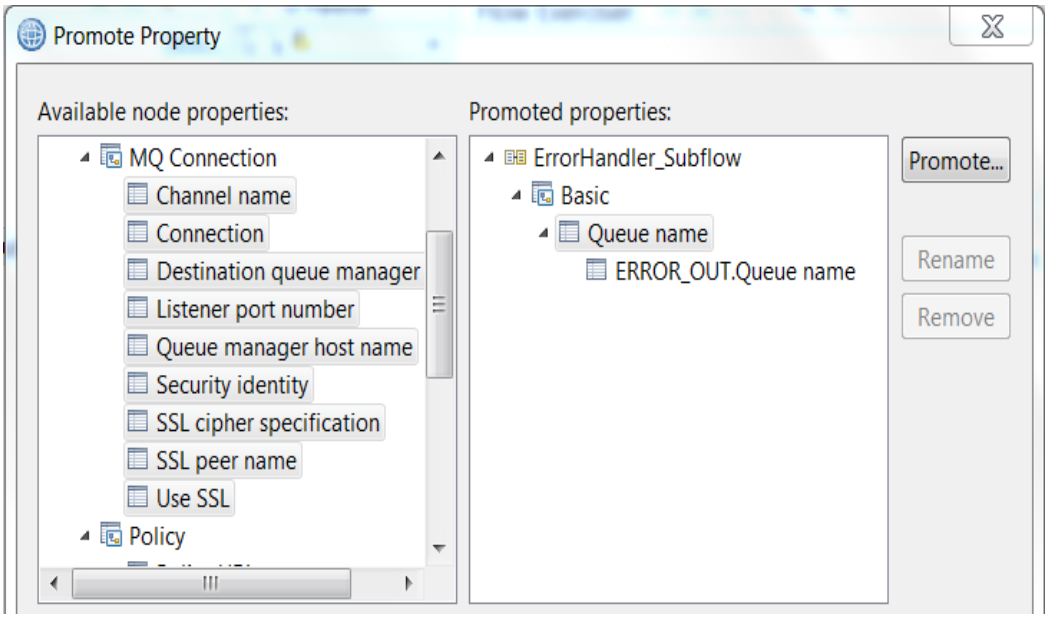


Note

If two or more properties within a message flow have the same name, and you do not assign them to separate groups, a change to the value of the promoted property changes the value of all of the properties by that name within that group. For example, the **Queue name** property is shown as a property to all MQ Input and MQ Output nodes. If you promote one **Queue name** property, changing that promoted value changes all of the values of the **Queue name** property that are within that group.

__ 2. Promote the **MQ Connection** properties of the ERROR_OUT MQ Output node in ErrorHandler_Subflow.

__ a. From the **Available node properties** window, expand **MQ Connection**, hold the Shift key, and select all the properties under **MQ Connection**. Click **Promote**.



__ b. On the **Target Selection** window, click **ErrorHandler_Subflow** and then click **OK**.

__ c. The **MQ Connection** properties are added to the promoted properties under a group that is named **Group1**. If you expand the properties, you see the property name, prefixed by the node name.

