Course Exercises Guide

IBM App Connect Enterprise V11 Application Development

Course code WM668 / ZM668 ERC 1.0

IBM Training

January 2020 edition

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

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, which should always be completed. It might be necessary to complete these sections before you can start later exercises. If you have sufficient time and want an extra challenge, some exercises might also include optional sections that you can complete.

Important

The exercises in this course use a set of lab files that might include scripts, applications, files, solution files, PI files, and others. The course lab files can be found in the following directory:

```
C:\labfiles
```

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

User accounts

Type	User ID	Password
Operating system	Administrator	passw0rd
Developer user	AceAdmin	passw0rd

Important

Online course material updates might exist for this course. To check for updates, see the Instructor wiki at http://ibm.biz/CloudEduCourses.

Exercise 1. Importing and testing a message flow

Estimated time

00:30

Overview

This exercise introduces you to the IBM App Connect development environment. To become familiar with the IBM App Connect Enterprise Toolkit views and navigator, you import a simple message flow project and examine the message flow components and properties. You also use the IBM App Connect Enterprise Toolkit Flow exerciser to test the message flow. Before importing the message flow, you create and start an integration server.

Objectives

After completing this exercise, you should be able to:

- Create and start an integration server
- Import an IBM App Connect Enterprise project interchange file
- Use the Message Flow editor to examine the message flow components and properties
- Test the message flow by using the IBM App Connect Enterprise Toolkit Flow exerciser

Introduction

In the first part of this exercise, you use the command line utility to create an integration server. In IBM App Connect Enterprise V11, integration servers can exist independently and do not need an integration node definition. After starting the integration server, you access it from the IBM App Connect Enterprise Toolkit.

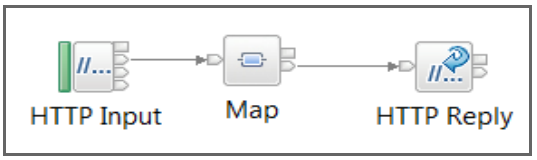
Important

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 the second part of this exercise, you import an IBM App Connect Enterprise 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 App Connect Enterprise 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 IBM App Connect Enterprise 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 App Connect Enterprise V11 Enterprise Toolkit
- The lab files in the `C:\labfiles\Lab01-TestSimpleFlow` directory

Exercise instructions

Part 1: Create the integration server by using the Console

In this part of the exercise, you create and start an integration server. Then, you access the server in the IBM App Connect Enterprise Toolkit so that you can develop and test a simple message flow.

__ 1. Create and start an integration server.

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

User name: `AceAdmin`

Password: `passw0rd`

__ b. If Windows started Server Manager, close it.

__ c. Open the IBM App Connect Enterprise Console.

Before starting an integration server for the first time, you create a working directory

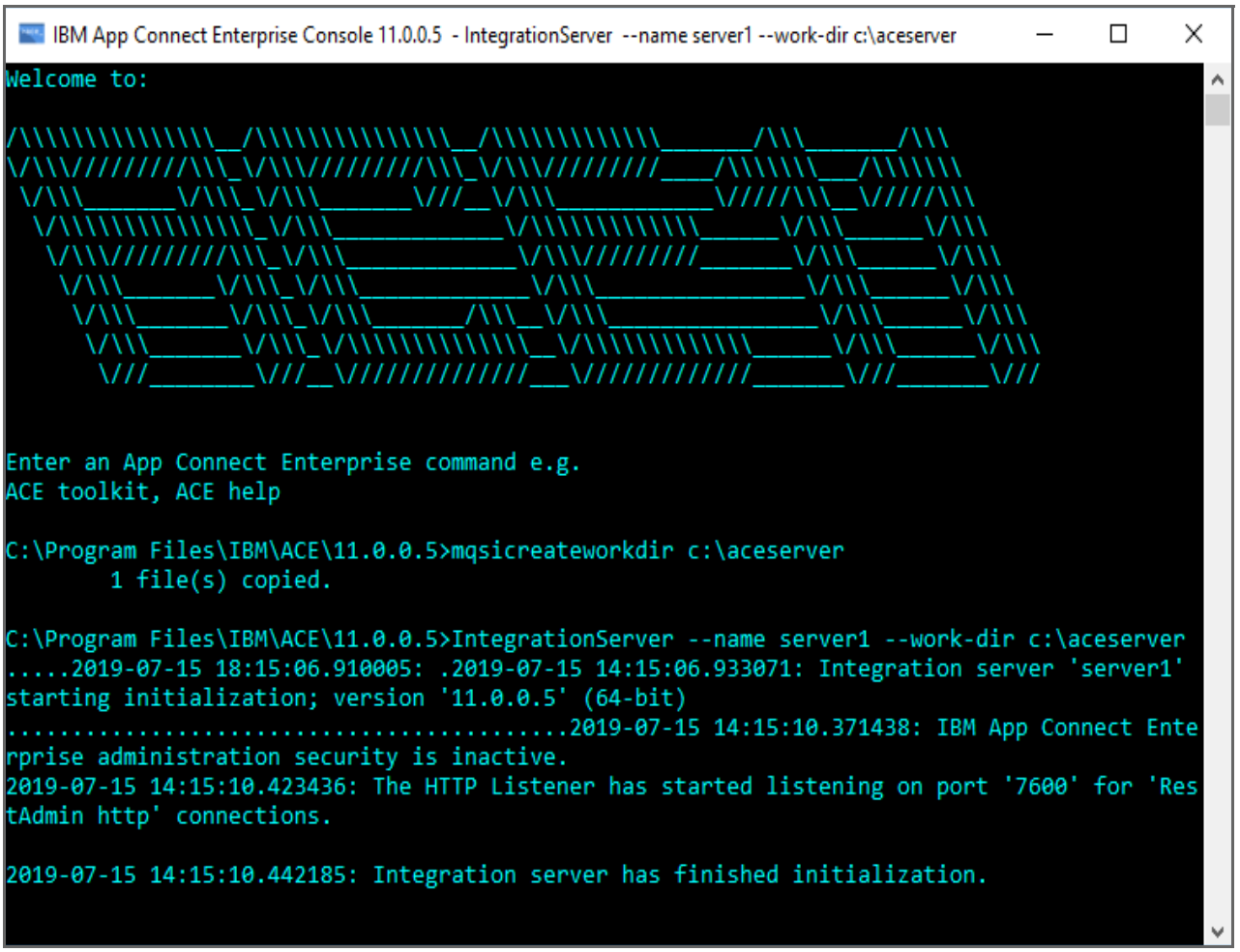
__ d. Create a work directory for your integration server, by running the `mqsicreateworkdir` command, specifying the full path to the directory that you want to create. Enter the following command in the App Connect Enterprise Console:

```
mqsicreateworkdir c:\aceserver
```

__ e. Start the integration server by running the `IntegrationServer` command. Enter the following command in the Console:

```
IntegrationServer --name server1 --work-dir c:\aceserver
```

When the integration server is ready, you will see a message that initialization has finished.



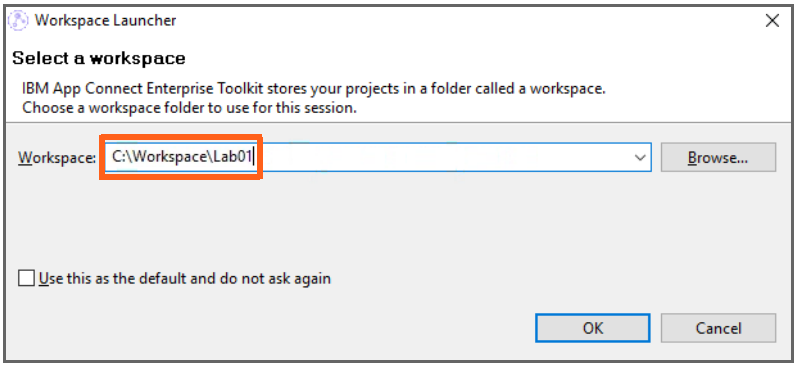
You can now interact with the running integration server; for example, by using the Enterprise Toolkit or the web user interface. You can minimize this window, but do not close it.

__ 2. Use the IBM App Connect Enterprise Toolkit to connect to the integration server.

__ a. Start the IBM App Connect Enterprise Toolkit by double-clicking the **IBM App Connect Enterprise Toolkit 11.0.0.5** shortcut on the desktop.

You can also start the IBM App Connect Enterprise Toolkit by clicking **Start** > A**ll Programs** > **IBM App Connect Enterprise 11.0.0.5** > **IBM App Connect Enterprise Toolkit 11.0.0.5** from the Windows desktop.

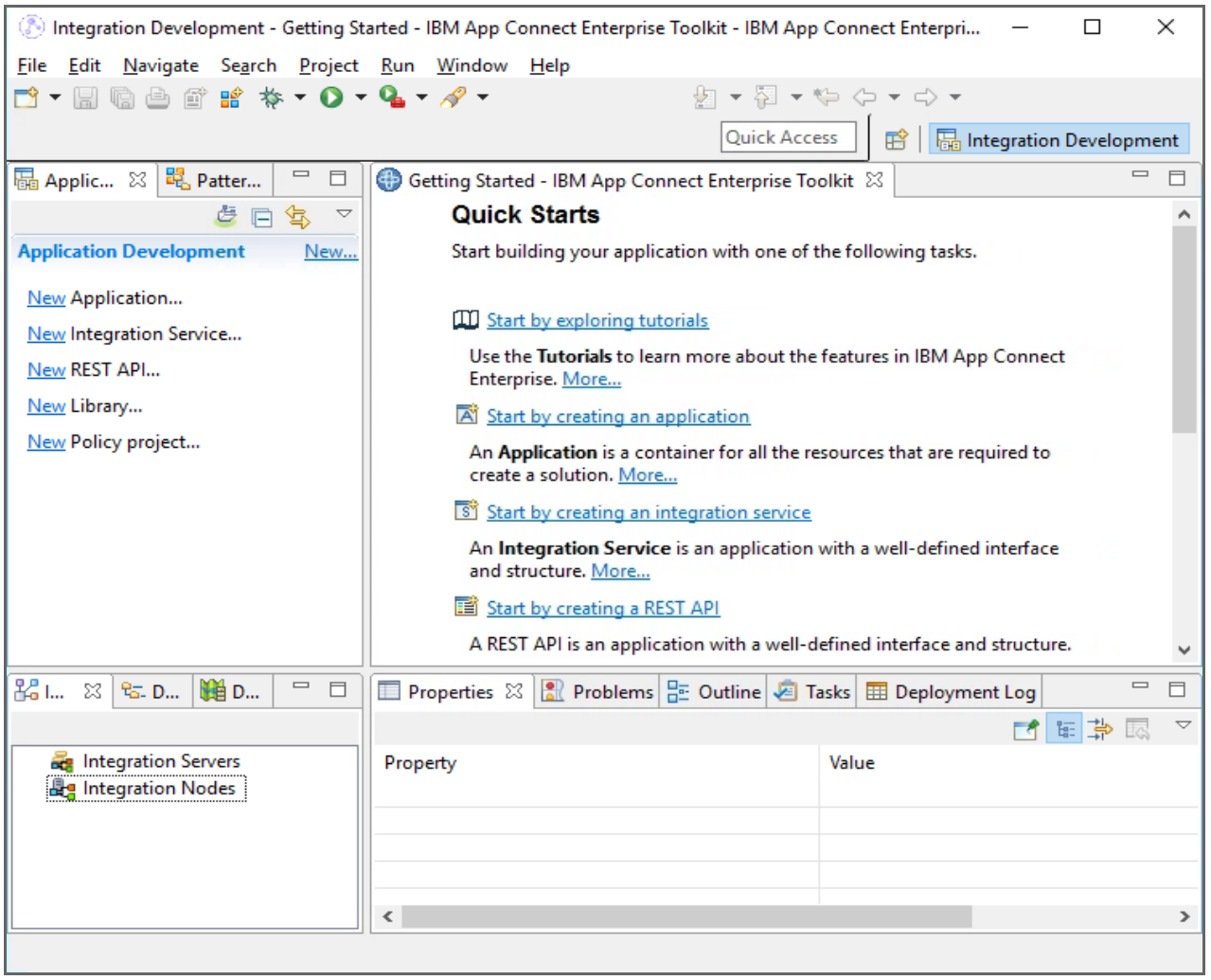
__ b. For the Workspace, enter `C:\Workspace\Lab01`



At the beginning of each lab, you create a new folder to contain the exercise artifacts.

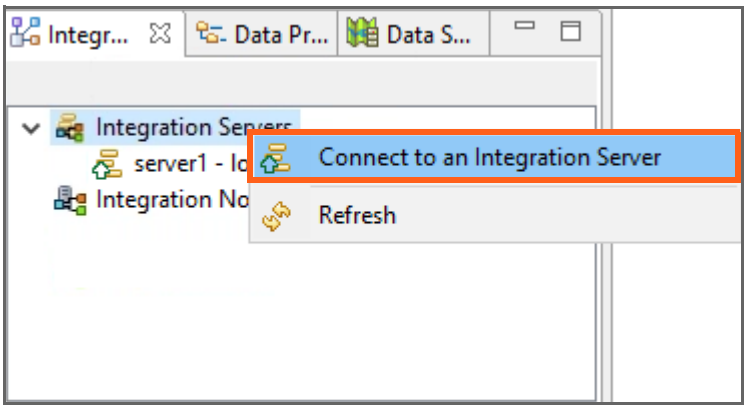
__ c. Click **OK**. After several moments, the IBM App Connect Enterprise Toolkit starts and the Welcome page displays.

__ d. Close the Welcome window by clicking the **Go to the IBM App Connect Enterprise Toolkit** link at the top right. The Toolkit opens in the Integration Development perspective with a tab labeled Getting Started - IBM App Connect Enterprise Toolkit.

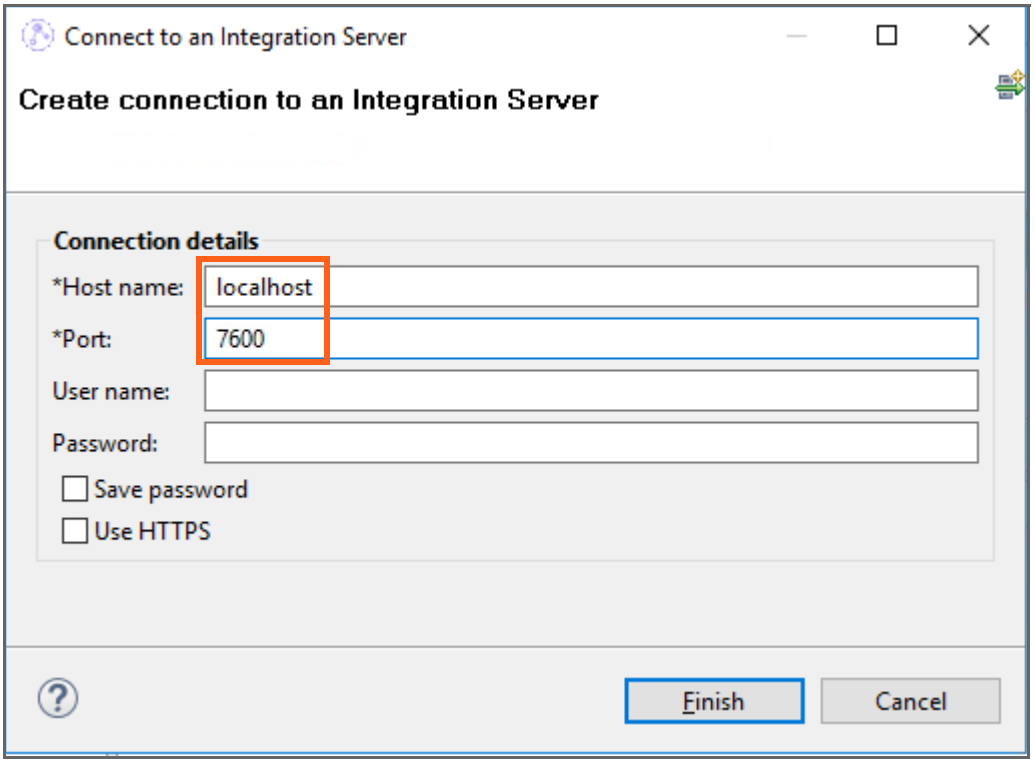


__ 3. Use the IBM App Connect Enterprise Toolkit to connect to the integration server.

__ a. Right-click **Integration Servers** in the Integration Explorer view in the bottom left corner and then select **Connect to an integration server**.

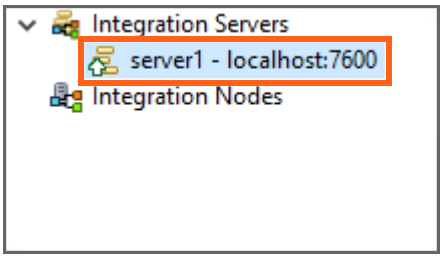


__ b. For the connection details, enter `localhost` for the **Host name** and `7600` for the **Port** of the integration server. Because the integration server is not secured, you do not need to specify the user name and password.



__ c. Click **Finish**.

__ d. Expand Integration Servers to view the status of the server. A green up arrow indicates the server running.

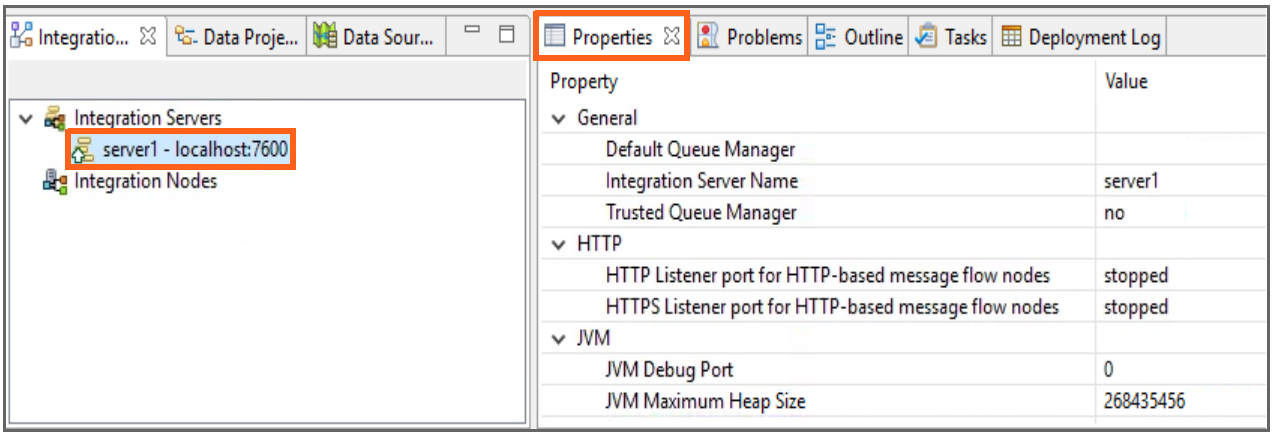


__ 4. Examine the properties of the integration server: server1 - localhost: 7600.

__ a. Click the integration server **server1**.

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

Scroll through the properties to view information about server1. You should also see that there is no trusted queue manager associated with this integration server.



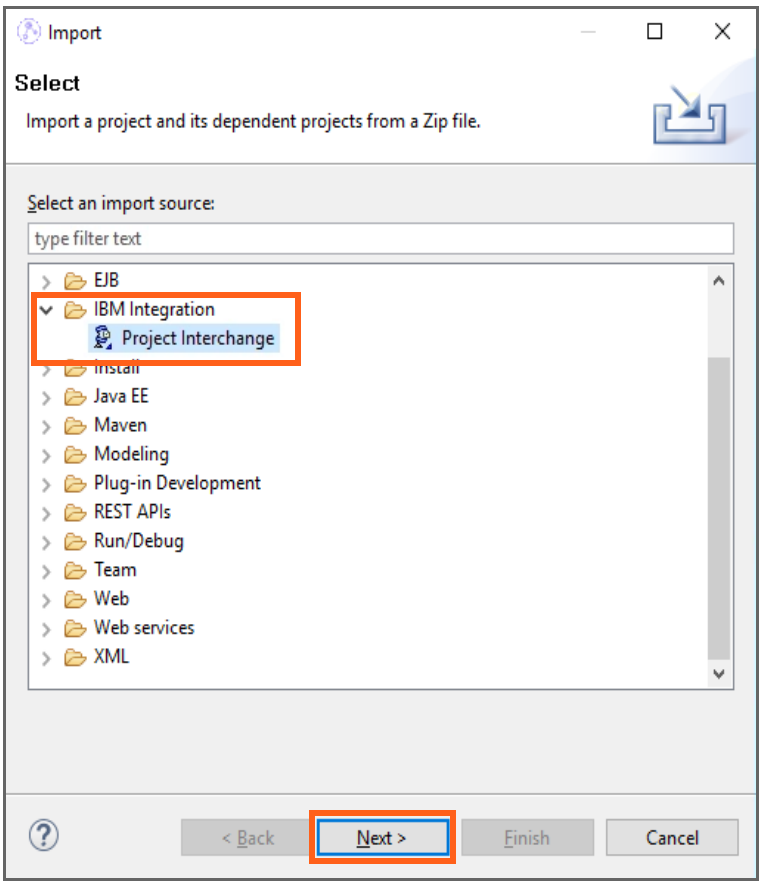
Part 2: Import a project interchange file

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

__ 1. Import a project interchange file.

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

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

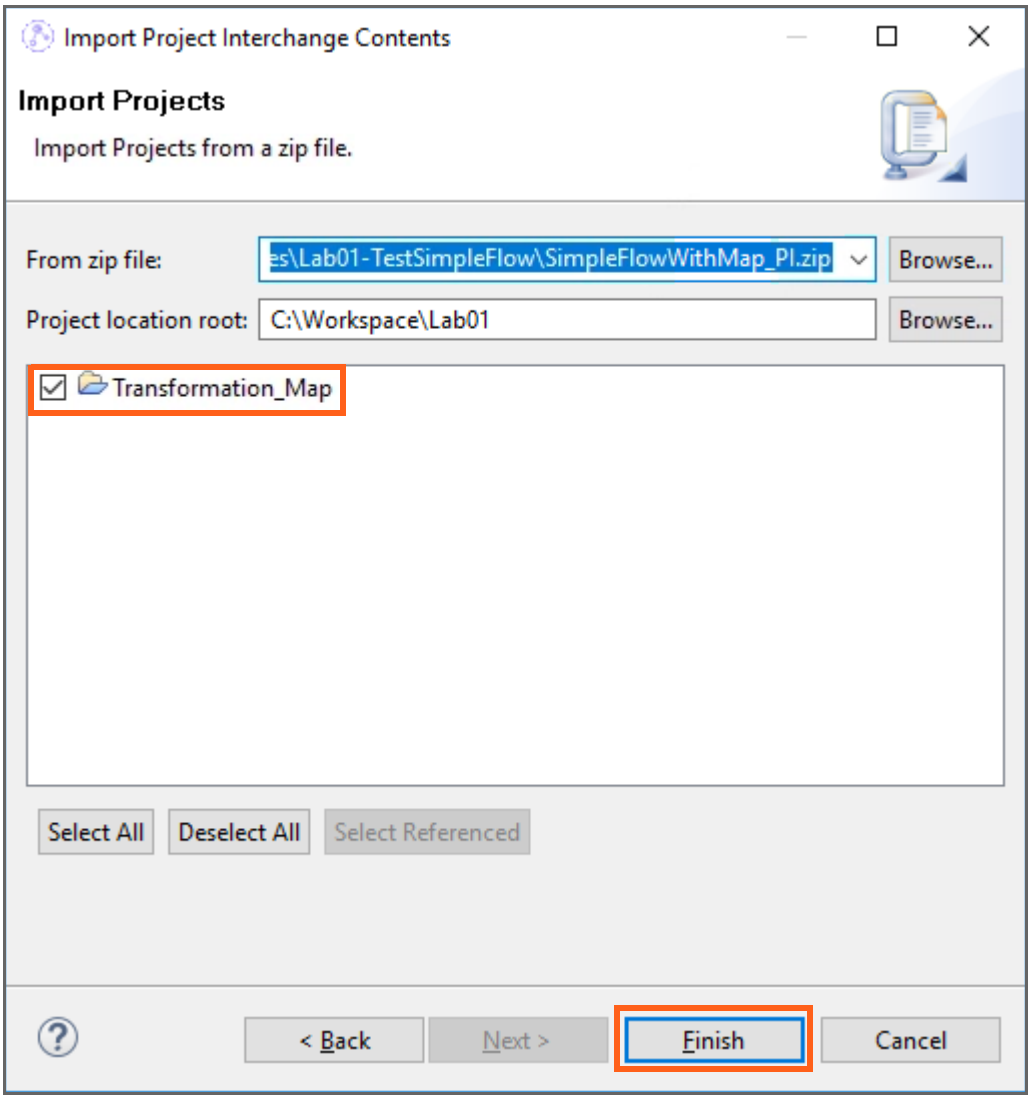


__ c. To the right of **From zip files**, click **Browse** to locate the project interchange file.

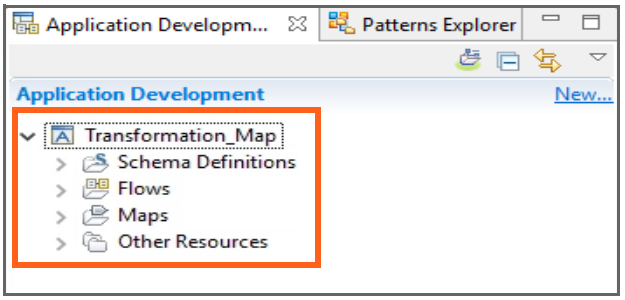
__ d. Browse to the `C:\labfiles\Lab01-TestSimpleFlow` directory, select the `SimpleFlowWithMap_PI.zip` file, and then click **Open**.