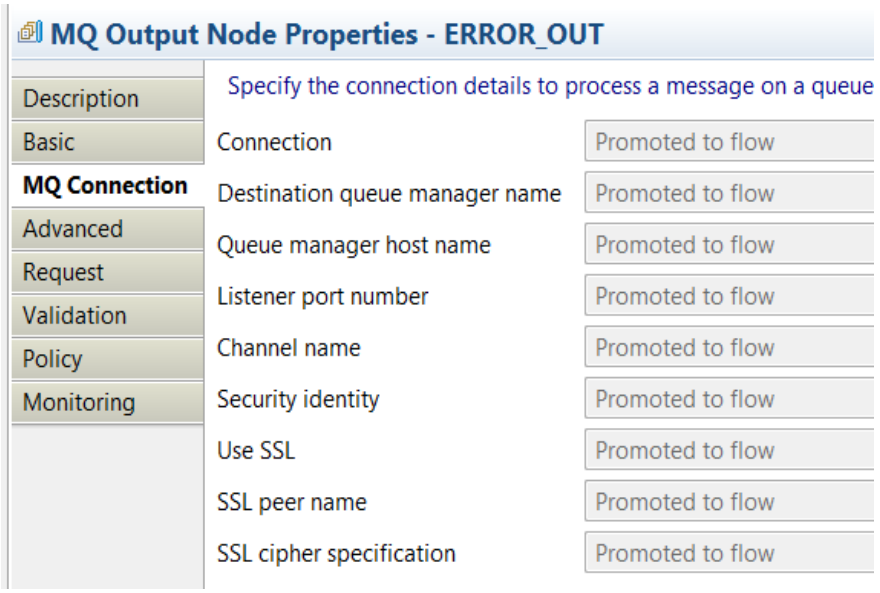
__ d. Rename the group **Group1** to **MQ Connection**.

Click **Group1**, click **Rename** and then type: `MQConnection`.

__ e. In the **Promote Property** window, click **OK**.

__ 3. Click the ERROR_OUT node to display the node Properties view.

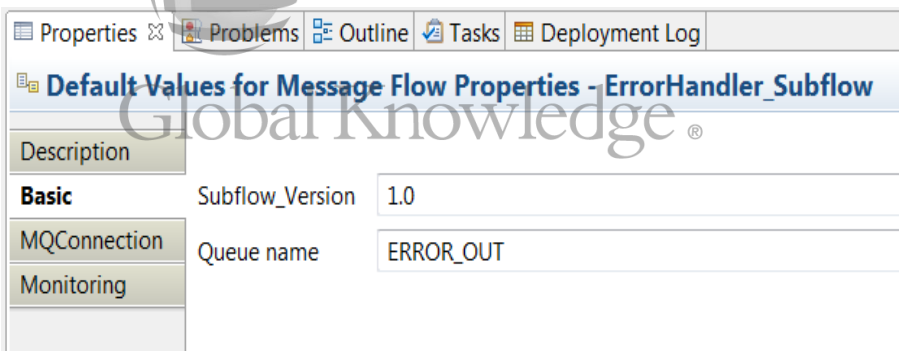
Observe that the **Queue name** on the **Basic** tab and all the properties on **MQ Connections** tab now show as **Promoted to flow**. This display indicates that you cannot change those values here at the “node” level; you must change them at the “flow” level.



__ 4. Review the promoted properties.

__ a. Click in the white space in the Message Flow editor drawing canvas to display the message flow properties.

__ b. The **Queue name** and **MQ Connection** properties that you promoted in the previous steps are displayed, as is the Subflow_Version user-defined property. Although you are not changing the values as this time, this location is where you would do so.



Part 3: Define custom keywords

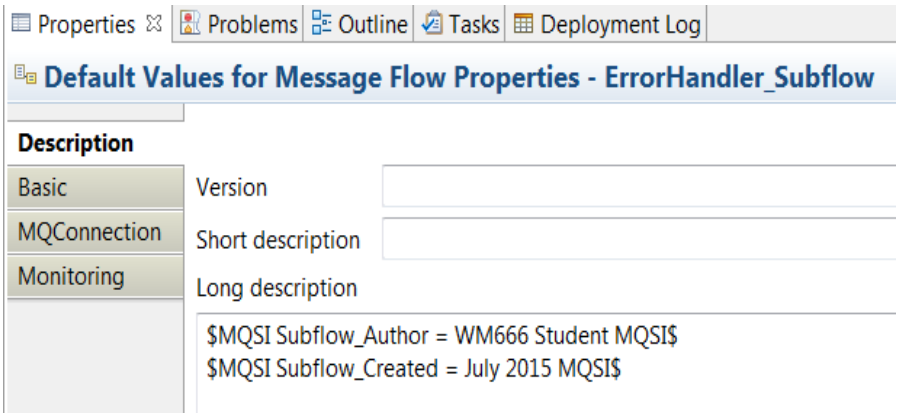
You can define custom keywords to provide more information about message flow, such as the message flow author and a version. The IBM Integration Toolkit interprets custom keywords as additional information to be displayed, in the **Properties** view.