__ e. Ensure that the **Project location root** is set to the current workspace of `C:\Worksplace\Lab01`.

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



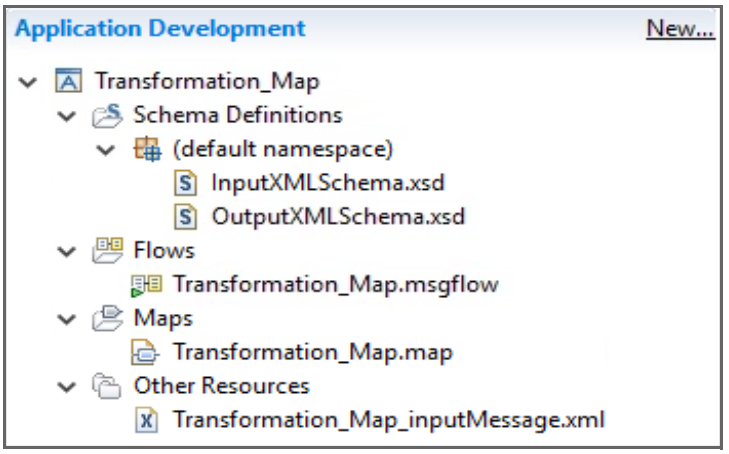
__ g. Verify that the Transformation_Map application is imported into the Application Development navigator by expanding it.



Part 3: Examine the application objects and message flow components

__ 1. View the Transformation_Map objects.

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

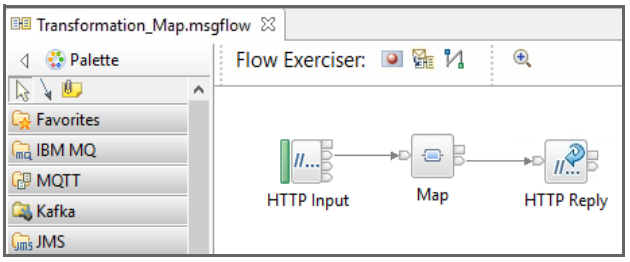


The figure shows the objects that are included in the **Transformation_Map** application. The application includes the following files:

- 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. View the properties for the HTTP Input node.

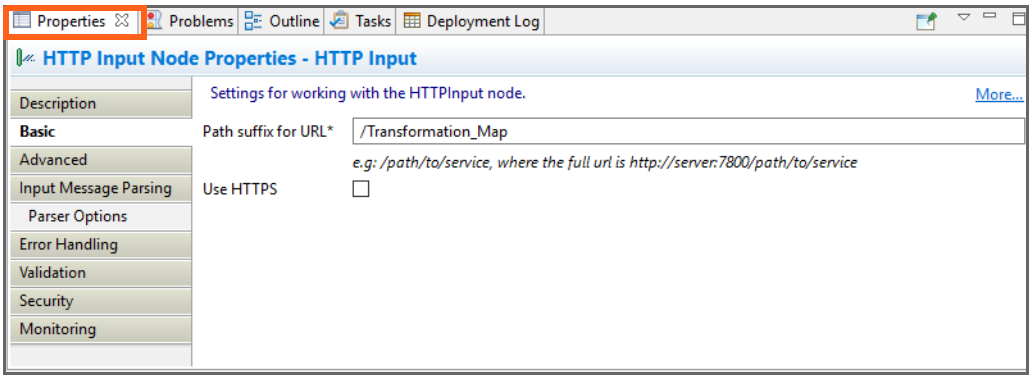
__ a. Double-click the message flow file **Transformation_map.msgflow** in the Application Development view to open it in the Message Flow editor.



The message flow contains three nodes:

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

__ b. In the Message Flow editor, click the **HTTP Input** node to display the node properties. The node properties are shown in the **Properties** view at the bottom of the IBM App Connect Enterprise Toolkit.



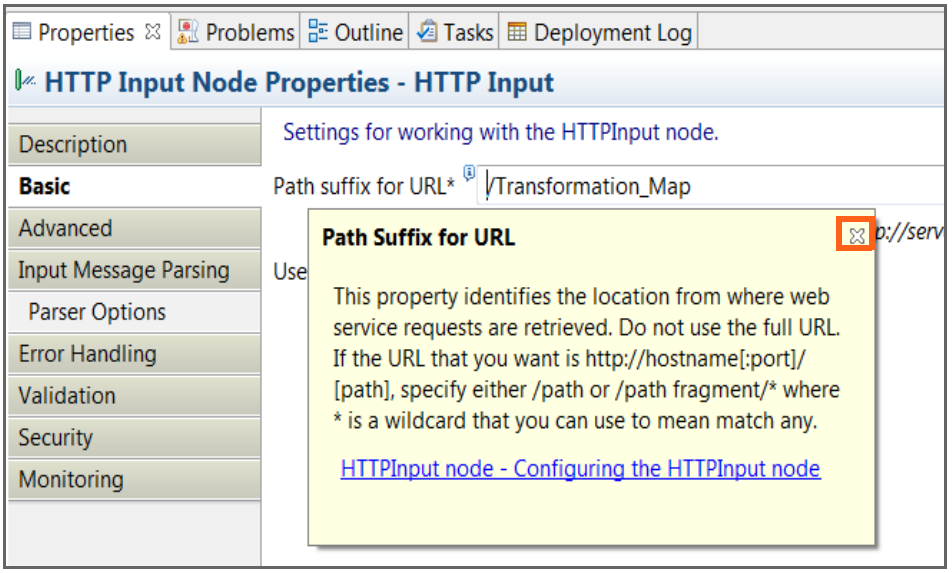
Several properties tabs are shown. By default, the **Basic** tab is selected.
The IBM App Connect Enterprise Toolkit provides context-sensitive help for node properties.

__ 3. 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.



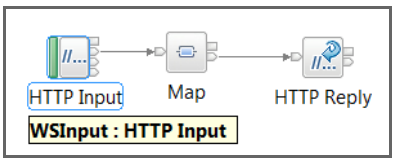
__ 4. Click each of the **Properties** tabs for the HTTP Input node to examine the node properties and answer the following questions.

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

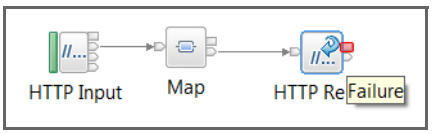
__ b. What is the parse timing?

__ 5. Examine the node properties for the other nodes in the message flow.

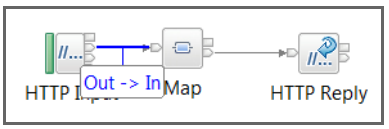
__ a. In the Message Flow editor, hover the mouse pointer over each of the nodes. The node type and node name are displayed in the format `NodeType : NodeName`.



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



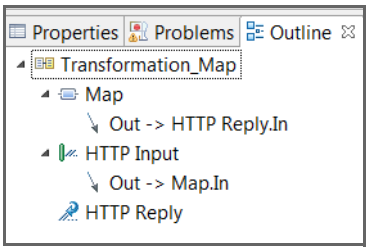
__ c. In the Message Flow editor, hover the mouse pointer over the wires that connect the nodes to display the connection information.



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

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

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



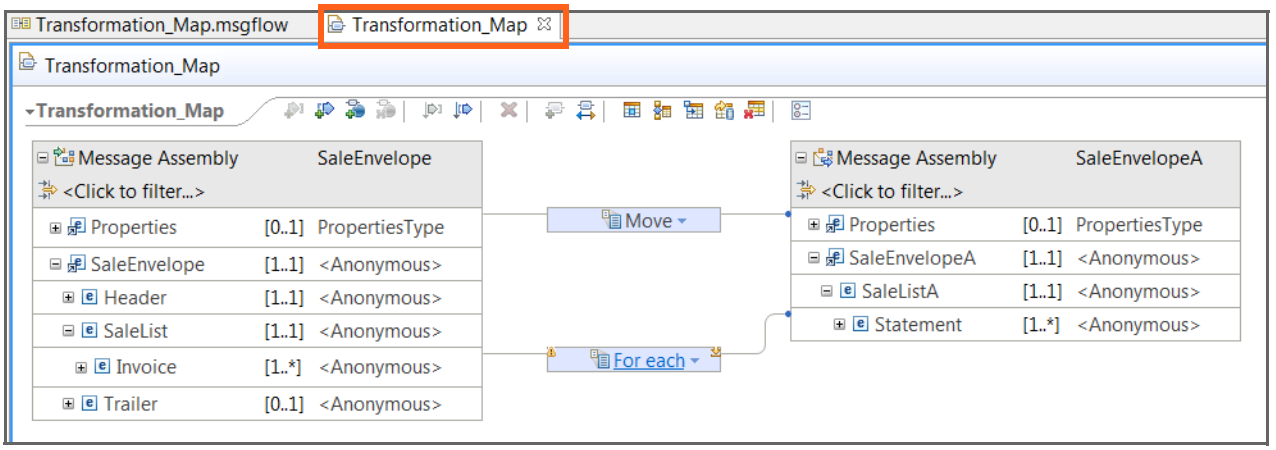
The Outline view shows that the **Out** terminal of the Map node is wired to the **In** terminal of the HTTP Reply node as denoted by **HTTP Reply.In**.

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

__ 6. 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.



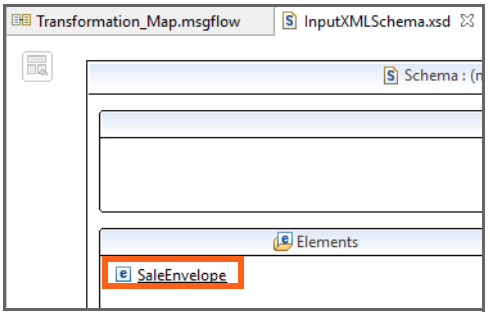
__ c. Double-click the **Transformation Map** tab again so that it resizes the view back to its original size.

__ 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.

__ 7. Examine the XML schema that defines the message flow input.

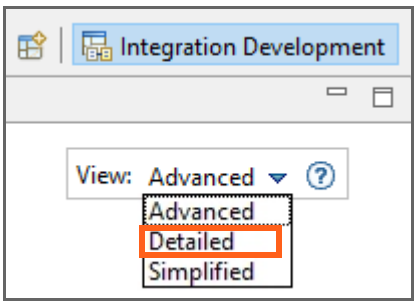
__ a. In the Application Development view on the left, double-click the **InputXMLSchema.xsd** file under the Schema Definitions > (default namespace) folder to open it in the XML Schema editor.

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

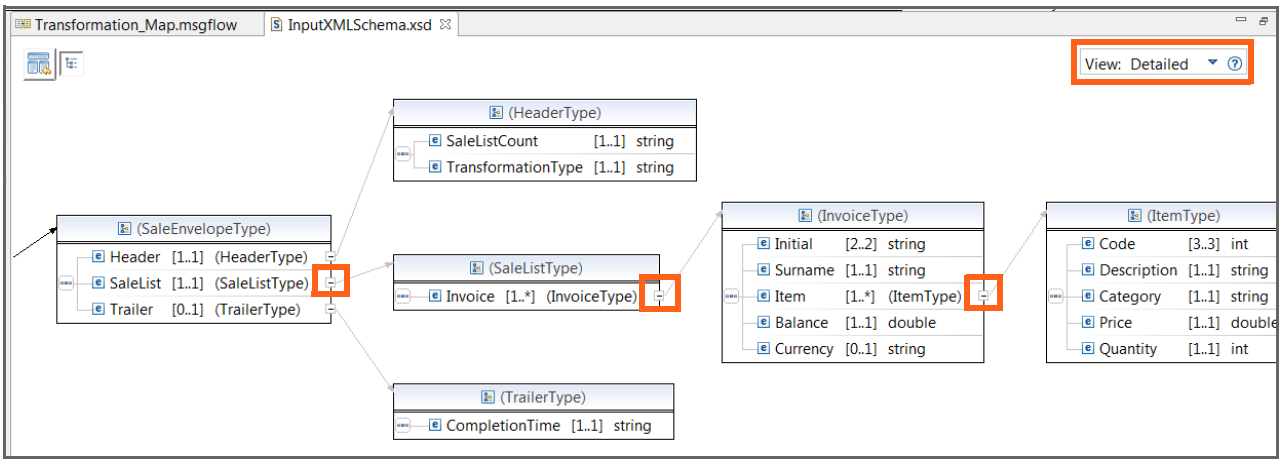


__ c. Double-click the **InputXMLSchema.xsd** tab to expand the XML Schema editor view.

__ d. Ensure that **View** is set to **Detailed**.



__ e. Click the plus signs to expand the XML schema.



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

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

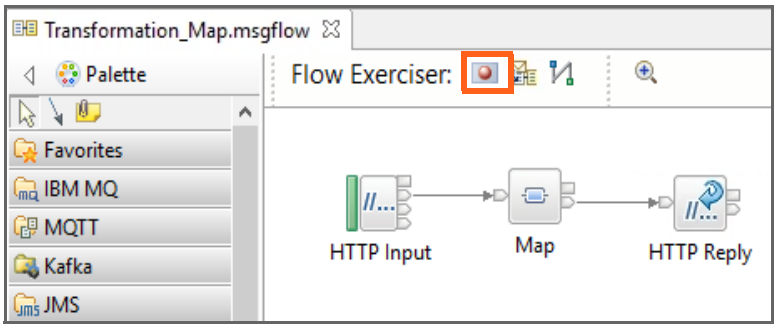
Part 4: Test with the Flow exerciser

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

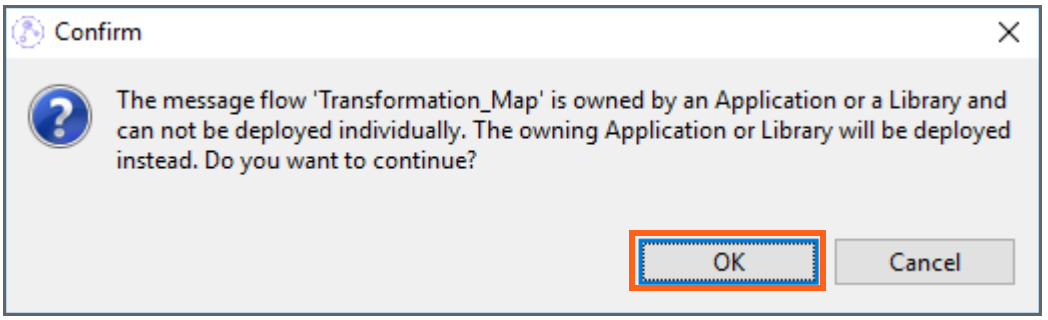
__ 1. Start the Flow exerciser

__ a. Click 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.

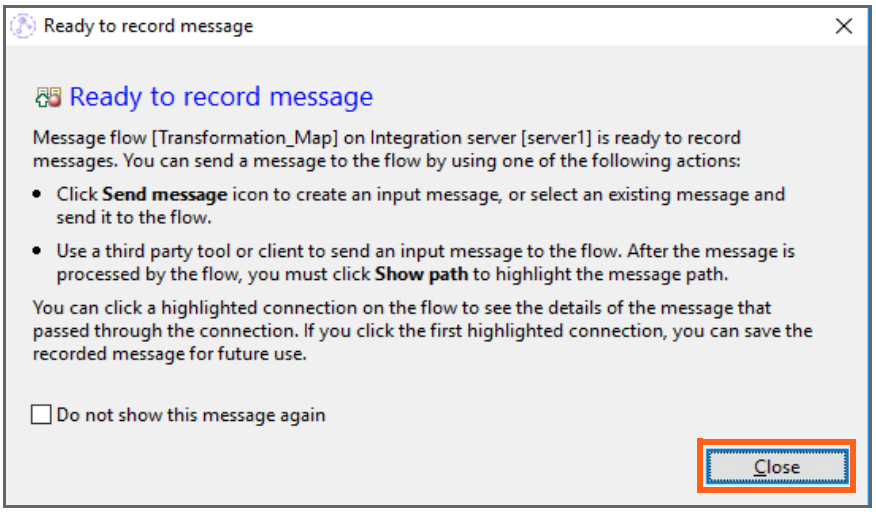


__ b. Click **OK** in the Confirm message box.



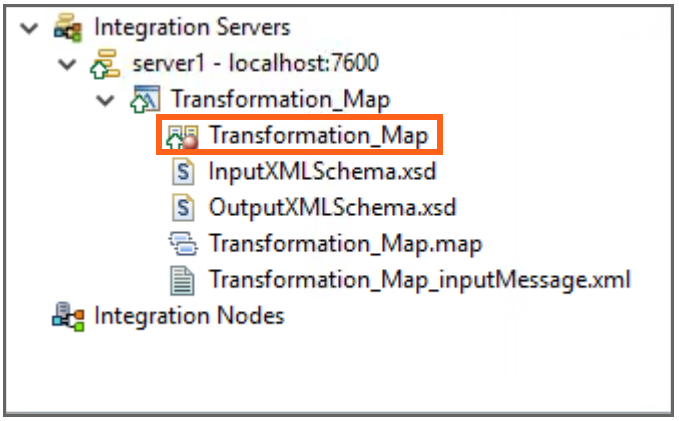
The Flow exerciser creates the BAR file and deploys it to the integration server. If any messages are displayed while the Flow exerciser is running, ignore them; they are information messages and disappear on their own.

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

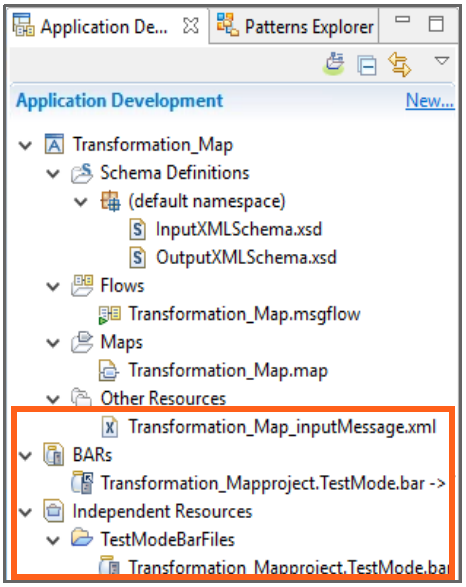


__ 2. Verify that the Flow exerciser Record mode started and that the application was deployed.

__ a. In the Integration Explorer view, expand the server to display the contents. Verify that **Transformation_Map** is in record mode (signified by the Recording icon red circle).



__ b. 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.

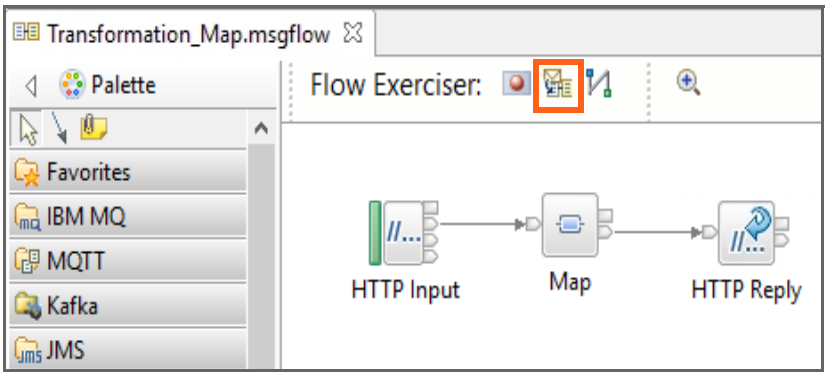


These projects contain the BAR file (with a `.bar` extension) that the Flow exerciser built and deployed to the integration server.

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

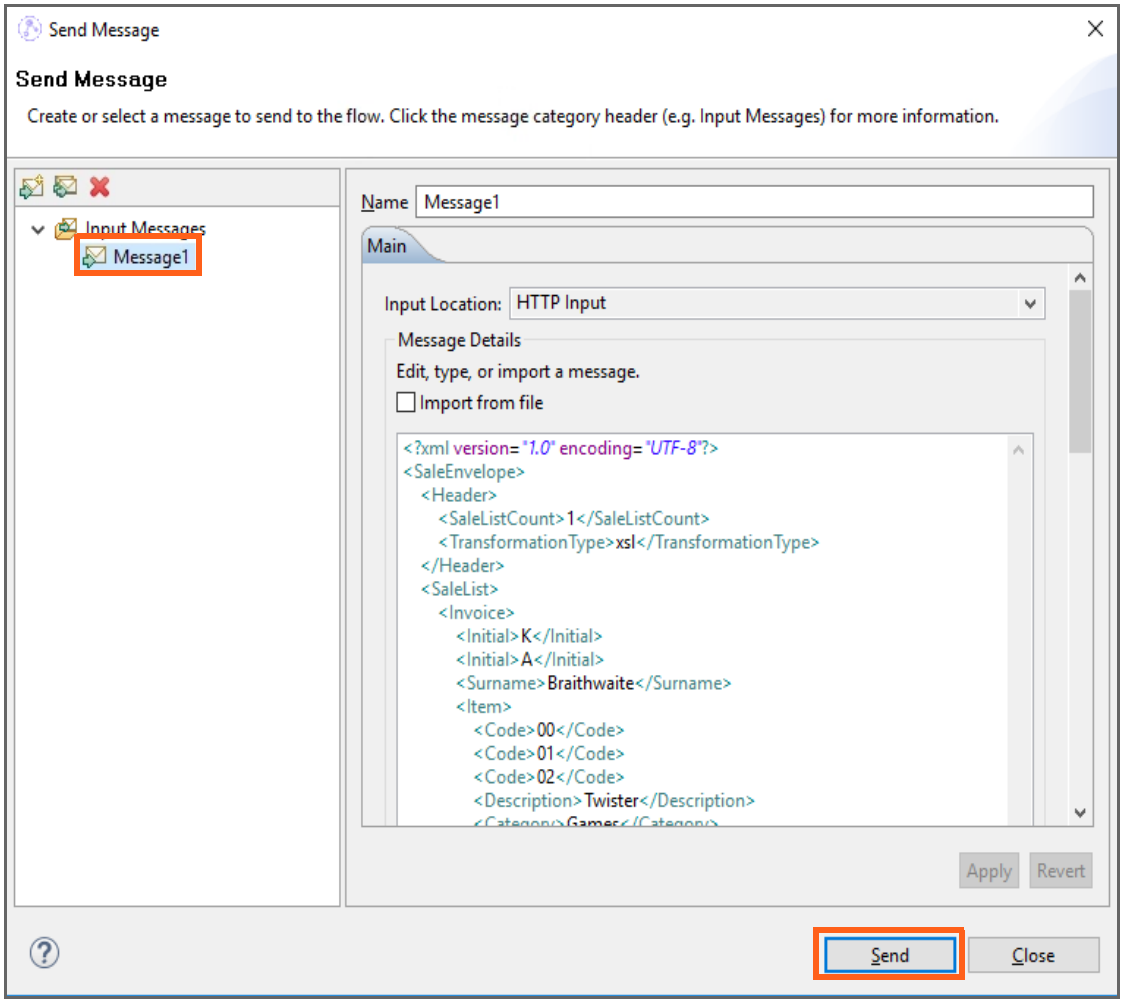
__ 3. Send a test message to the Transformation_Map message flow.