In this part of the exercise, you create keywords for the subflow to be displayed in the deployed message flow.

__ 1. In the **Properties** tab of the ErrorHandler_Subflow, click the **Description** tab.

__ 2. In the Long Description property, type the following lines:

```
$MQSI Subflow_Author = WM666 Student MQSI$
$MQSI Subflow_Created = July 2015 MQSI$
```



This action defines two new keywords, Subflow_Author and Subflow_Created, and assigns values to them. You can use different values from the values that are shown here if you want.

These values are displayed when you examine the properties of the deployed subflow.

__ 3. Save the message flow.

Part 4: View the promoted properties in the BAR file

In this part of the exercise, you build and deploy a BAR file manually so that you can examine the promoted properties.

__ 1. Create a BAR file.

__ a. Click **File > New > BAR** file. The New Bar File windows opens.

__ b. Leave **Container** and **Folder** fields set to their default values. For the **Name** field, type: `UserProperties`

__ c. Click **Finish**. The BAR file is created in the **BARfiles** project and the BAR file opens in the BAR File editor on the **Prepare** view.

__ d. In the BAR file editor **Prepare** view, click **RouteComplaint**.

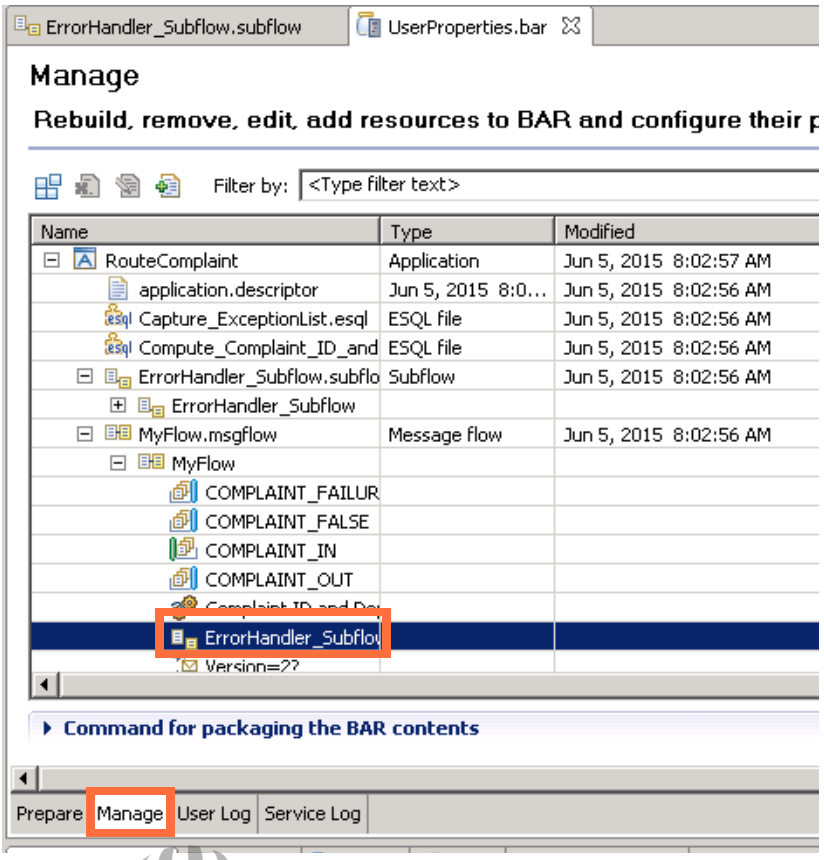
__ e. In the upper-right area of the BAR File editor window, click **Build And Save**. The components are added to the BAR file.