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



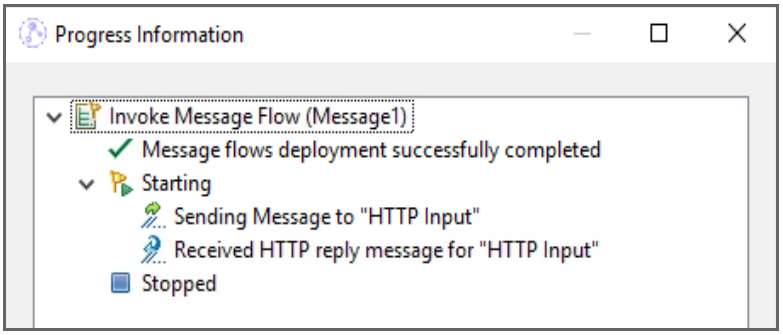
__ b. In the Send Message window, click **Message1**. Take a moment to examine the message.

__ c. Click **Send**.

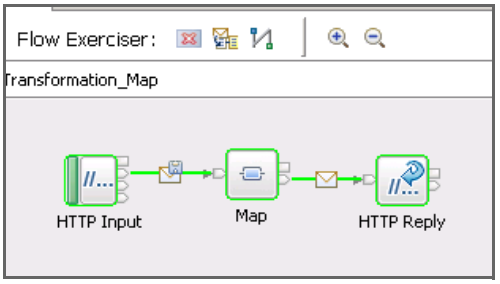


After some moments, the test runs. The Progress Information window shows you the status of the message flow test. In this exercise, the Progress Information shows that the message was sent to the HTTP Input node and that the reply message was received.

__ d. Click **Close**.

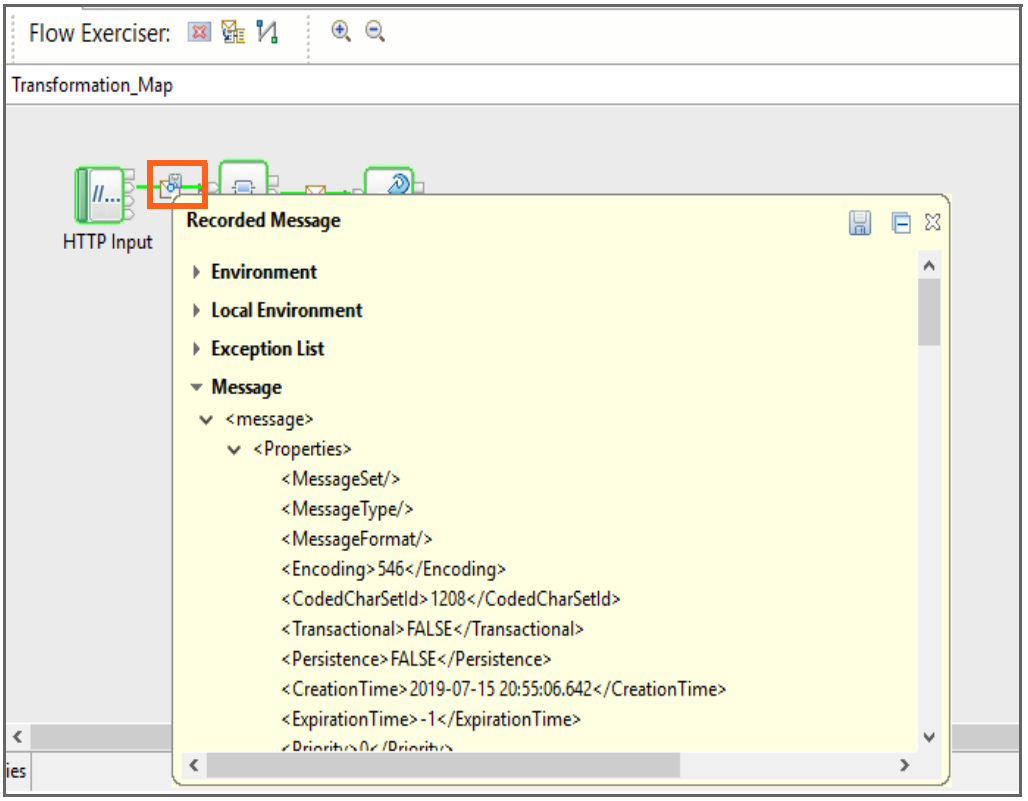


In the Message Flow editor, the message flow updates to show the message path through the flow. The message icons on the connections indicate the points where you can view the message.



__ 4. View the test message.

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



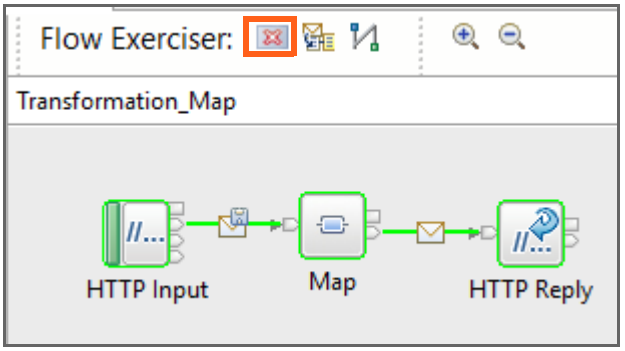
__ b. After you review the input message, close the message window.

__ 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.

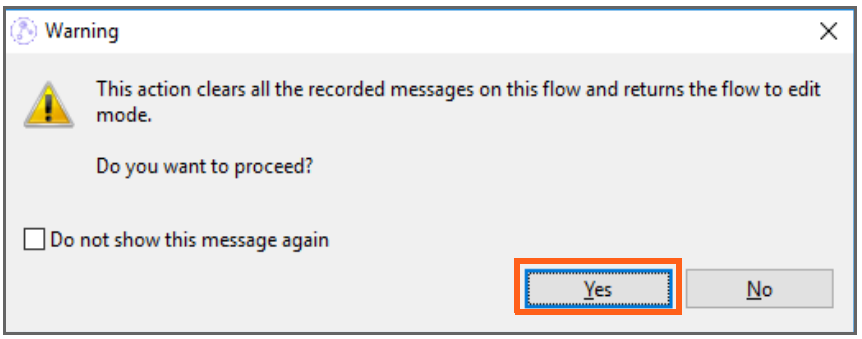
__ 5. Return to the edit mode and stop the Flow Editor

__ a. Click the **Return flow to edit mode** icon at the top of the Message Flow editor.

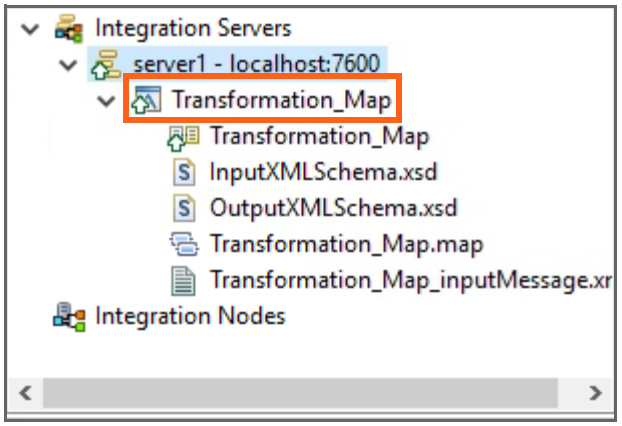


A warning message reminds you that changing back to edit mode clears all the recorded messages on this flow. It also reminds you to stop recording mode on the integration server if you are finished with testing,

__ b. Click **Yes**.



__ c. In the Integration Explorer view, expand the server to display the contents. Verify that Transformation_Map is no longer in record mode.



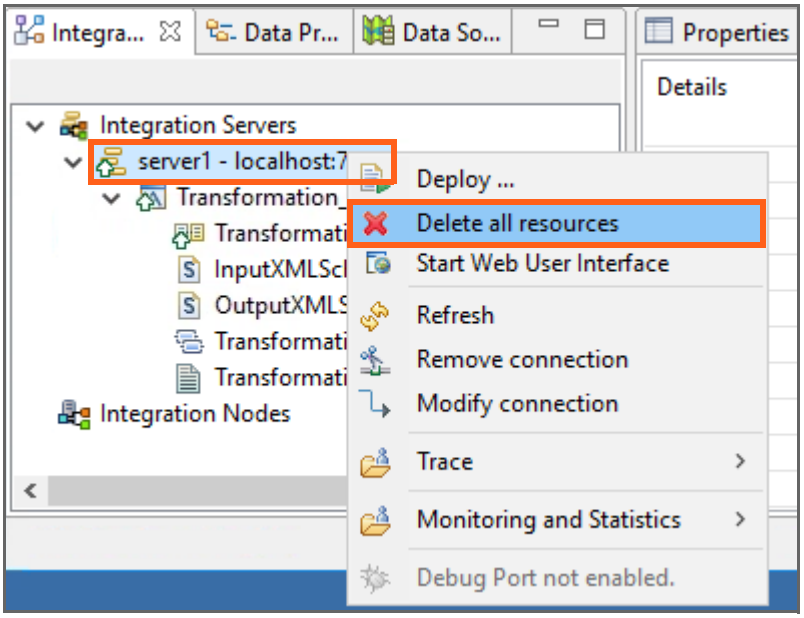
__ d. Close the Transformation_Map message flow in the Message flow editor.

Part 5: Exercise clean-up

At the end of each exercise, an environment clean-up ensures that no potential conflicts occur with the exercises.

__ 1. Remove the **Transformation_Map** application from the integration server.

__ a. Right-click **server1** and then click **Delete all resources**.



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

Transformation_Map is removed from server1.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you created and started an integration server by using the IBM App Connect Enterprise Console. Then, you connected to the server in the IBM App Connect Enterprise Toolkit.

In the second part of this exercise, you imported an IBM App Connect Enterprise 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 App Connect Enterprise map.

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

Exercise 2. Creating a message flow application

Estimated time

01:00

Overview

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

Objectives

After completing this exercise, you should be able to:

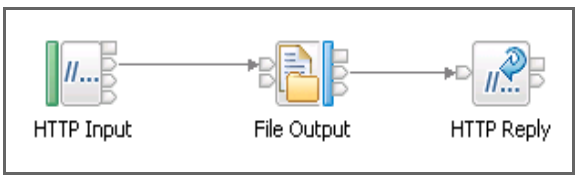
- Create a message flow application
- Use the IBM App Connect Enterprise Flow Exerciser to test the message flow application
- Use the IBM App Connect Enterprise web user interface to check the status of an integration server and message flow application.

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 IBM App Connect Enterprise Flow Exerciser.

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

Requirements

- A lab environment with the IBM App Connect Enterprise V11 Enterprise Toolkit
- The lab files in the `C:\labfiles\Lab02-CreateSimpleFlow` directory

Exercise instructions

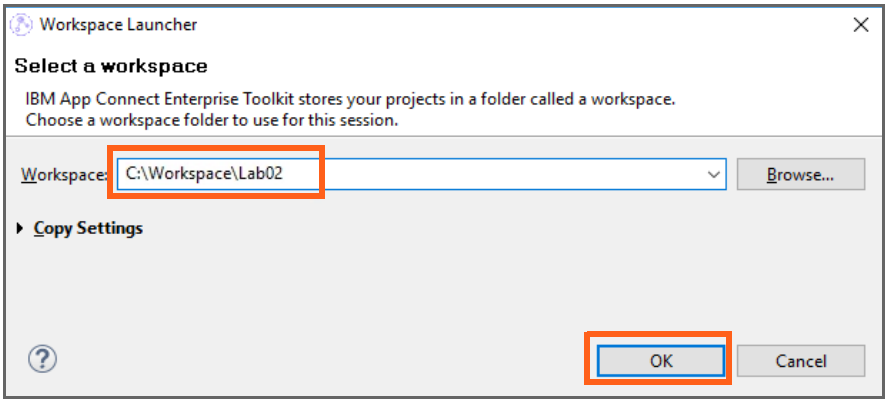
Part 1: Create the message flow application

In this part of the exercise, you create a simple HTTP flow that acts as a 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 IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

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

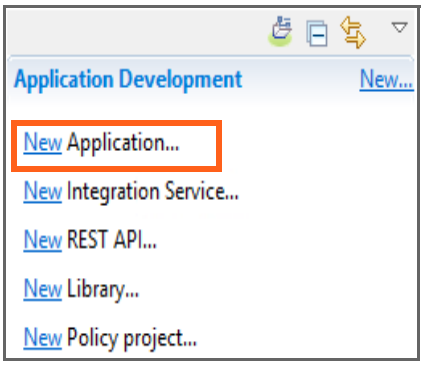


__ c. Click **OK**. After a brief pause, the IBM App Connect Enterprise Toolkit restarts in the new workspace.

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

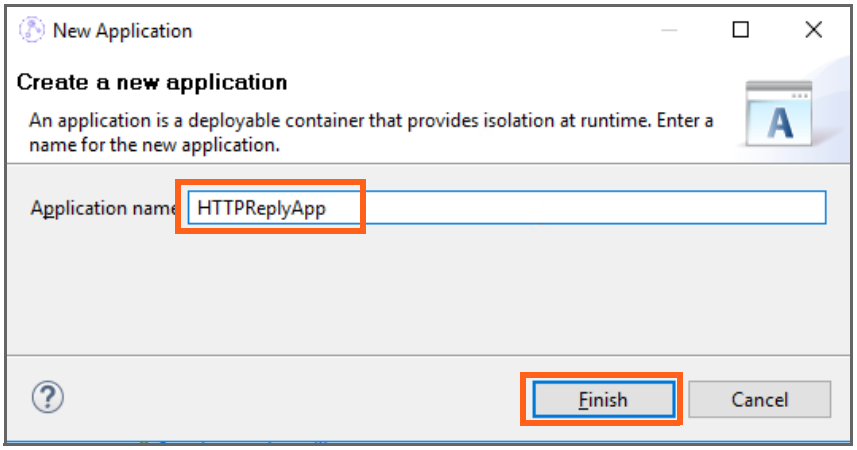
__ 2. Create an application that is named HTTPReplyApp.

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



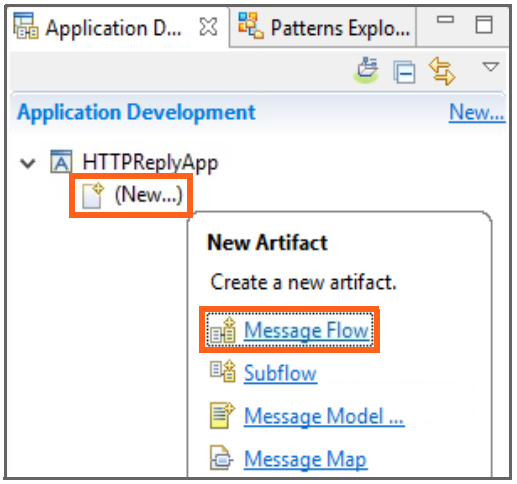
__ b. For the application name, enter `HTTPReplyApp`

__ c. Click **Finish**.



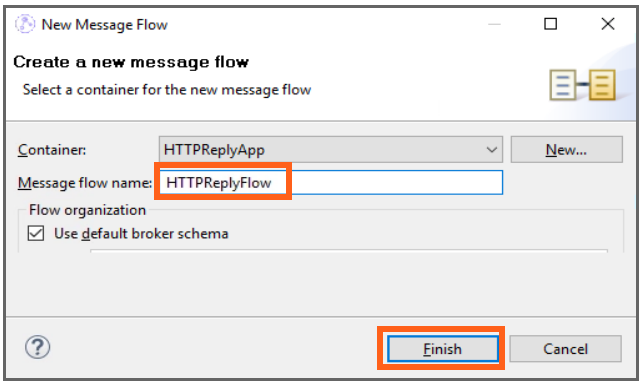
__ 3. Create a message flow that is named HTTPReplyFlow.

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

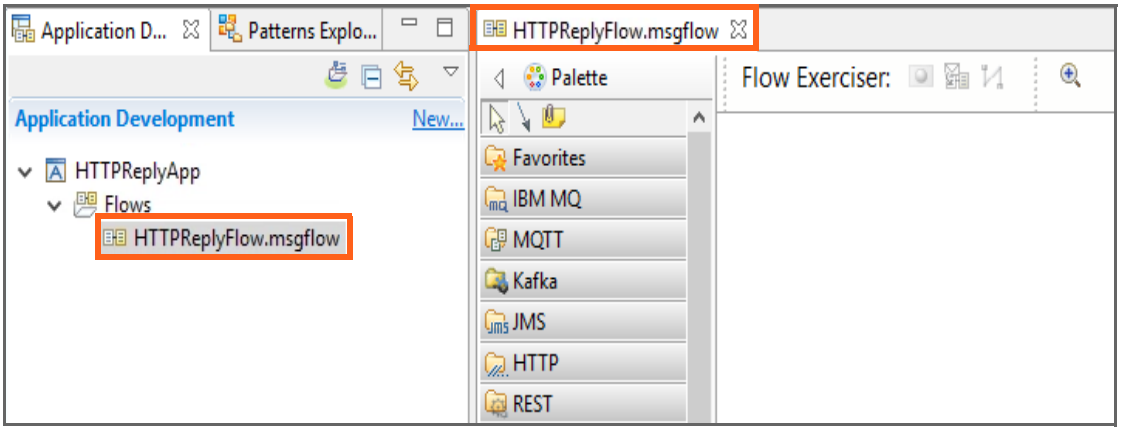


__ b. For the Message flow name type: `HTTPReplyFlow`

__ c. Click **Finish**.



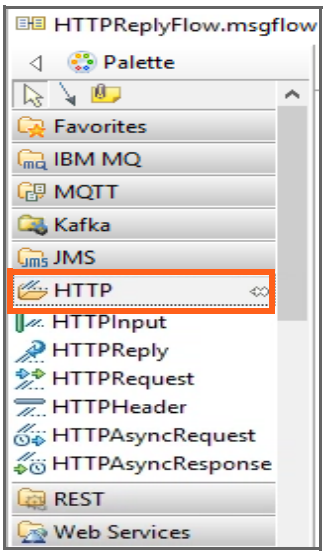
You should see the new message flow in the Application Development view under the Flows folder in the application. The **HTTPReplyFlow.msgflow** file is also open in the Message Flow editor.



__ 4. Add nodes to the message flow.

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.



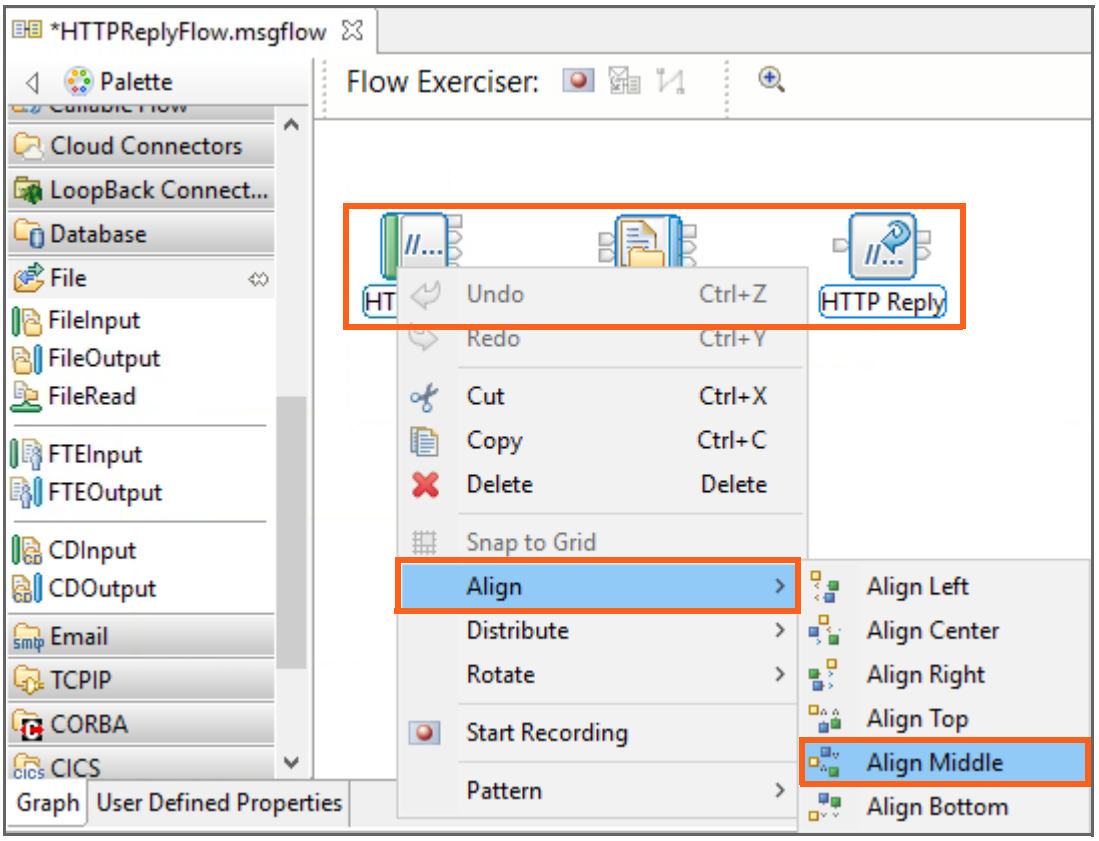
__ b. Click **HTTPInput** and then click the Message Flow editor view to add the node on the canvas.

__ 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.

__ f. Highlight all items and right-click the canvas and select **Align > Align Middle**.



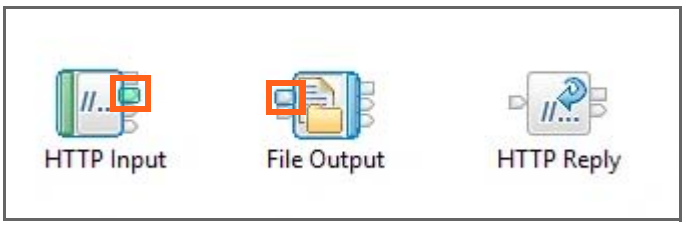
If your nodes are not distributed evenly, use the **Distribute > Distribute Horizontally** to create an even amount of space between them.

Information

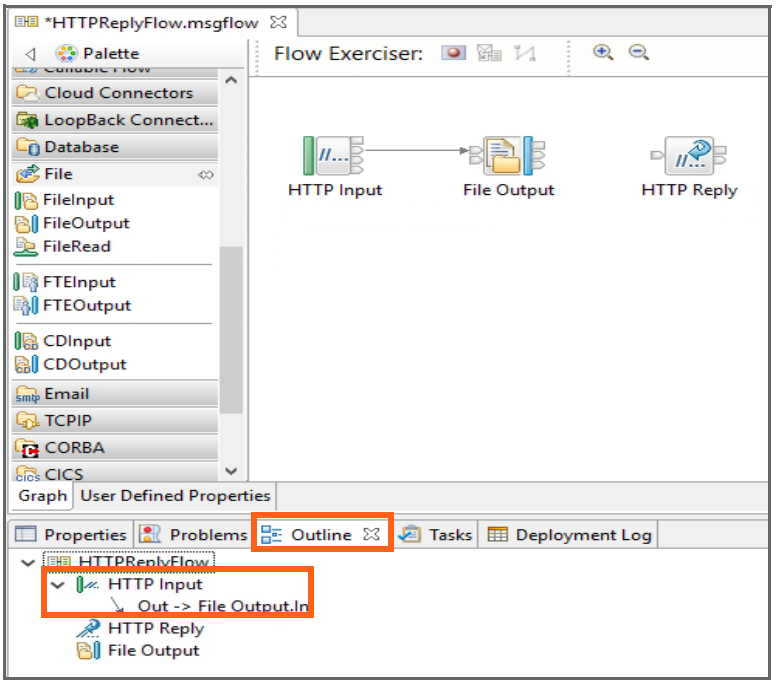
You can use the Align and Distribute functions to control the layout of the message flow. The Rotate function allows you to rotate the terminals to a different location on the node. These functions are helpful when the message flows contain a large number of nodes.

__ 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.

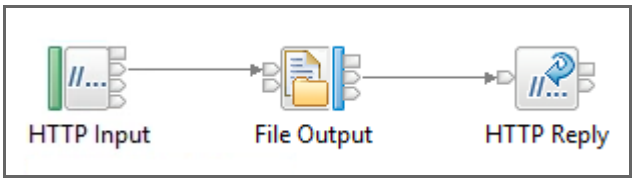


Hover your cursor over the **terminals** or use the **Outline** view to ensure that you wired the terminals correctly.



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

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

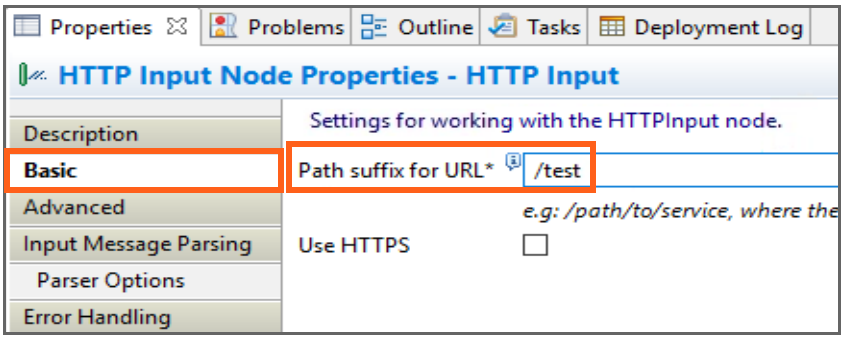


The Problems tab reports an error. This error is corrected in the next step.

__ 6. 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.

The error in the Problems tab is now gone.

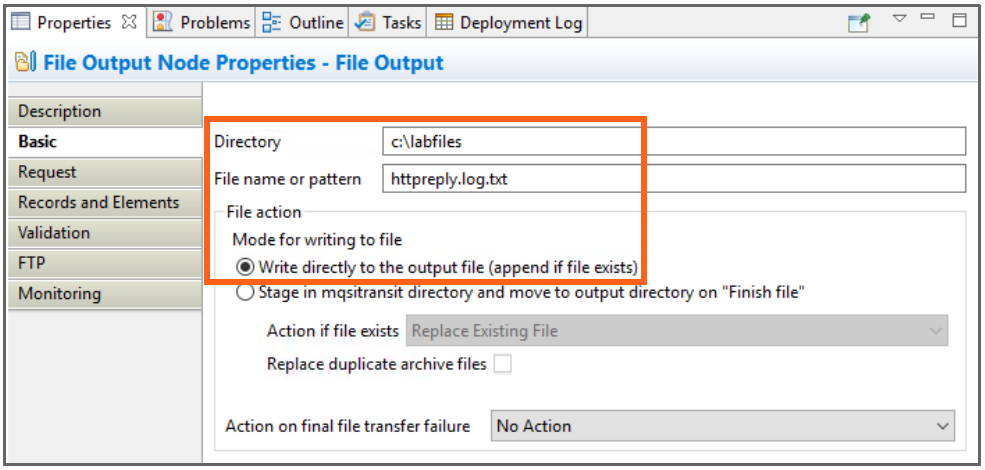
__ 7. Edit the Basic properties of the FileOutput node so that the file is written to the **httpreply.log.txt** file in the `c:\labfiles` directory.

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

__ b. For the **Directory** property, enter `c:\labfiles`

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

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



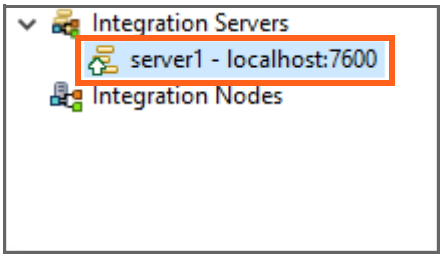
__ e. Save the message flow.

Part 2: Test the message flow

In this part of this exercise, you test the message flow by using a web browser and the IBM App Connect Enterprise Flow Exerciser.

__ 1. Start the Flow Exerciser.

__ a. Verify **server1** is running.

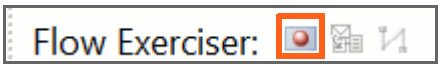


Troubleshooting

If server1 is not displayed in the Toolkit, right-click **Integration Servers**, and then select **Connect to an integration server**. For the connection details, enter `localhost` for the host name and `7600` for the port of the integration server.

If the server is not running, repeat steps **Part 1: Start the integration server** in Exercise 1.

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

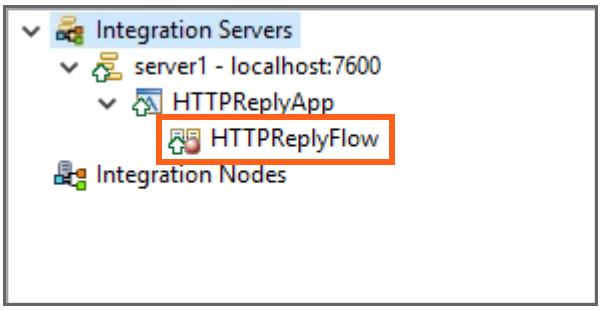


__ c. Click **OK** in the Confirm box.

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

The **HTTPReplyApp** application is automatically deployed to the **server1** integration server.

__ e. In the Integration Explorer view, expand **server1** to display the contents. Verify that **HTTPReplyFlow** is in record mode (signified by the Recording icon).



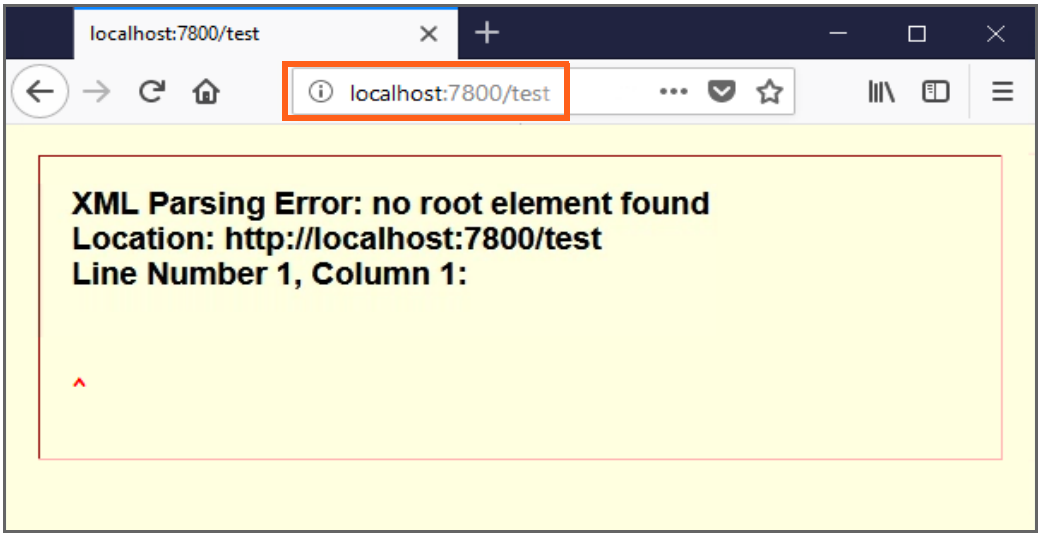
__ 2. Send a web request to the integration server

__ a. Start a web browser such as Firefox by double-clicking the icon on the desktop.

__ b. Enter the following URL in the address bar: `http://localhost:7800/test`

In the URL, `localhost` is the server location of the integration server. Port `7800` is the port on which the integration server receives HTTP connections. When you defined the

HTTPInput node properties, you specified that the message flow would use the `/test` URL.

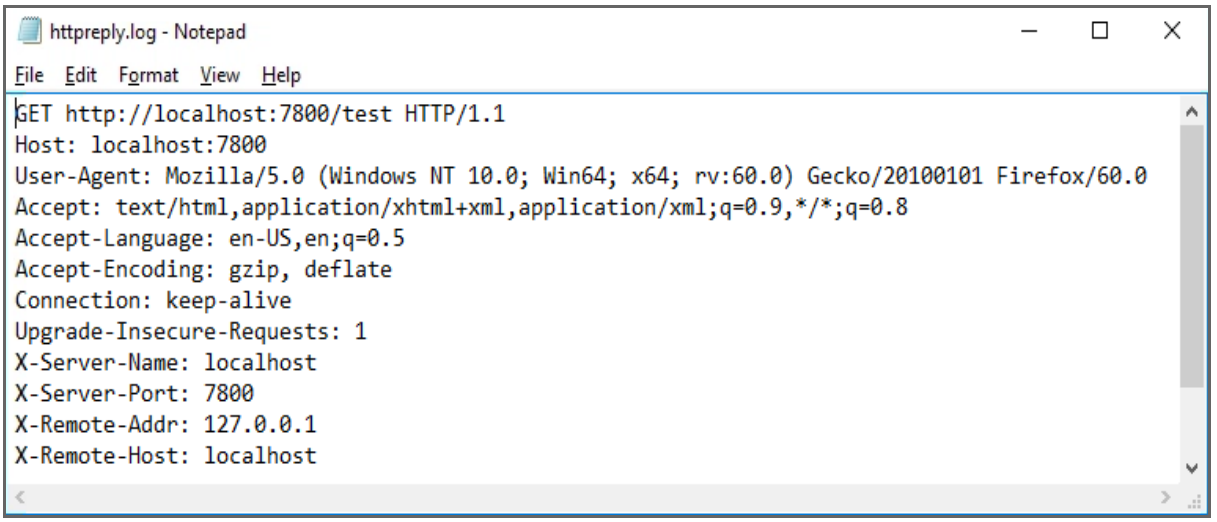


A response is not expected on the browser window from the message flow, but notice there is no error, so the request was successful.

__ 3. Check the output file httpreply.log.txt in the `C:\labfiles` directory.

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

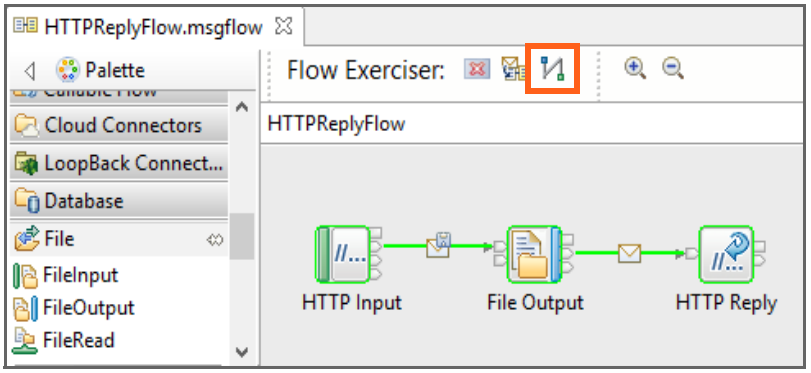
__ b. Double-click the **httpreply.log.txt** file in Windows Explorer to view the file in Notepad.



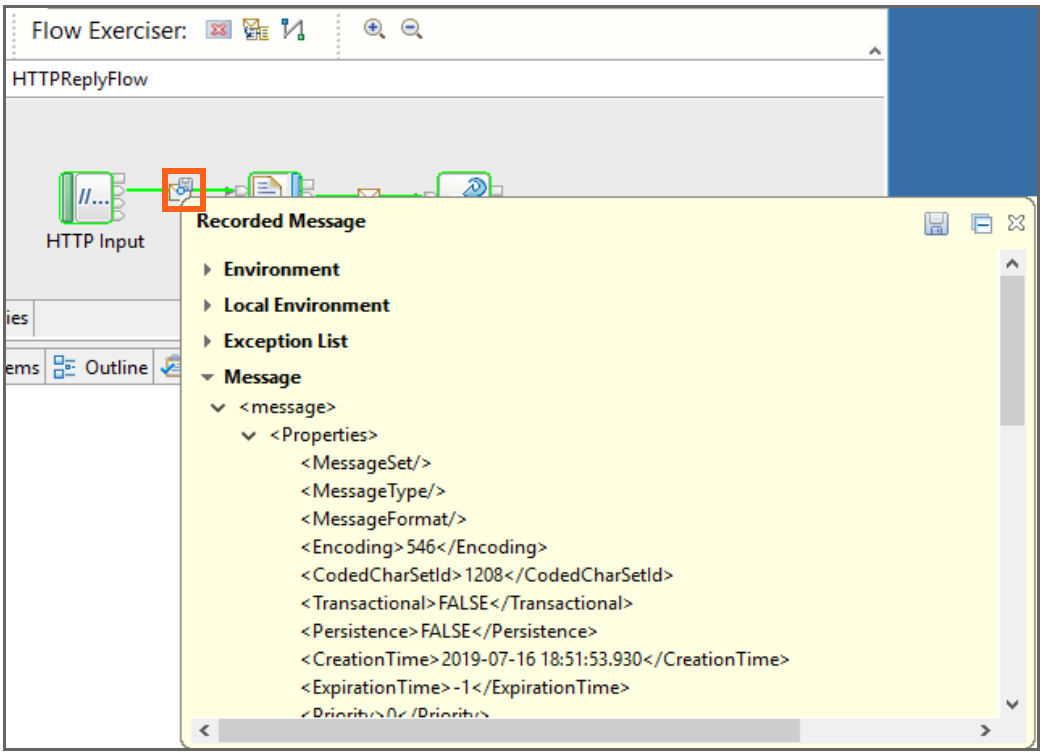
__ c. Close Notepad after reviewing the log.

__ 4. View the message in the Toolkit.

__ a. 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 the Flow exerciser captured.

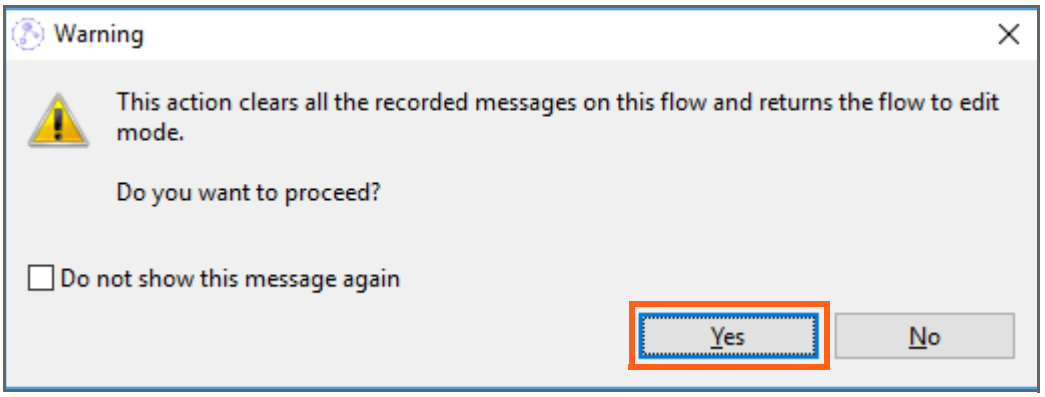


__ b. Click the message icons to examine the message before and after the FileOutput node processes it.



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

__ d. Click **Yes** in the Warning box.

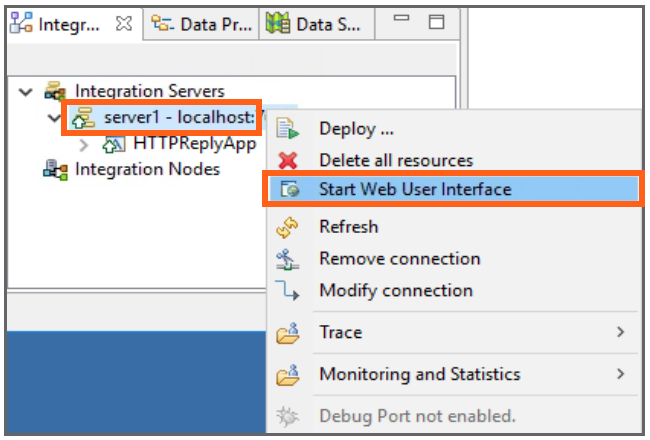


Part 3: Use the IBM App Connect Enterprise web user interface to view status

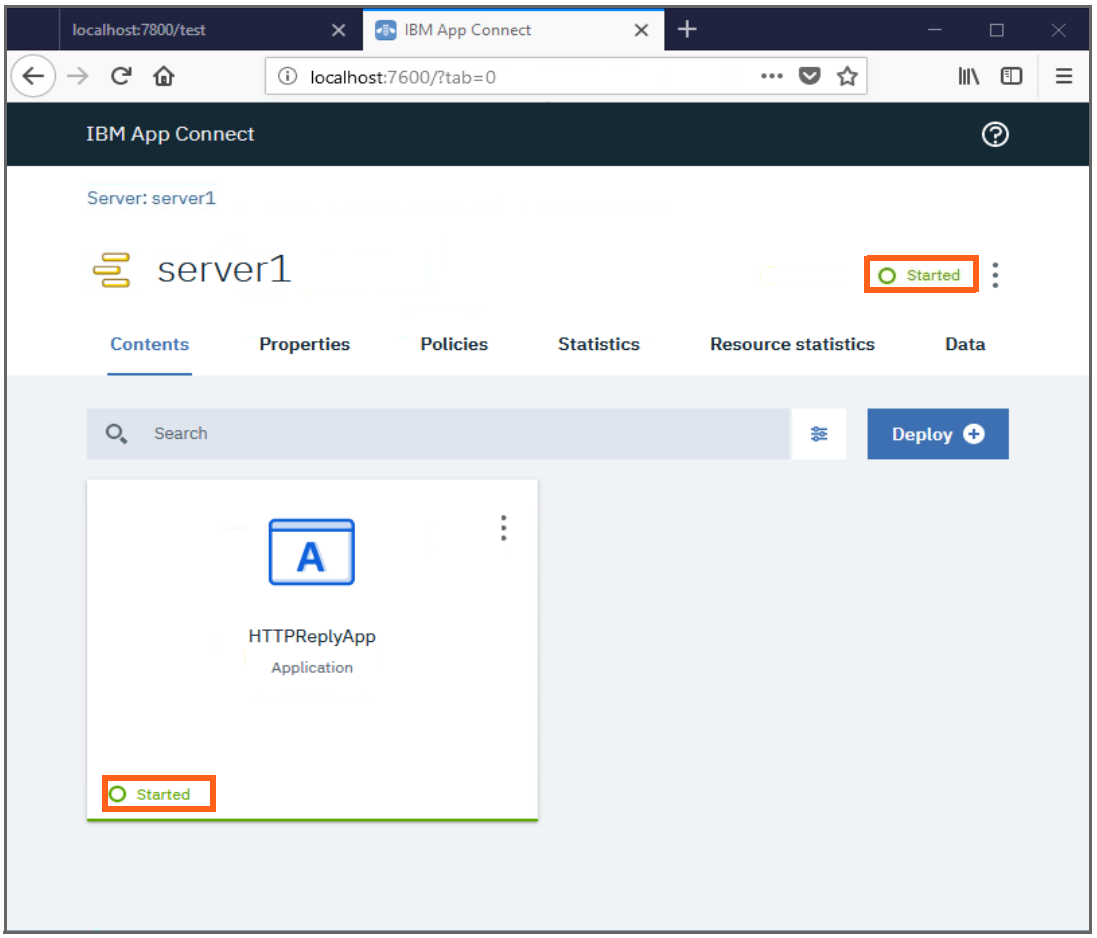
In this part of the exercise, you use the IBM App Connect Enterprise web console to check the status of the integration server and message flow application.

__ 1. View the server status in the web console.

__ a. Right-click **server1** in the Integration Explorer view and then select **Start Web User Interface** to start the IBM App Connect Enterprise web user interface in a web browser.



After a few seconds, the IBM App Connect Enterprise web user interface opens in Firefox.



All applications and their status are displayed. Currently, there is only the HTTPReplyApp application that is deployed and it is started. The server is also in a started state.

Information

You can use the **server.conf.yaml** configuration file to configure the port that is used to connect to the web user interface, and to secure the connection.

__ 2. View the properties of the message flow.

__ a. Click the **HTTPReplyApp** icon.

The HTTPReplyFlow is displayed with status as Started.

__ b. Click the **HTTPReplyFlow** icon.

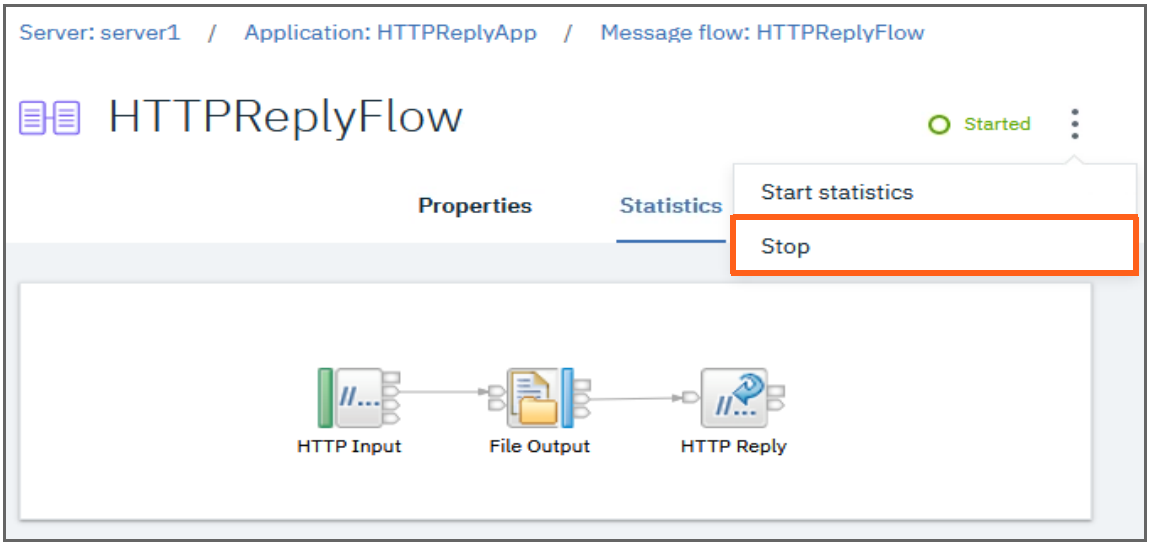
Properties and Statistics tabs are displayed with the Properties tab open.