__ f. Click **OK** on the confirmation window.

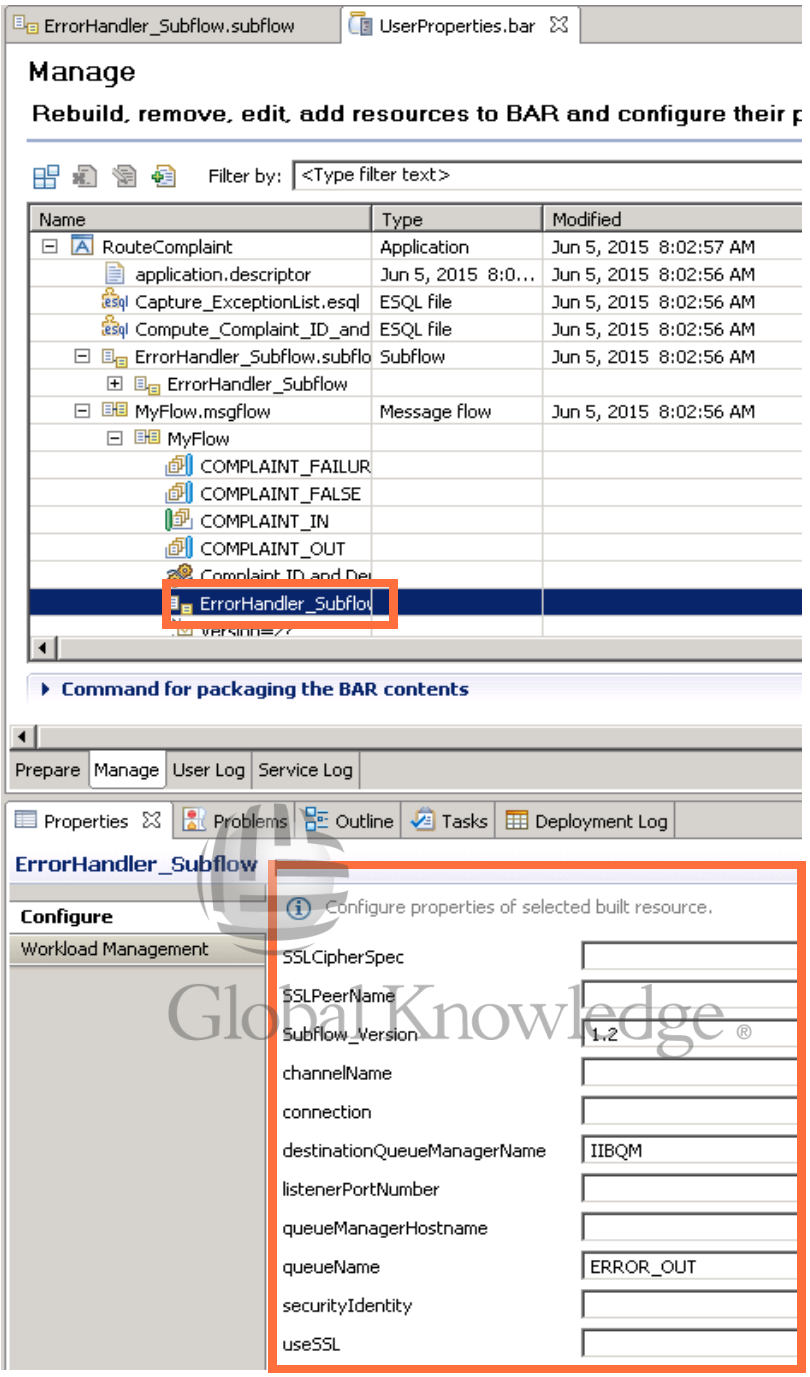
__ 2. Click the **Manage** tab at the bottom of the BAR File editor to go to the BAR File editor **Manage** view.

__ 3. In the BAR file editor **Manage** view, expand the main message flow **MyFlow.msgflow**.

__ 4. Click **ErrorHandler_Subflow** under the **MyFlow** folder.



__ 5. Verify that the properties you promoted to the main flow from the subflow in Part 2 of this exercise are shown as properties of the main flow.



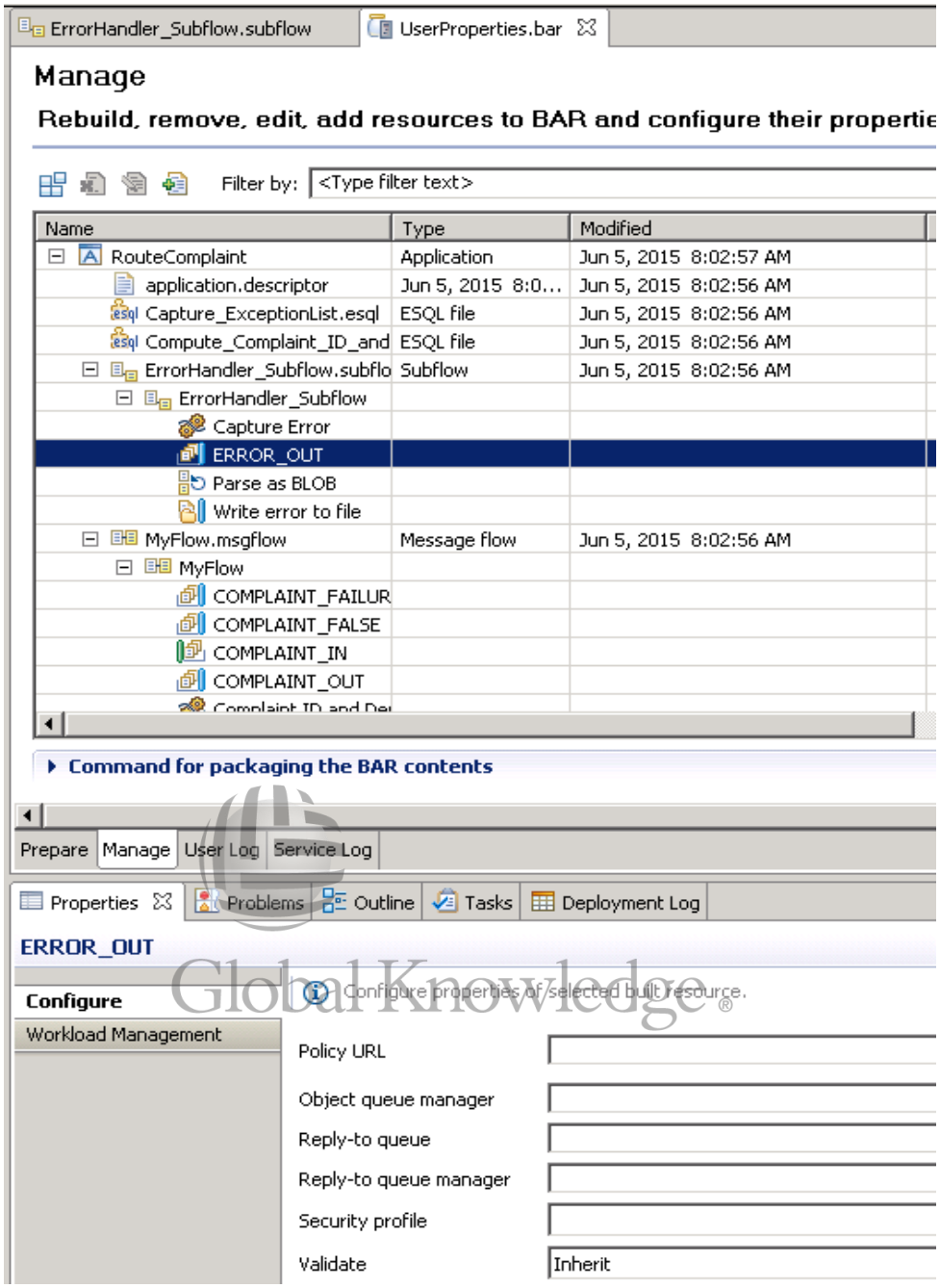
__ 6. In the **ErrorHandler_Subflow** properties view, set the **SubFlow_Version** property to **1.2**.