__ c. Scroll down to view the properties.

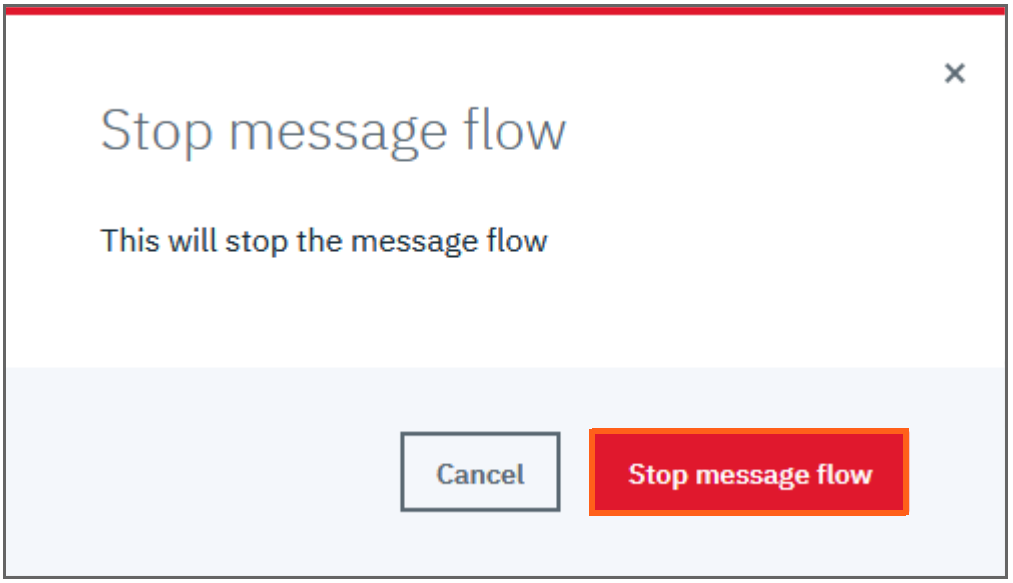
__ 3. Start and stop the message flow from the web user interface.

__ a. Click the ellipsis on the right.

__ b. Select **Stop**.



__ c. Click **Stop message flow**.



A notification is presented briefly stating the message flow was successfully stopped. The server status on the page is refreshed and now displays Stopped.

__ d. Click the ellipsis on the right.

__ e. Select **Start**.

__ f. Click **Start message flow**.

A notification is presented briefly stating the message flow was successfully started. The server status on the page is refreshed and now displays Started.

__ 4. Use the breadcrumb trail at the top to navigate among the server, application, and message flow. Use the **Properties** tabs to answer the following questions.

__ a. For the message flow, what is the start mode? Maintained

__ b. What is the location of the event log for server1? HTTPReplyApp/HTTPReplyFlow.msgflow

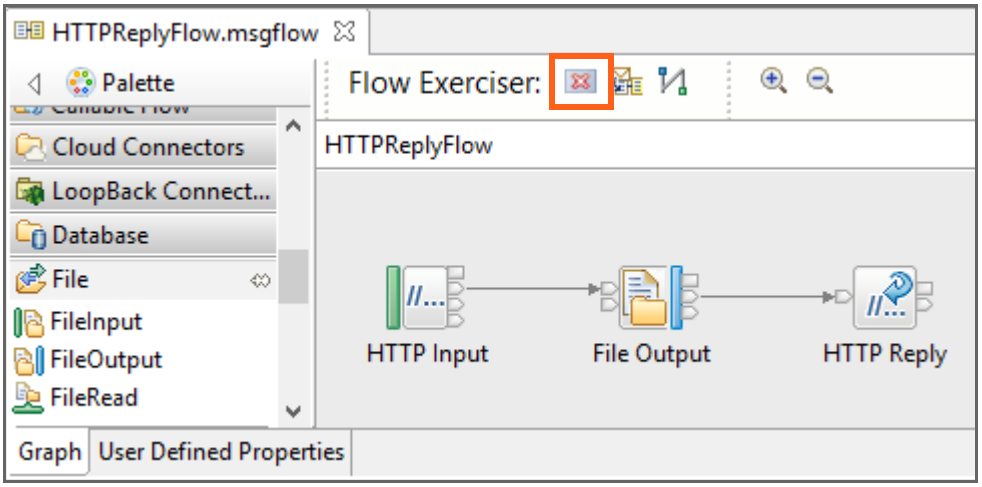
__ c. Are Trace Nodes enabled for server1? True

Part 4: Exercise clean-up

__ 1. Return to the edit mode and stop the Flow Editor

__ a. Open the Toolkit.

__ b. Click the **Return flow to edit mode** icon at the top of the Message Flow editor.



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,

__ c. Click **Yes**.

__ d. In the Integration Explorer view, expand the server to display the contents. Verify that **HTTPReplyApp** is no longer in record mode.

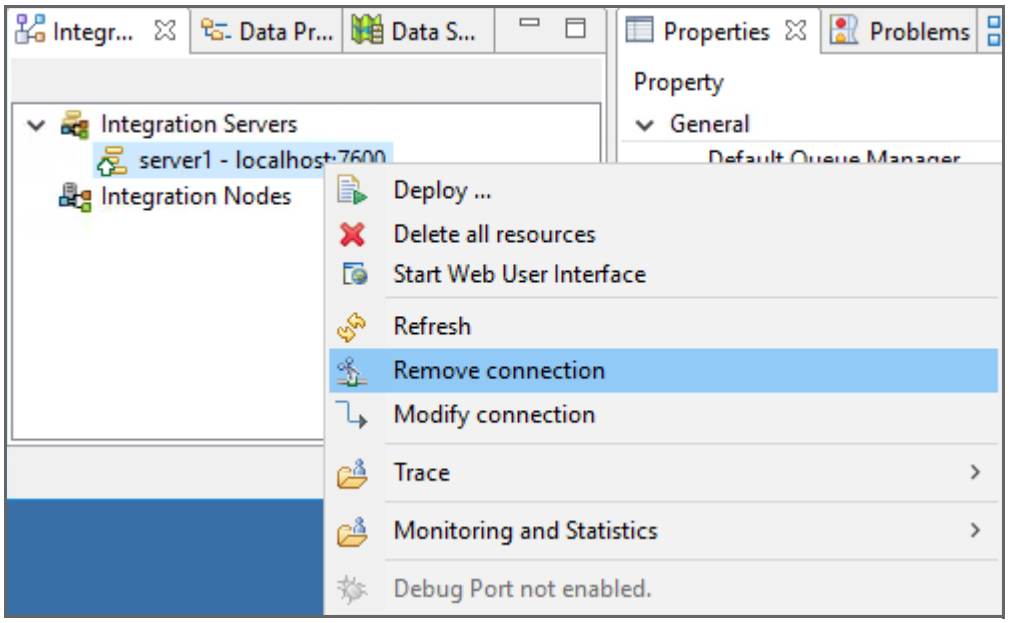
__ e. Close the **HTTPReplyFlow.msgflow** message flow in the Message flow editor.

__ f. Right-click **server1** and then click **Delete all resources**.

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

HTTPReplyApp is removed from server1.

__ h. Right-click **server1** and then click **Remove connection**.



The server is removed.

__ i. Close the IBM App Connect Enterprise web user interface.

__ j. Close the IBM App Connect Enterprise Console.

__ k. Close Windows Explorer.

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 IBM App Connect Enterprise Flow exerciser.

In the third part of this exercise, you used the IBM App Connect Enterprise web user interface to check the status of the integration server and message flow application. You also started and stopped a message flow.

Having completed this exercise, you should be able to:

- Create a message flow application
- Use the IBM App Connect Enterprise Flow Exerciser to test the message flow application
- Use the IBM App Connect Enterprise web user interface to check the status of an integration server and message flow application.

Exercise 3. Connecting to IBM MQ

Estimated time

01:30

Overview

In this exercise, you create an IBM MQ queue manager and IBM App Connect Enterprise integration nodes that use the same queue manager as the 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.

Objectives

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

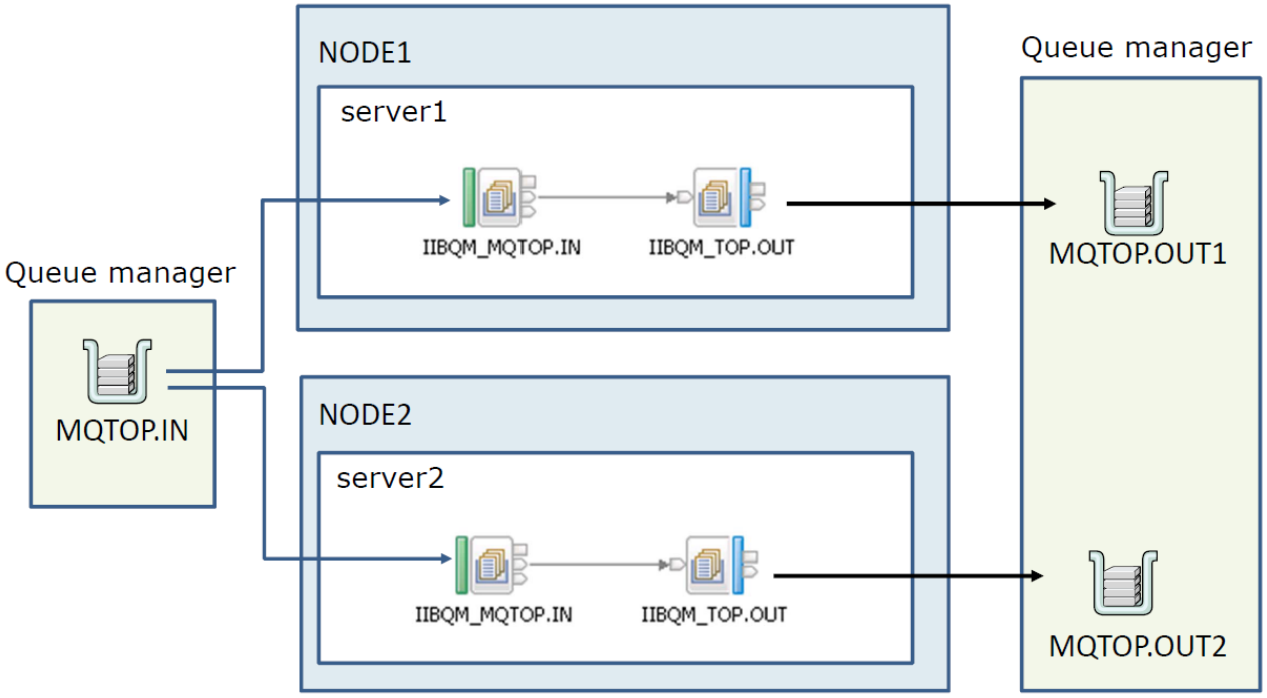
Introduction

You can develop and deploy many message flow applications with IBM App Connect Enterprise independently of IBM MQ; however, some IBM App Connect Enterprise 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 App Connect Enterprise, 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 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 that use the same queue manager as the 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 the default queue manager.

In the fourth part of the exercise, you test the message flow 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 the second node. 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 both nodes get messages from the MQTOP.IN queue.

Note

All IBM WebSphere MQ object names are case-sensitive.

Requirements

- A lab environment with the IBM App Connect Enterprise V11 Toolkit and IBM MQ V9
- The lab files in the `C:\labfiles\Lab03-MQ` directory
- The user `aceadmin` is a member of the “mqm” group

Exercise instructions

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

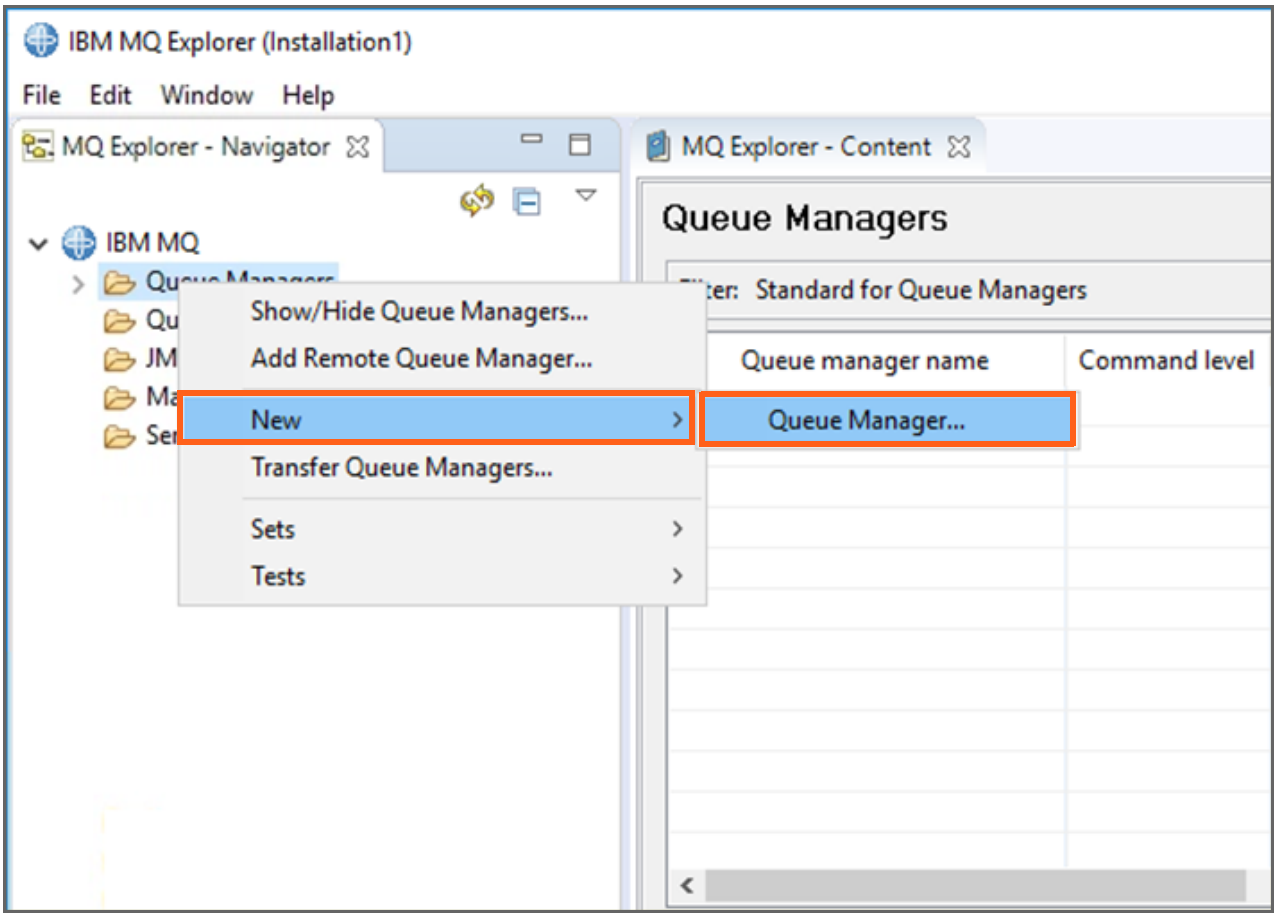
In this part of the exercise, you use IBM MQ Explorer to create a queue manager that is named ACEQM and the queues that are used in this exercise: DLQ, MQTOP.IN, MQTOP.OUT1, and MQTOP.OUT2.

__ 1. Create a queue manager that is named `ACEQM` with a dead-letter queue that is named `DLQ`

__ a. Double-click the MQExplorer icon on the desktop.

Alternatively, from the Windows **Start** menu, click **All Programs > IBM MQ > MQ Explorer (Installation1)**.

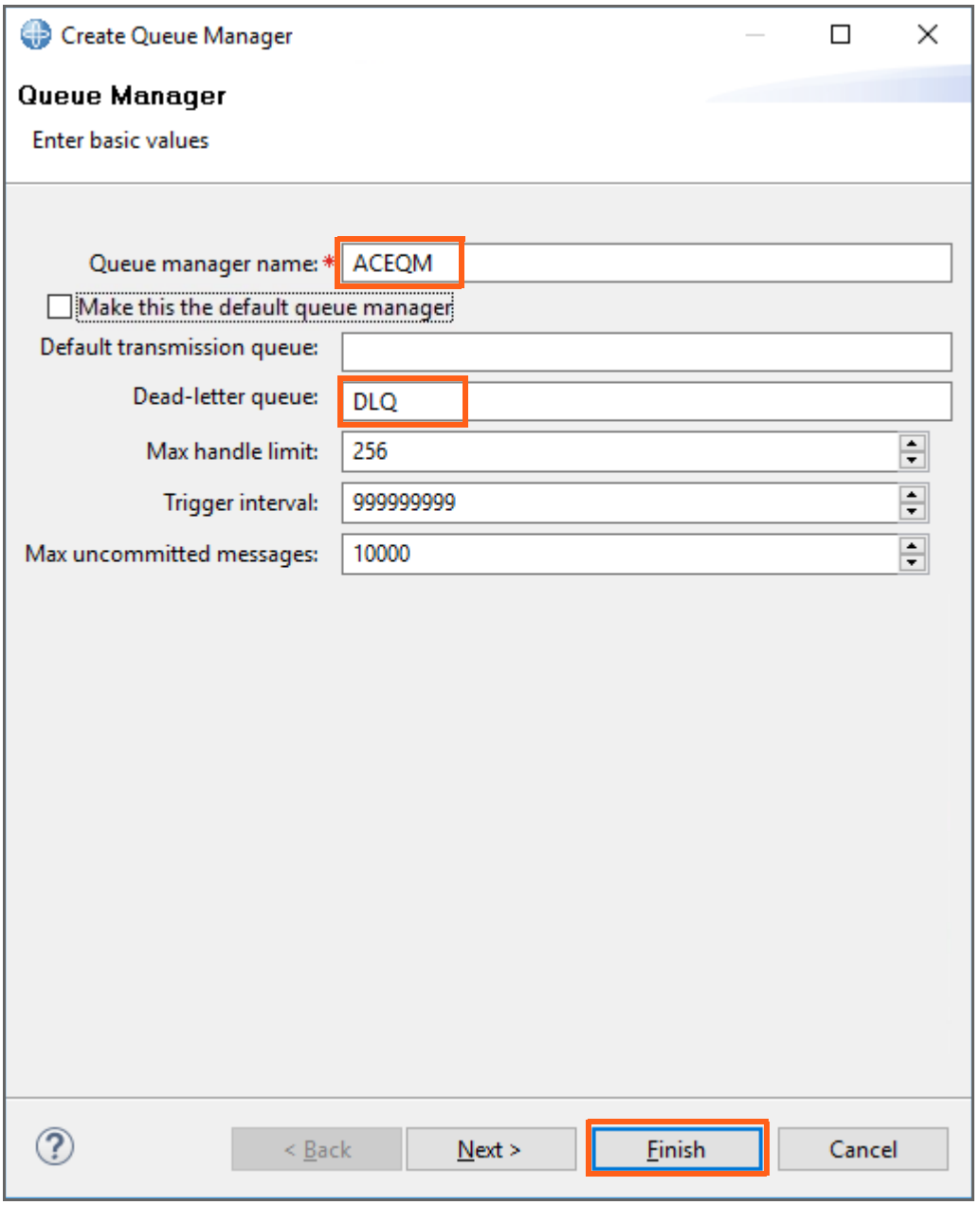
__ b. In the **MQ Explorer Navigator** view, right-click **Queue Managers** and then click **New > Queue Manager**.



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

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

__ e. Click **Finish**.



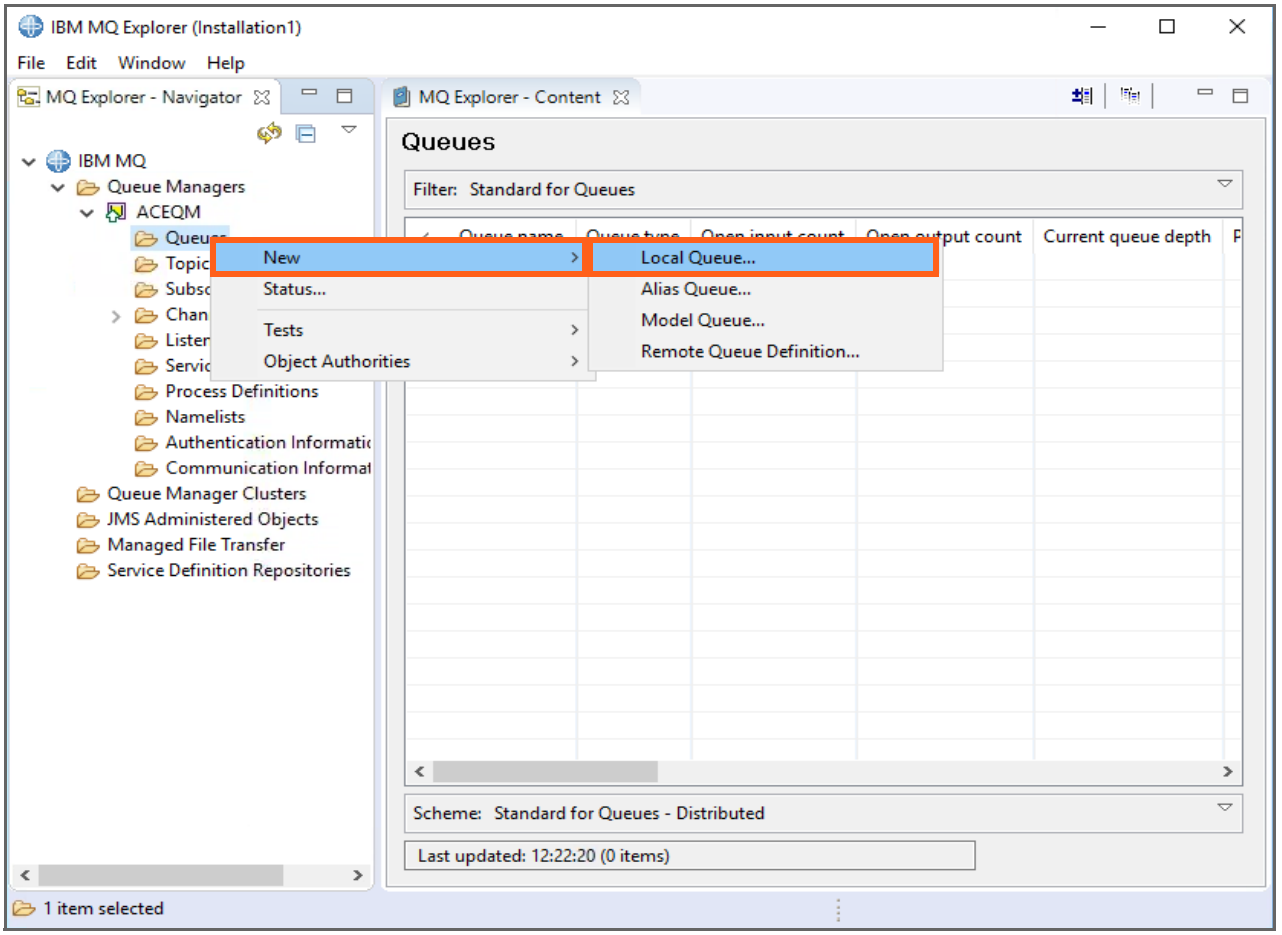
After a brief pause, the queue manager is created and started. A listener is also started on the default TCP port of 1414.

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

__ 2. Create a queue that is named `DLQ` for the dead-letter queue.

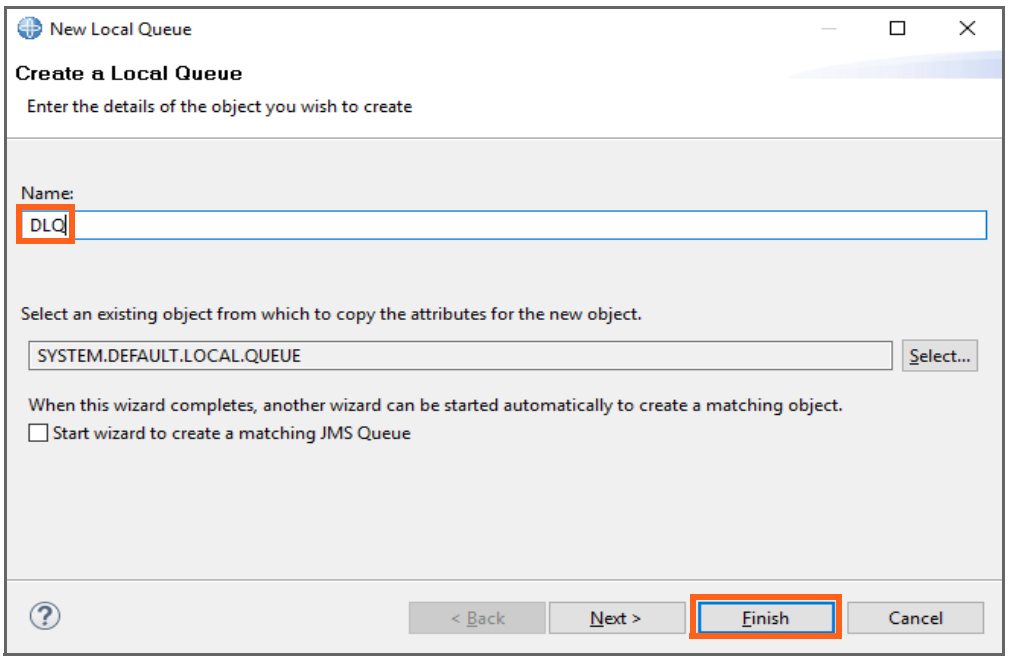
__ a. In the **MQ Explorer - Navigator** view, expand the **ACEQM** queue manager folder.