__ 7. Check the subflow user-defined properties and promoted properties.

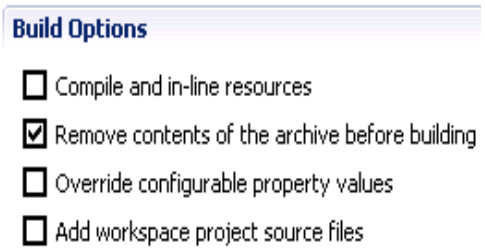
__ a. Click the **Manage** tab on the BAR file editor for `UserProperties.bar` file.

__ b. Expand **ErrorHandler_Subflow.subflow** and click **ERROR_OUT**.

__ c. Verify that the **Queue name** and **MQ Connection** properties are not available in the subflow.



__ 8. On the **Prepare** tab, clear the **Override configurable property values** check box to use the configurable properties that you modified on the **Manage** tab.



__ 9. Click **Build And Save**. The components are added to the BAR file.

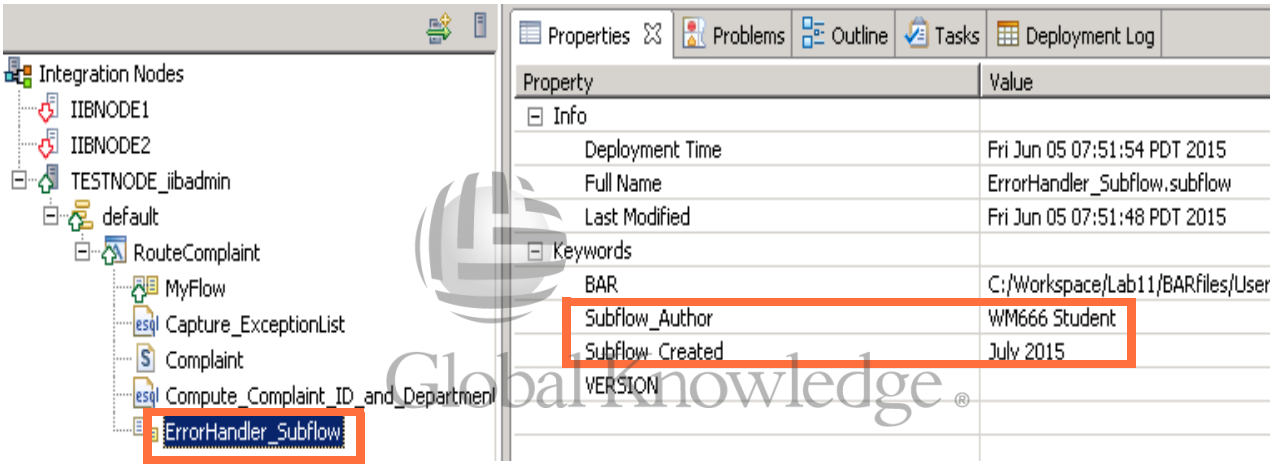
__ 10. Deploy the BAR file to the **default** integration server on TESTNODE_iibadmin.