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



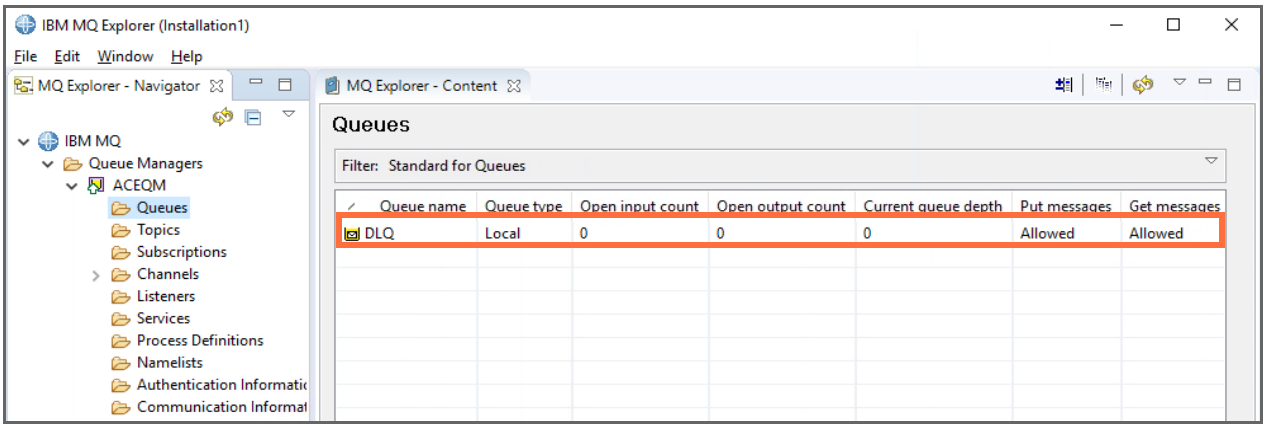
__ c. For the queue **Name**, type: **DLQ**

__ d. Click **Finish**.



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

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

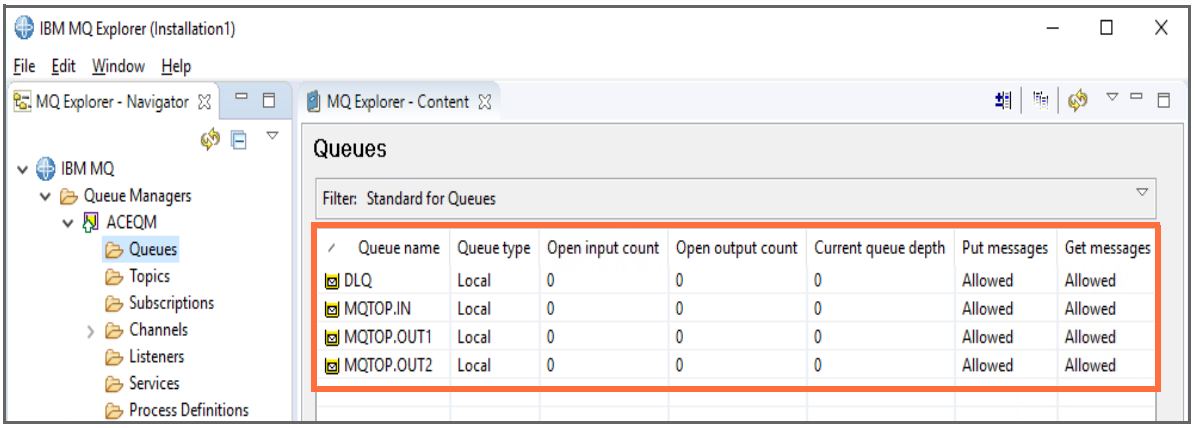


__ 3. Following the same procedure that you used to create the queue in Step 3, create the IBM MQ input queue and output queues that are used in this exercise.

__ 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`



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

Part 2: Create the integration nodes by using the Console

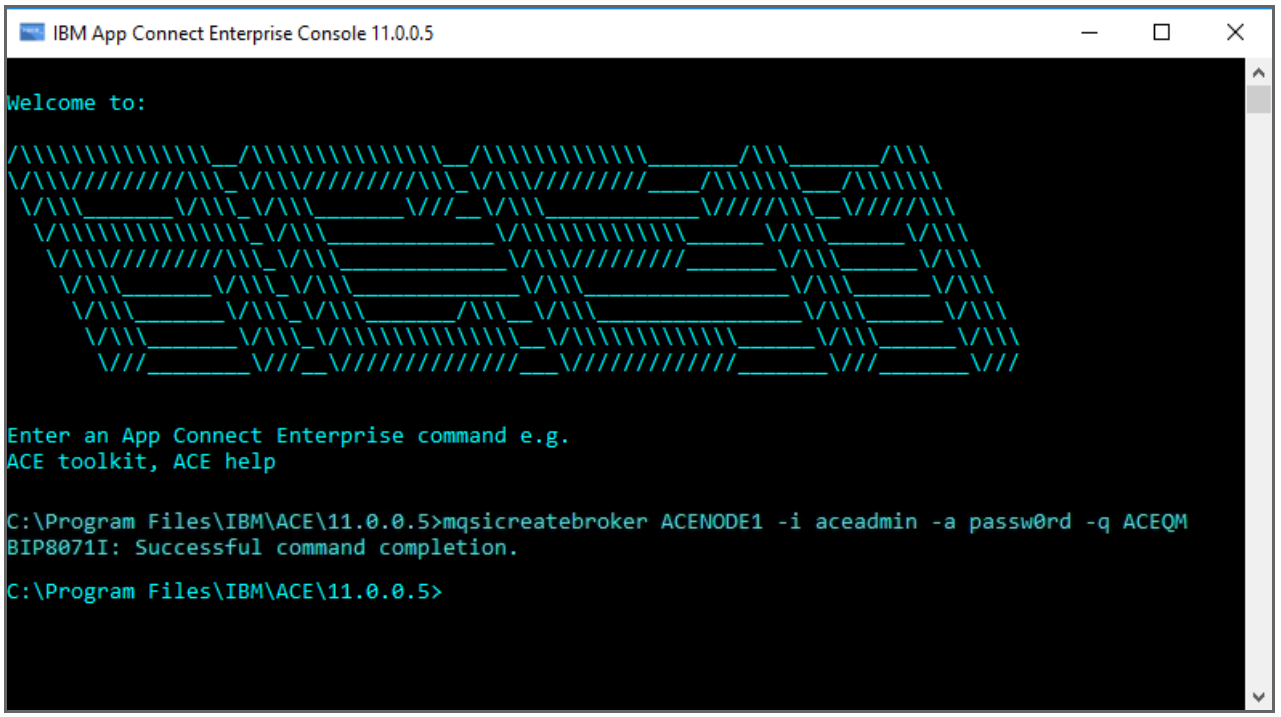
In this part of the exercise, you create the integration nodes by using the console.

__ 1. Create the integration nodes.

__ a. Open the IBM App Connect Enterprise Console.

__ b. Create the first integration node with the queue manager created in the last section by entering the following command in the App Connect Enterprise Console:

```
mqsicreatebroker ACENODE1 -i aceadmin -a passw0rd -q ACEQM
```



__ c. Create the second integration node by entering the following command:

```
mqsicreatebroker ACENODE2 -i aceadmin -a passw0rd -q ACEQM
```

Information

When you create your first integration node, the web user interface uses the default port **4414**, and for each integration node created afterward, the port number increments automatically by one. You can change the port number to be used by editing the integration node's **node.conf.yaml** file.

Part 3: Create the integration servers by using the Toolkit

In this part of the exercise, you create the integration servers for the two nodes just created.

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab03`.

__ a. Start the IBM App Connect Enterprise Toolkit if it is not already running

__ b. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

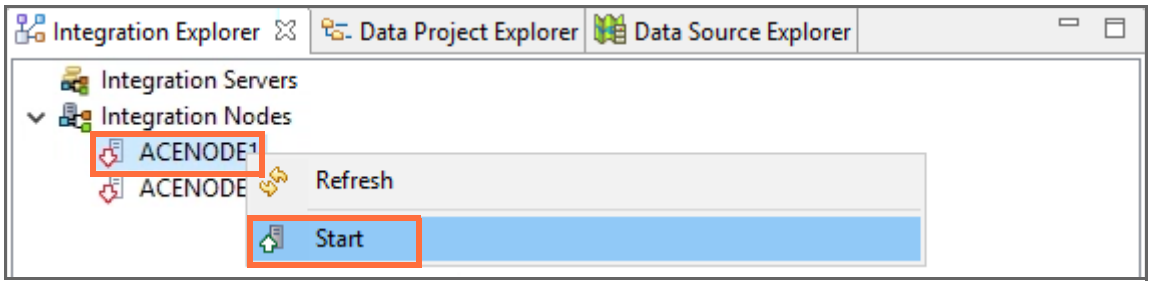
__ c. For the **Workspace**, enter `C:\Workspace\Lab03`

__ d. Click **OK**. After a brief pause, the IBM App Connect Enterprise Toolkit restarts in the new workspace.

__ e. Close the Welcome window to go to the Application Development perspective.

__ 2. Use the IBM App Connect Enterprise Toolkit to start the integration nodes.

__ a. In the Integration Explorer view, right-click **ACENODE1** and select **start**.

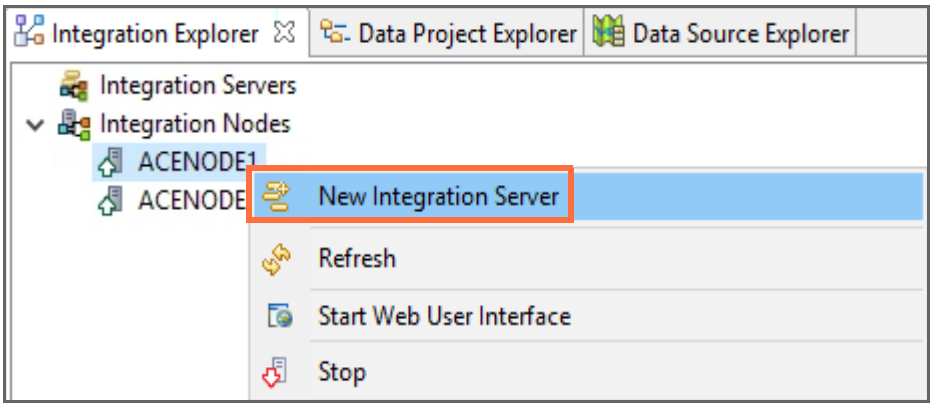


After a moment, the status of the server changes from stopped (downward facing red arrow) to started (upward facing green arrow).

__ b. Right-click **ACENODE2** and select **start**.

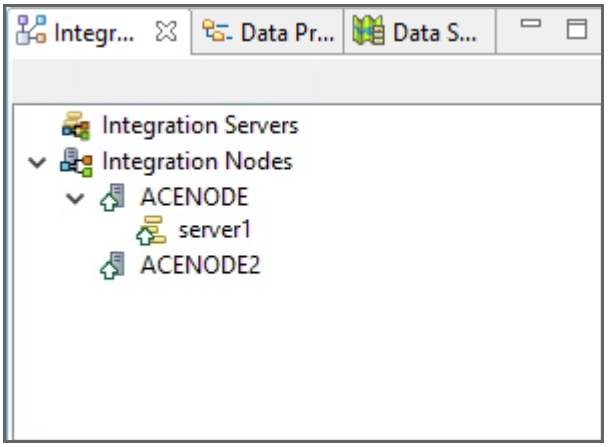
__ 3. Add the servers to the nodes.

__ a. Add server1 to ACENODE1 by right-clicking **ACENODE1** and selecting **New Integration Server**.



__ b. Enter `server1` for the name.

You should now have a node (ACENODE1) with a server (server1) running in it.



__ c. Add server2 to ACENODE2 by right-clicking **ACENODE2** and selecting **New Integration Server**

__ d. Enter `server2` for the name

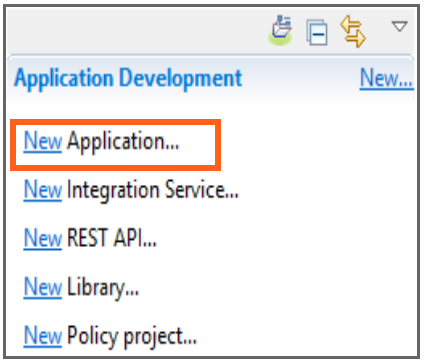
You should now have two nodes with one server running in each node.

Part 4: 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 App Connect Enterprise Toolkit, create an application that is named `MQTOP`.

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

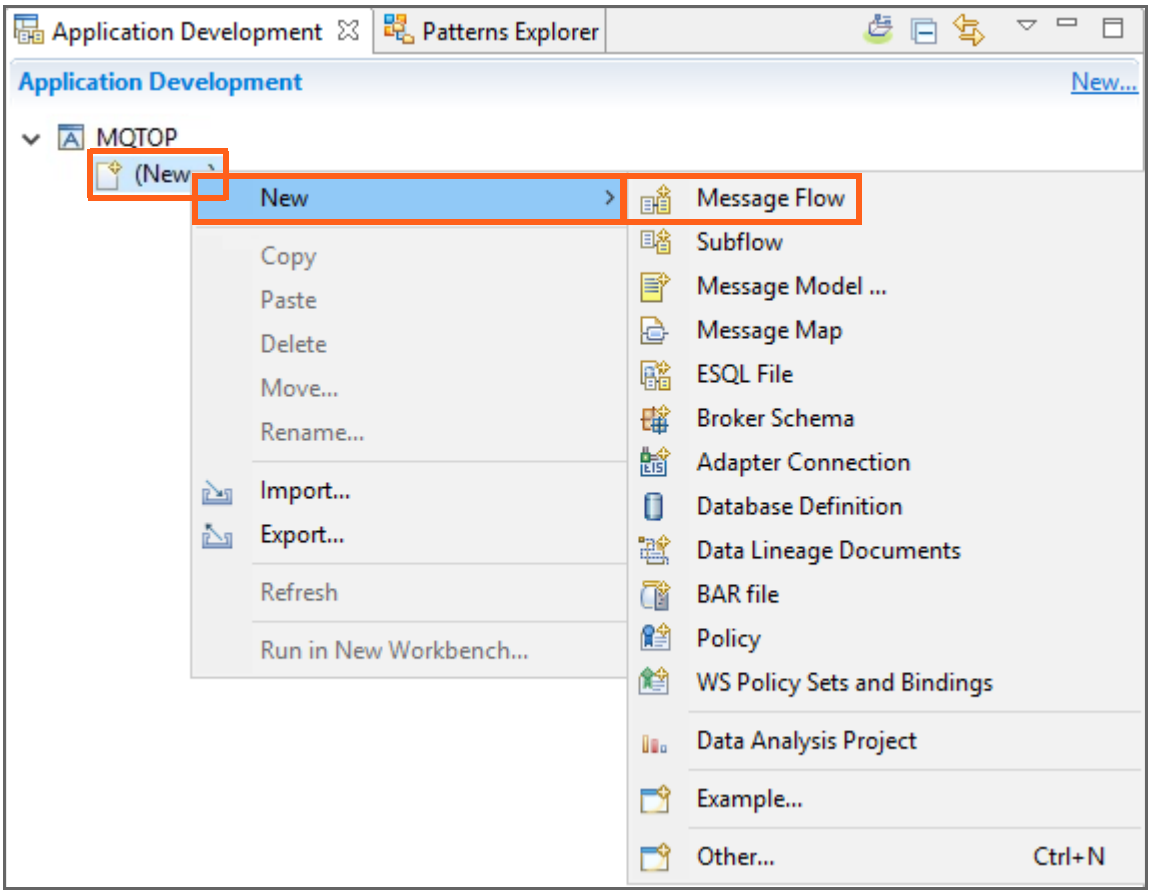


__ b. For the application name, type: `MQTOP`

__ c. Click Finish.

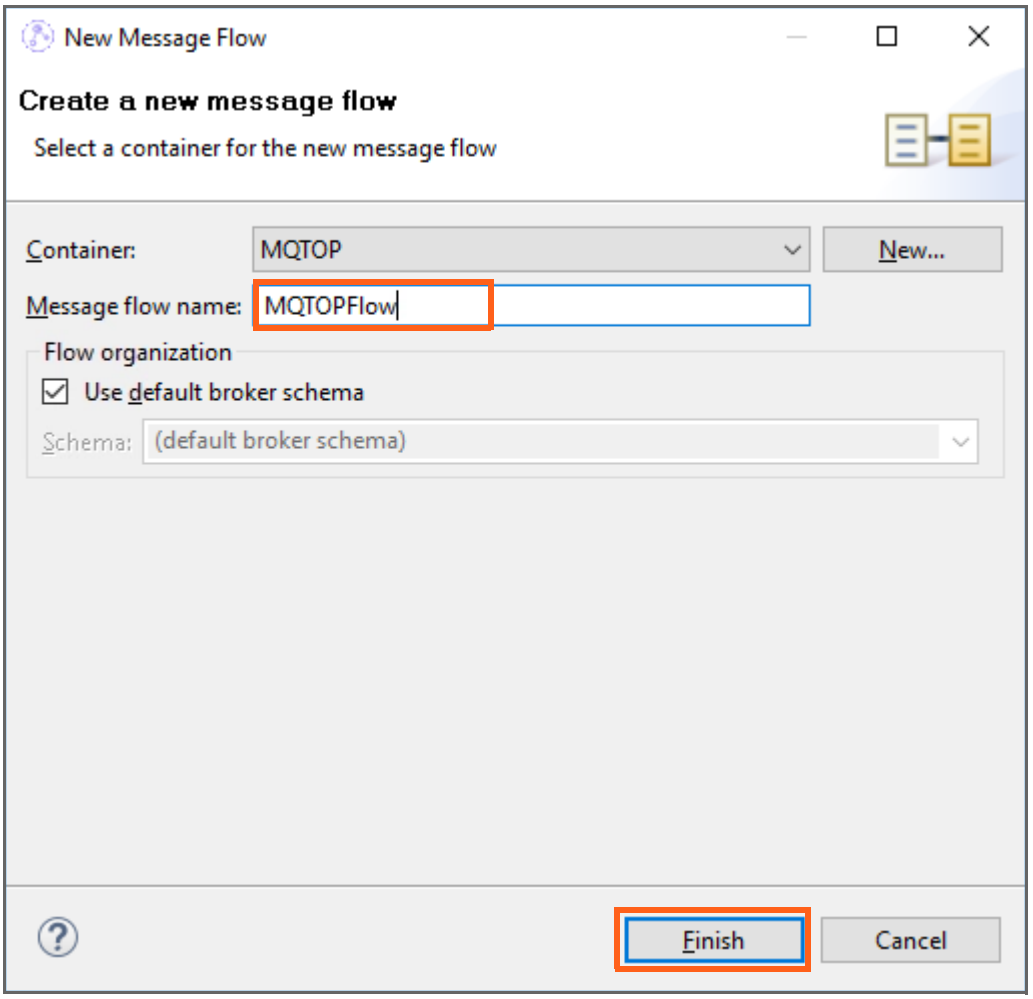
__ 2. In the MQTOP application, create a message flow that is named `MQTOPFlow`.

__ a. Right-click **(New…)** under the MQTOP folder in the Application Development view, and then click **New > Message Flow**.



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

__ c. Click Finish.

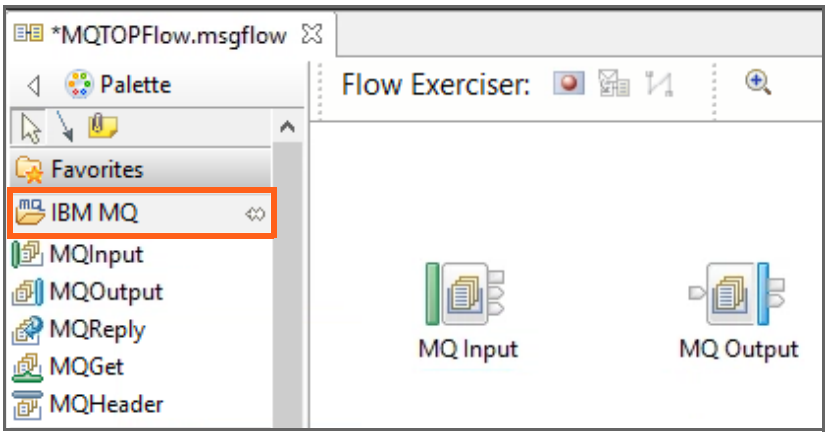


__ 3. Add the nodes to the Message Flow editor canvas.

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

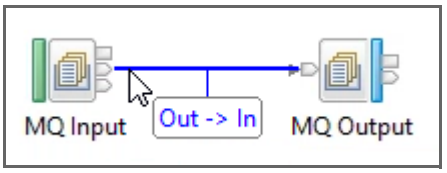
__ b. Drag an **MQInput** node to the Message Flow editor canvas.

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



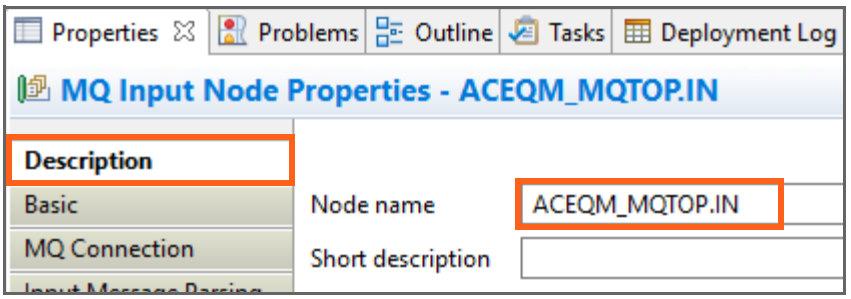
__ d. Wire the **Out** terminal on the MQ Input node to the **In** terminal on the MQ Output node.

__ e. Verify correct wiring by hovering the mouse over the wire.

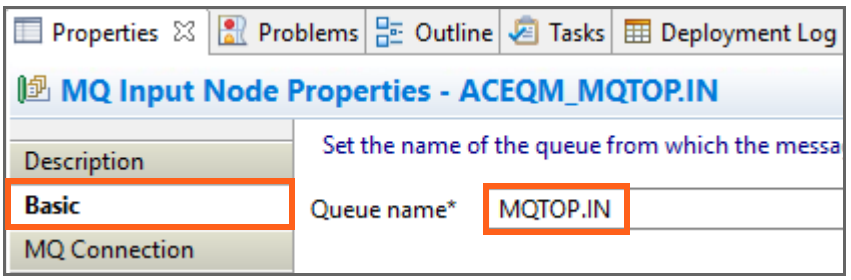


__ 4. Configure properties for the MQ Input node.

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



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



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

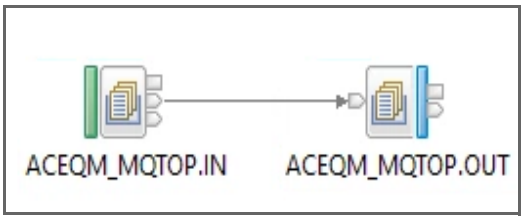
__ 5. Configure the node properties for the MQ Output node.