__ 11. Review the BAR file attributes.

__ a. In the **Integration Nodes** view, expand the **RouteComplaint** application.

__ b. Click **ErrorHandler_Subflow** to display the subflow properties.

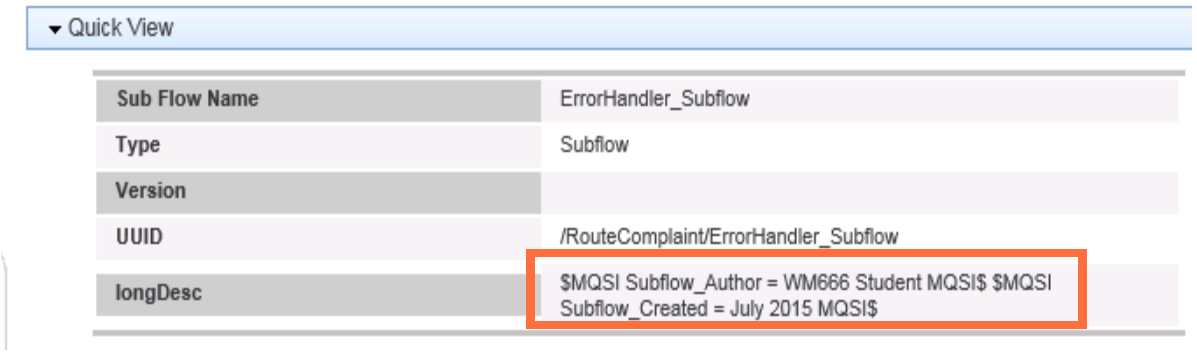
__ c. Verify that the **Properties** view contains the **Subflow Author** and **Subflow Created** keywords that you defined for earlier in the exercise.



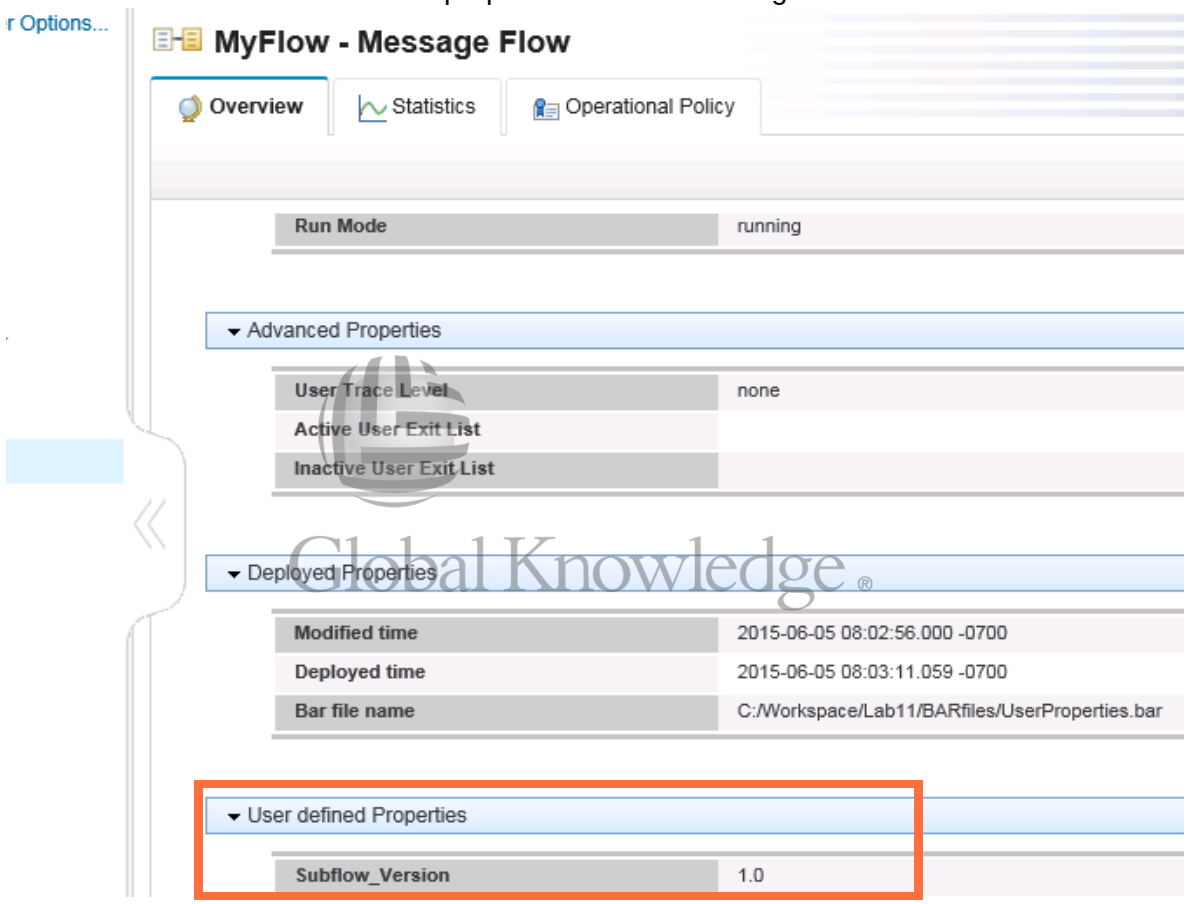
__ 12. Verify that you see the **Subflow_Author** and **Subflow_Created** keywords in the **Properties** view.