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

__ d. Save the flow.



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

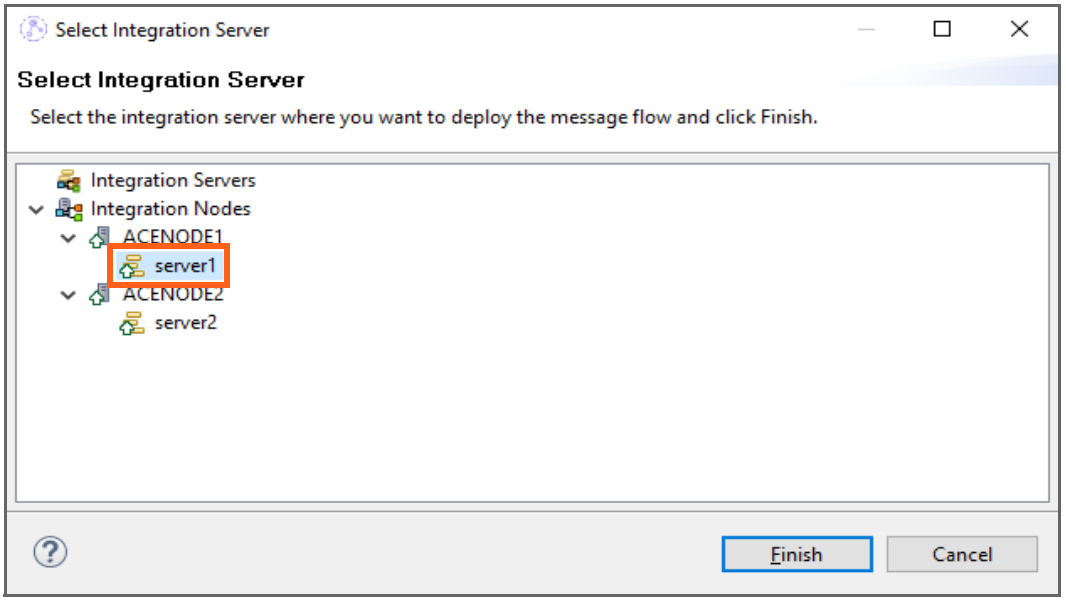
Part 5: Test the message flow

In this part of the exercise, you test the message flow with ACENODE1. You put messages on the queue by using 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. Start the Flow exerciser

__ a. Click the **Start flow exerciser** icon at the top of the Message Flow editor.

__ b. In the **Select Integration Server** window, click **server1** under **ACENODE1**.



__ c. Click **Finish**.

__ d. Click **OK** in the Confirm box.

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.

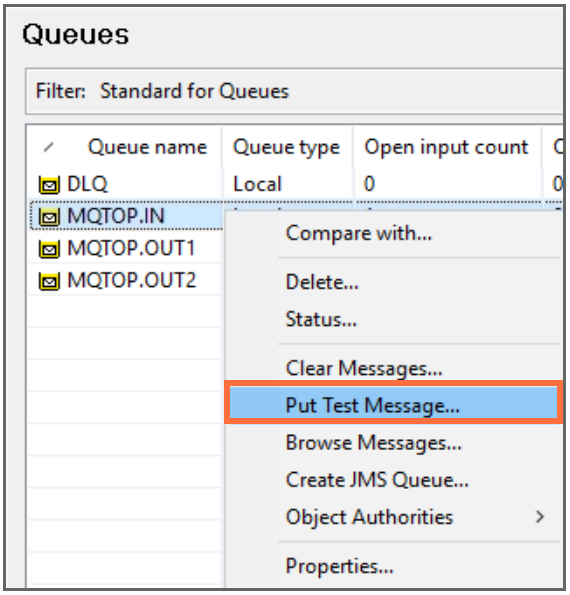
__ e. After a few seconds, you will see a window that indicates that the Flow exerciser is ready to record a message. When you see this information window, click **Close**.

__ f. In the Integration Explorer view, expand **ACENODE1** > **server1** to display the contents. Verify that **MQTOPFlow** is in record mode (signified by the Recording icon)

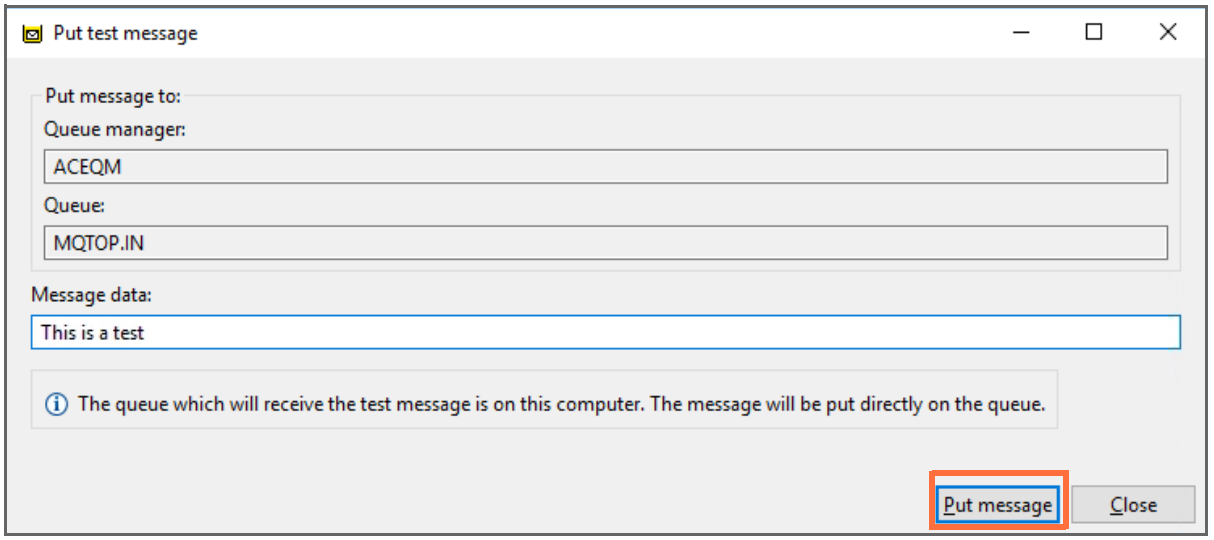
__ 2. Put a message on the input queue by using IBM MQ Explorer.

__ a. Return to IBM MQ Explorer.

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



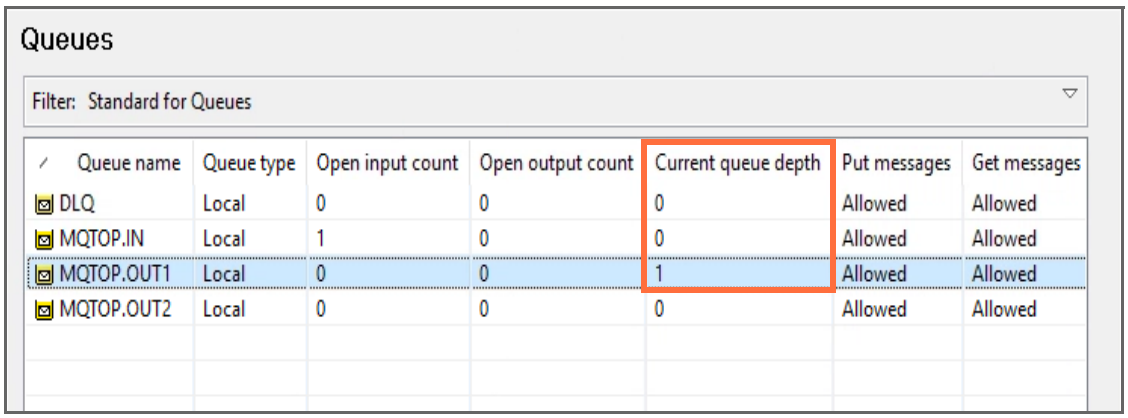
__ c. Type a test message in the **Message data** field and then click **Put message**.



__ d. Click **Close**.

In the IBM MQ Explorer **Queues** view, you should see the **Current queue depth** for the input queue MQTOP.IN increment to 1. After a brief pause, you should see **Current**

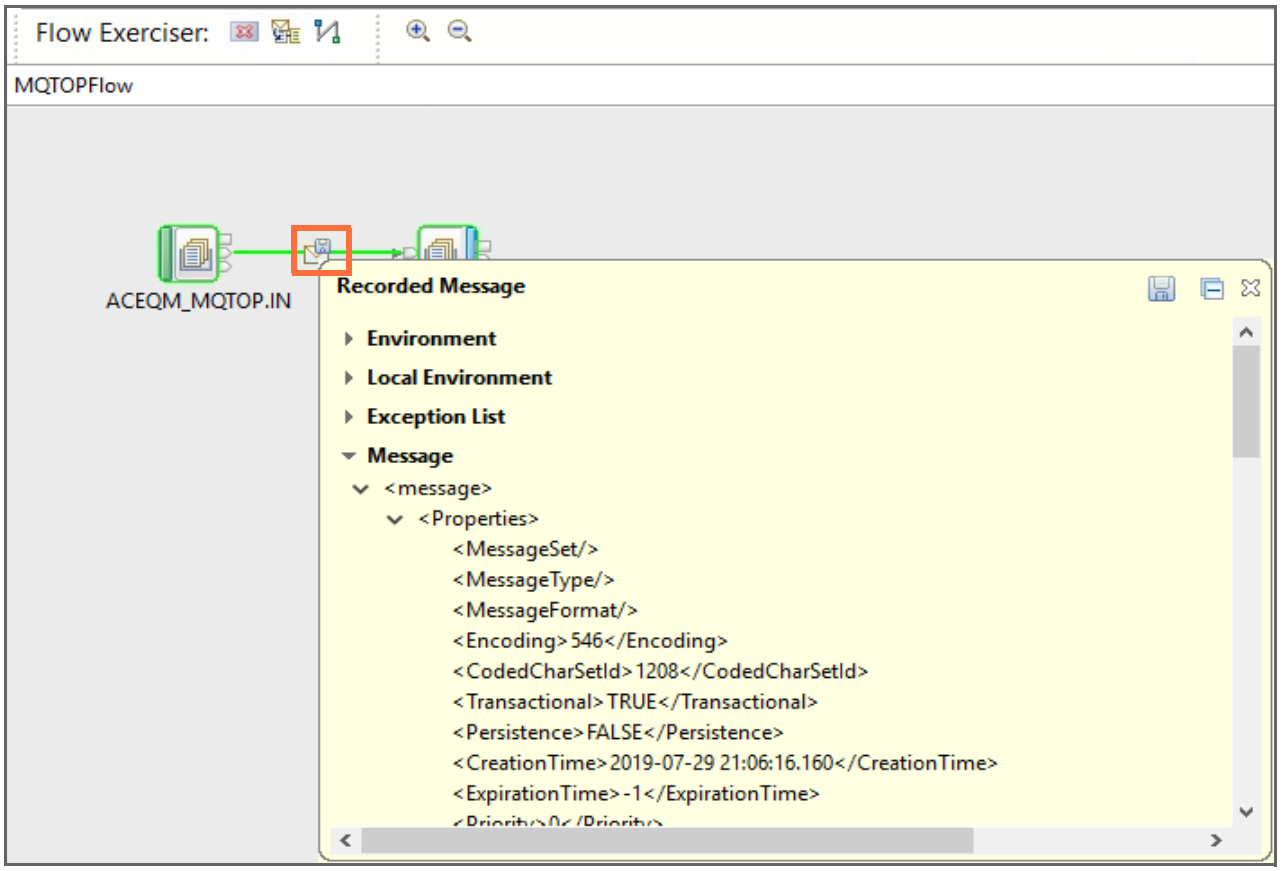
queue depth on MQTOP.IN decrement and return to zero. You should also see the **Current queue depth** on the output queue MQTOP.OUT1 increment to 1.



__ 3. Verify the message flow processed

__ a. Return to the IBM App Connect Enterprise Toolkit and click the Flow exerciser **View Path** icon.

__ b. Click the message icon to display the contents of the message.

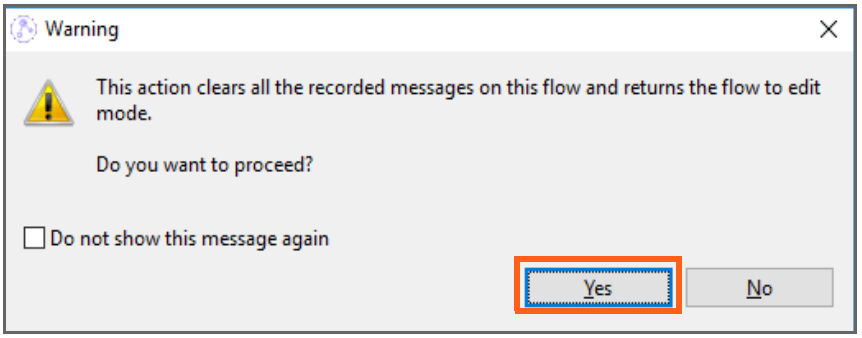


__ c. After you review the input message, close the message window.

__ d. Click the **Return flow to edit mode** icon at the top of the Message Flow editor.

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,

__ e. Click **Yes**.



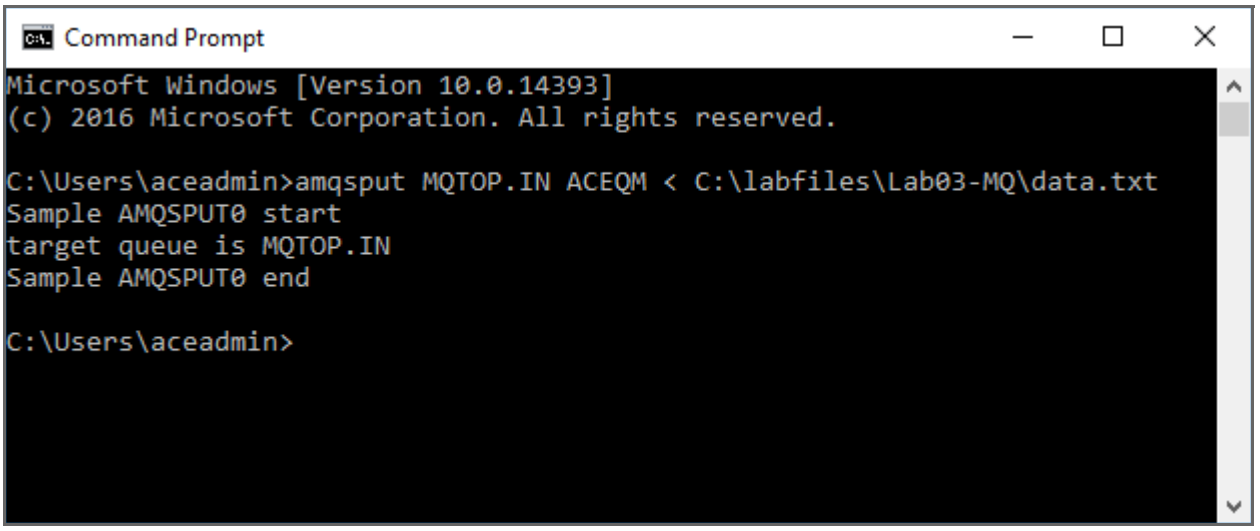
__ f. In the Integration Explorer view, expand server1 to display the contents. Verify that MQTOPFlow is no longer in record mode

__ 4. 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.

IBM MQ comes with a sample application called **amqsput**. This application puts a message to a predefined queue.

__ a. Open a Command Prompt window.

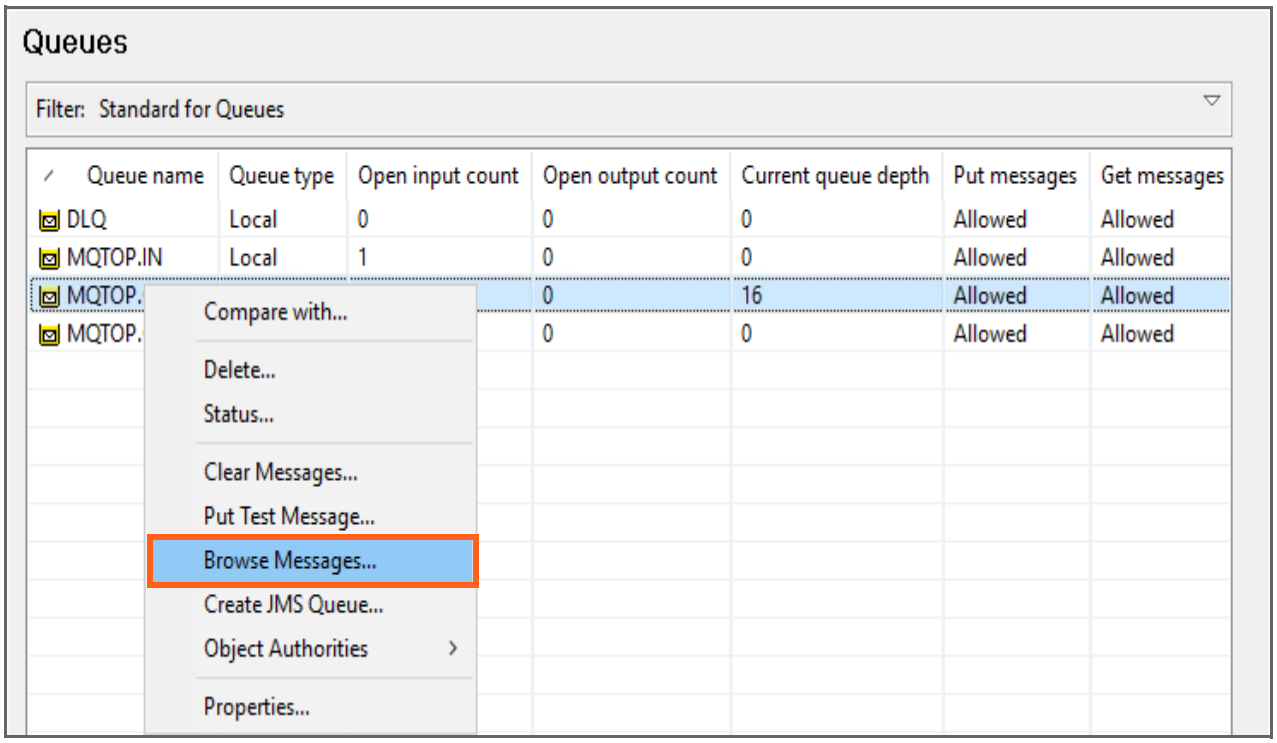
__ b. Type: `amqsput MQTOP.IN ACEQM < C:\labfiles\Lab03-MQ\data.txt`



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

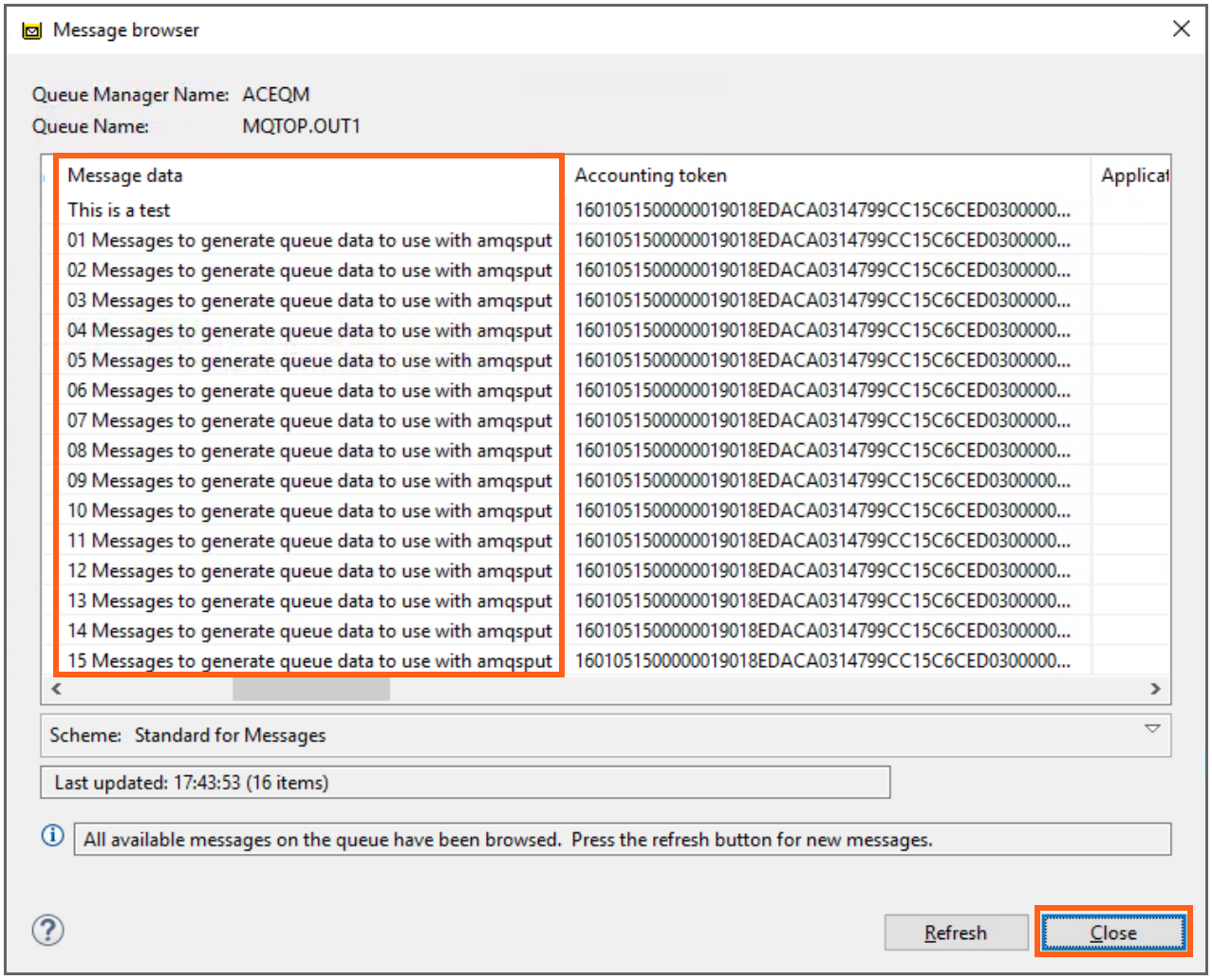
__ 5. By using IBM MQ Explorer, verify that you now have 16 messages on the MQTOP.OUT1 queue (one message from Step 2 and the 15 messages from Step 4).

__ 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.

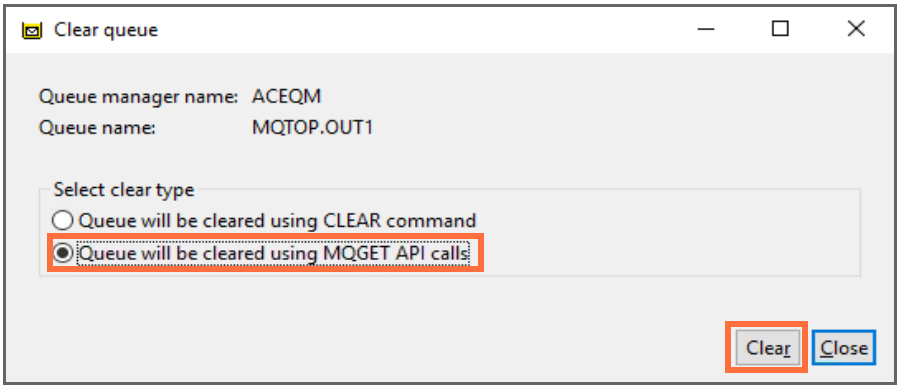


__ 6. Clear the messages on the MQTOP.OUT1 queue.

__ 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.

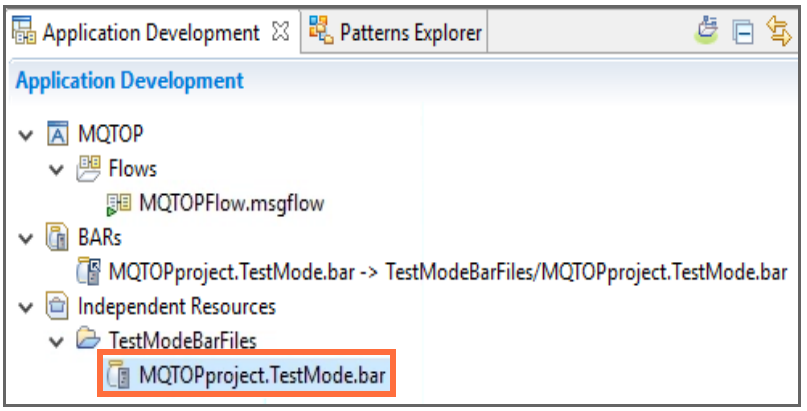
Part 6: Modify the BAR file and manually deploy the message flow

In Part 5 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 ACENODE2. You then use the IBM MQ `amqsput` sample program to put multiple messages to the input queue.