Information

You can view the message flow and subflow properties in the IBM Integration web user interface.



You can also view the user-defined properties in the IBM Integration web user interface.



Clean up the environment

__ 1. Close all open editors.

__ 2. In the **Integration Nodes** view, right-click the **default** integration server and click **Delete > All Flows and Resources**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you added user-defined properties to a message flow.

In the second part of this exercise, you promoted subflow properties to the main flow.

In third part of this exercise, you created keywords for the subflow.

In the fourth part of this exercise, you built and deployed a BAR file manually so that you could examine the promoted properties and keywords.

Having completed this exercise, you should be able to:

- Add a user-defined property to message flow
- Promote subflow properties to the main flow
- Define custom keywords
- Set configurable properties in the BAR file
- View BAR file properties at run time

Appendix A. Exercise solutions

Exercise 1, Part 3, Step 5:

Click each of the **Properties** tabs for the HTTP Input node to examine the node properties and answer the following questions.

__ a. What is the message domain (parser) that the HTTP Input node uses in this flow?
Answer: XMLNSC (on the **Input Message Parsing** tab)

__ b. What is the parse timing?
Answer: On demand (on the **Input Message Parsing > Parser Options** tab)

Exercise 2

A solution project interchange file that contains the completed message flow is provided in the `C:\labfiles\Lab02-CreateSimpleFlow` directory.

Exercise 3

A solution project interchange file that contains the completed message flow and the modified BAR file is provided in the `C:\labfiles\Lab03-MQ` directory.

Exercise 4

A solution project interchange file that contains the completed message flow is provided in the `C:\labfiles\Lab04-Routing\solution` directory.

Exercise 5

The solution DFDL model is provided in the `DFDLModel_Solution_PI.zip` project interchange file in the `C:\labfiles\Lab05-DFDL` directory.

Exercise 6

The solution message flow is provided in the `FileToQ_Solution_PI.zip` project interchange file in the `C:\labfiles\Lab06-Files` directory.

Exercise 7

The solution project interchange file, user trace file, and Trace node file are provided in the `C:\labfiles\Lab07-PDTools\solution` directory.

Exercise 8

The solution project interchange file that contains the message flow and error handler subflow and a sample `errorout.txt` file are provided in the `C:\labfiles\Lab08-ErrorHandler\solution` directory.

Exercise 9

A solution project interchange file that contains the message flow, message models, and database definition file is provided in the `C:\labfiles\Lab09-DBmap\solution` directory.

Exercise 10

The solution project interchange files for the Compute node and the JavaCompute node are provided in the `C:\labfiles\Lab10-Compute\solution` directory.

Exercise 11

The solution project interchange file for this exercise, `RouteComplaintWithSubflow_Solution_PI.zip`, is provided in the `C:\labfiles\Lab11-UDP` directory.

Global Knowledge®

Global Knowledge®