__ 1. Modify the BAR file.

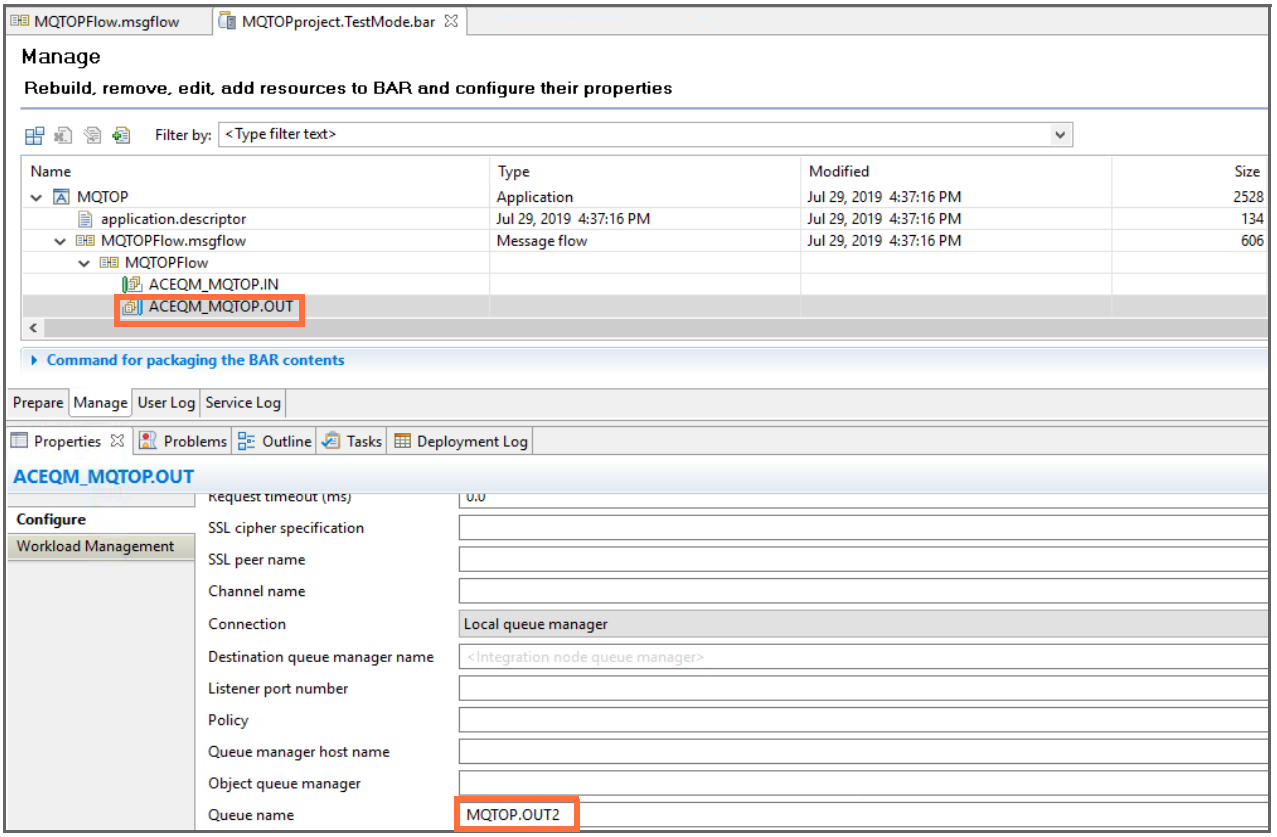
__ a. In the IBM App Connect Enterprise Toolkit Application Development pane, double-click the **MQTOPProject.TestMode.bar** file under **Independent Resources > TestModeBarFiles** to open it in the BAR File editor.



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

__ c. Select the **ACEQM_TOP.OUT** node in the BAR File editor to show the node **Properties** view.

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



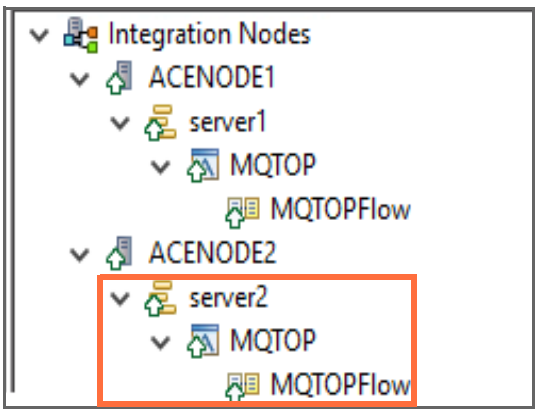
__ e. Save the BAR file.

__ 2. Deploy the BAR file.

__ a. Expand **ACENODE2** in the Integration Explorer view.

__ b. Deploy the updated BAR file to **server2** on ACENODE2 by dragging it from the Application Development view and dropping it onto **server2** in the **Integration Explorer view**.

__ c. Verify that the BAR file was deployed successfully by checking the **ACENODE2 > server2 > MQTOP > MQTOPFlow**.



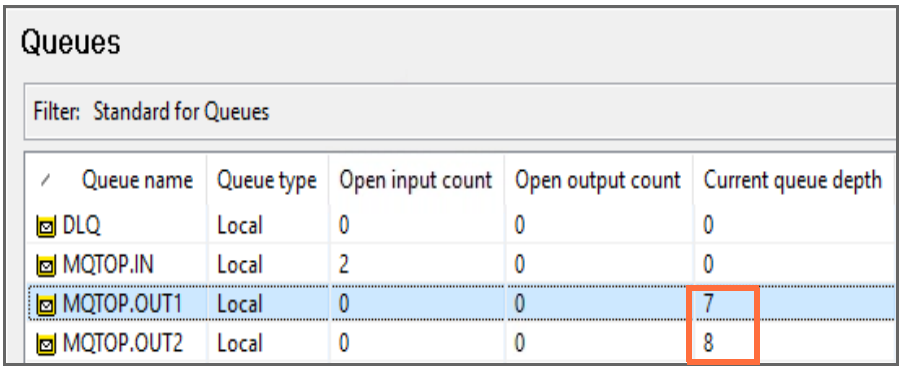
__ 3. Put multiple messages to the MQTOP.IN queue by using the IBM MQ `amqsput` sample program and the `data.txt` file in the `C:\labfiles\Lab03-MQ` directory.

__ a. In a Command Prompt window, type:

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

__ 4. Verify that the workload is shared between the integration nodes and that all 15 messages were processed.

__ a. 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

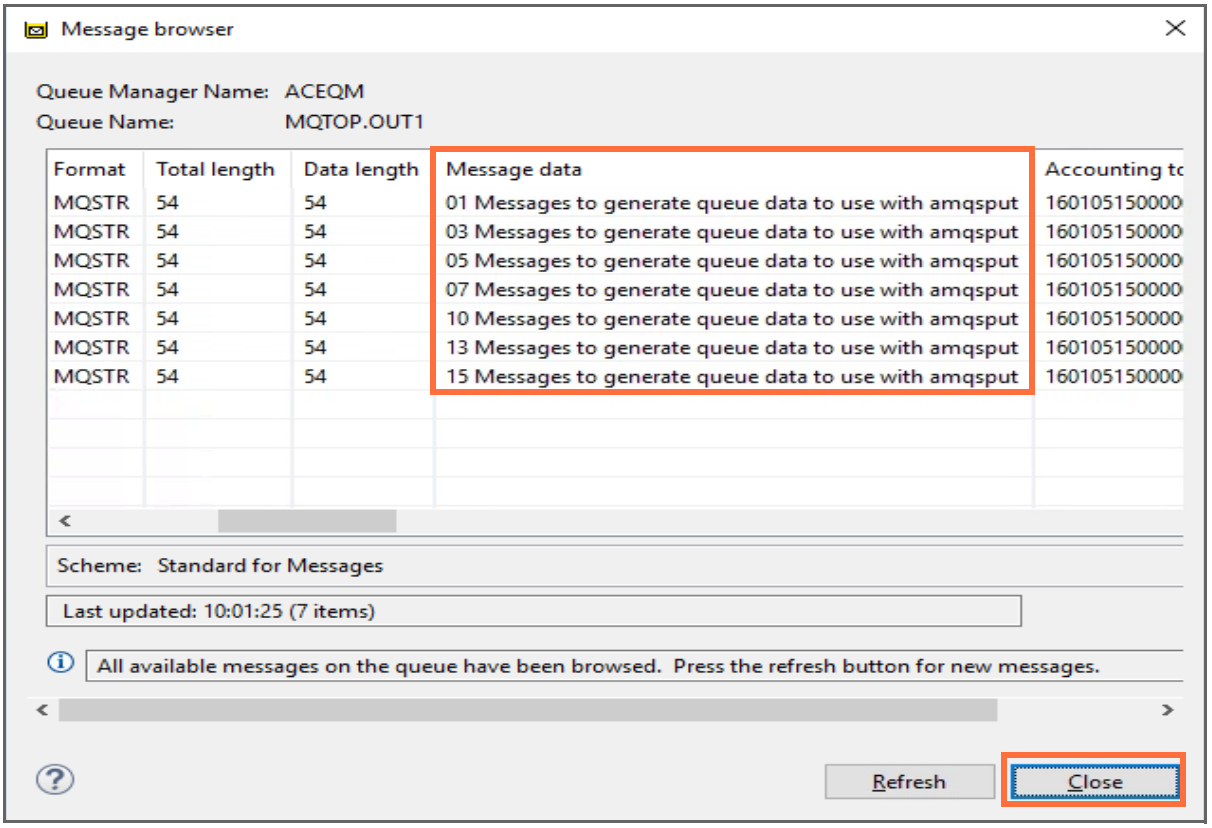
The distribution of the messages on the queues might not exactly match the screen capture that is shown here as an example.

__ b. The messages in the file are numbered so that you can verify that all 15 messages were processed successfully. In the IBM MQ Explorer **Queues** view, right-click the MQTOP.OUT1 and then click **Browse Messages**.

__ c. 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.

__ d. When you’re done viewing the message data, click **Close**.



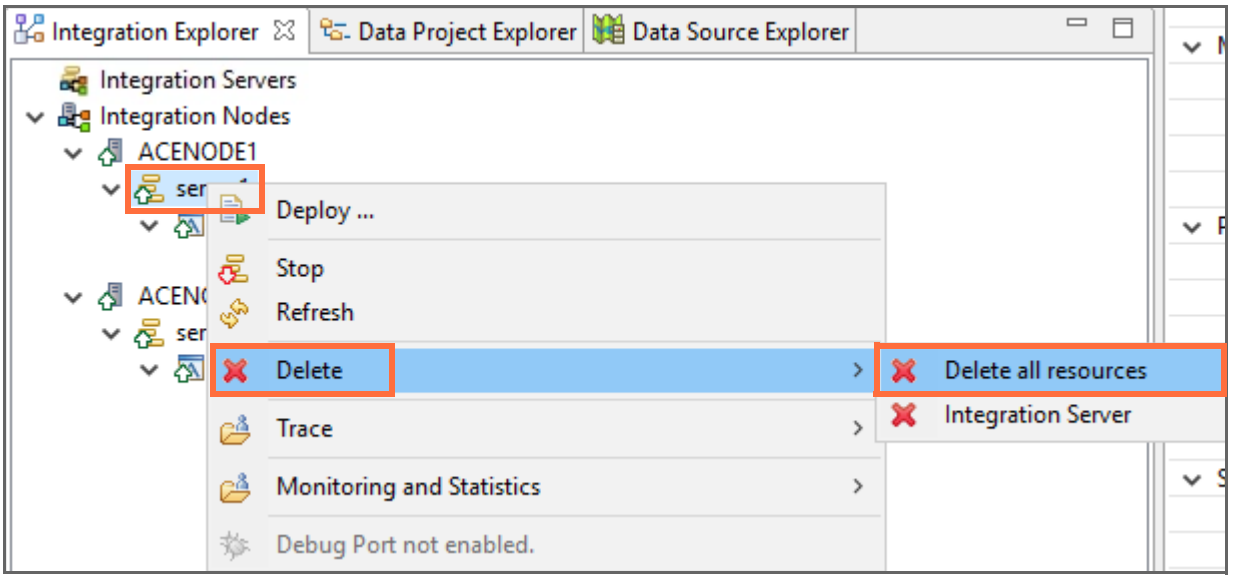
Information

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 acquire the workload.

Part 7: Exercise clean-up

__ 1. In the IBM App Connect Enterprise Toolkit **Integration Explorer** view, delete all flows and resources from **server1** integration server on ACENODE1 and **server2** integration server on ACENODE2.

__ a. Right-click **server1** and then click **Delete > Delete all resources**.



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

The MQTOP application is removed from server1.

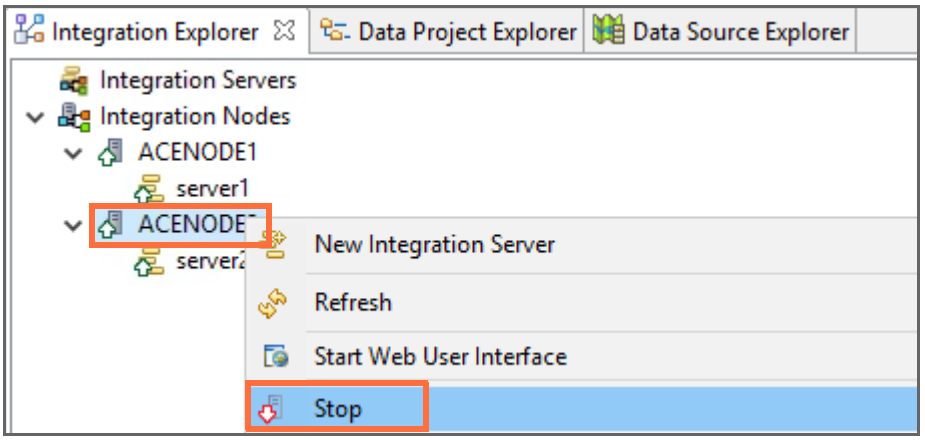
__ c. Right-click **server2** and then click **Delete > Delete all resources**

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

The MQTOP application is removed from server2.

__ 2. Stop and delete ACENODE2

__ a. In the **Integration Explorer** view, right-click ACENODE2 and click **Stop**.



ACENODE2 is now in a stopped state.

__ b. Using the IBM App Connect Enterprise Console, enter the following command to delete the node:

```
mqsideletebroker ACENODE2
```

ACENODE2 is now gone from the Integration Explorer view in the IBM App Connect Enterprise Toolkit.

__ 3. In IBM MQ Explorer, clear the queues

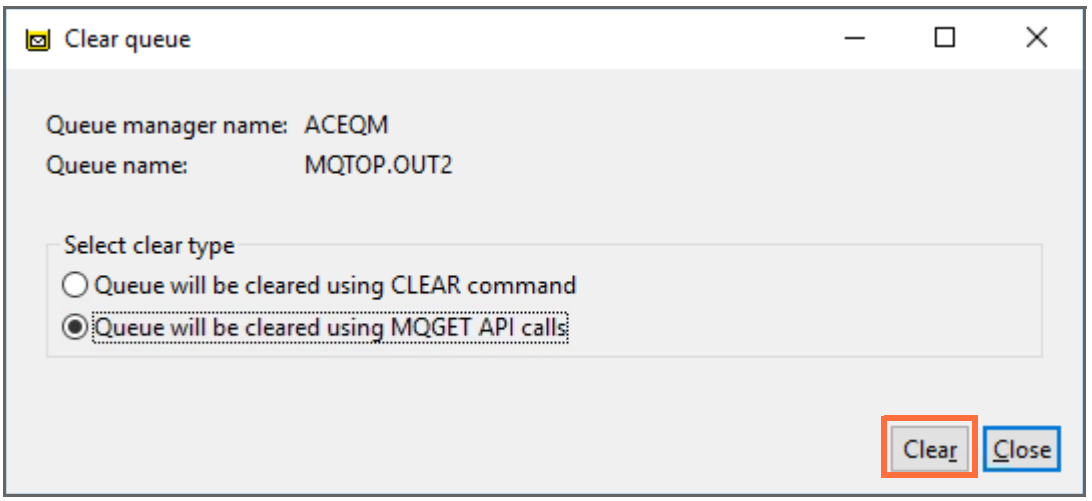
__ a. Return to IBM MQ Explorer.

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

__ c. Select MQTOP.OUT2

__ d. Right-click and select **Clear Messages...**

__ e. When presented with the option to clear the queue, select the **Queue will be cleared using MQGET API calls** option and click **Clear**.



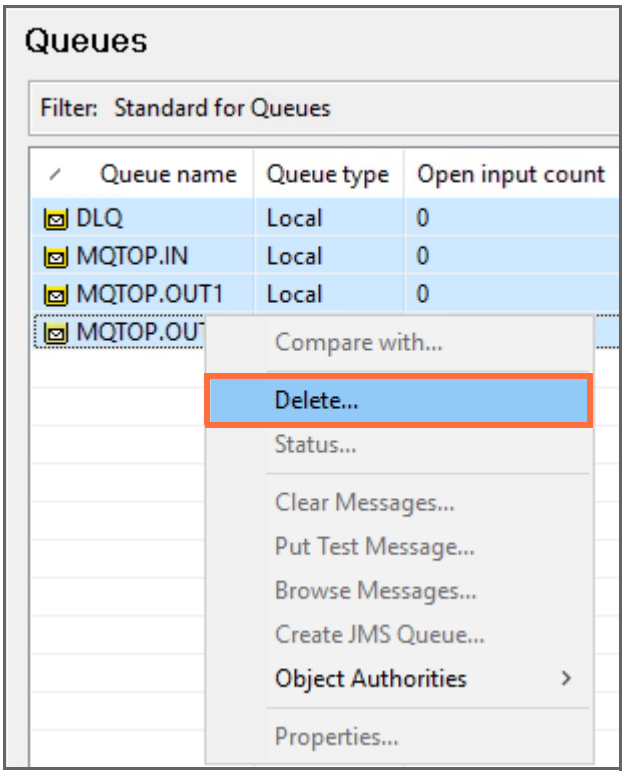
__ f. Click **OK** on the confirmation dialog.

__ g. Repeat the last three steps to clear the **MQTOP.OUT1** queue.

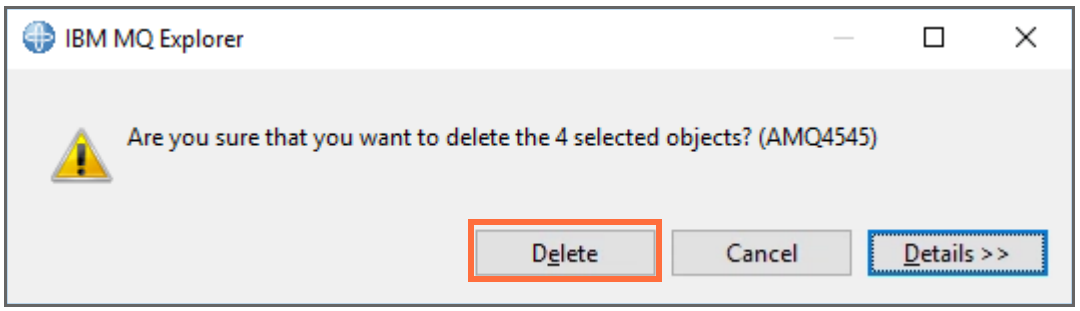
__ 4. In IBM MQ Explorer, delete all queues.

__ a. Select all queues.

__ b. Right-click the selection and select Delete.



__ c. Click Delete on the confirmation dialog.



__ d. Click OK on each successful deletion dialog for each object.

Information

When cleaning up the environment at the end of an exercise, be sure to delete resources from the integration server before deleting the queues in MQ.

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

Estimated time

01:00

Overview

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.

Objectives

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 App Connect Enterprise 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.

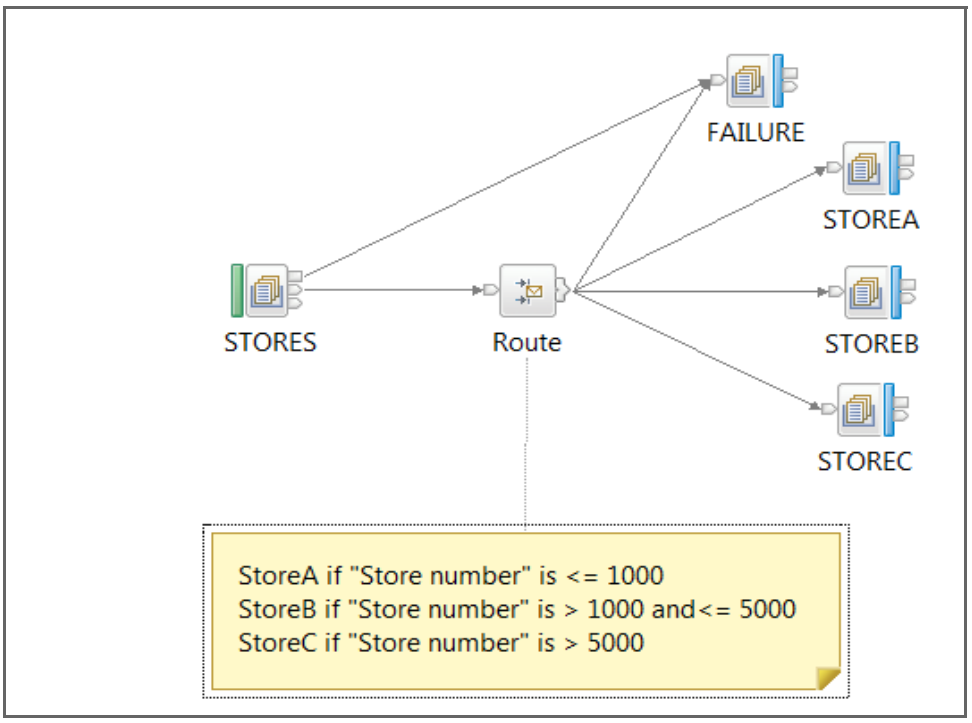
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="stock-id" type="xsd:string"/>
  <xsd:element name="store-number" type="xsd:string"/>
  <xsd:element name="inputrec">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element 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 App Connect Enterprise 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 App Connect Enterprise V11 Toolkit and IBM MQ V9
- A local IBM MQ queue manager that is named ACEQM with the following local queues: DLQ, FAILURE, STOREA, STOREB, STOREC, and STORES.IN
- Lab files in the `C:\labfiles\Lab04-Routing` directory
- The user `aceadmin` is a member of the “mqm” group

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab04`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

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

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

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

__ 2. Verify that the node and server are running.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 3. This exercise requires an IBM MQ queue manager.

If you completed Exercise 3, you created a queue manager that is named ACEQM. You can use that queue manager in this exercise. Proceed to the next step.

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

__ a. Start IBM MQ Explorer.

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

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

__ 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.

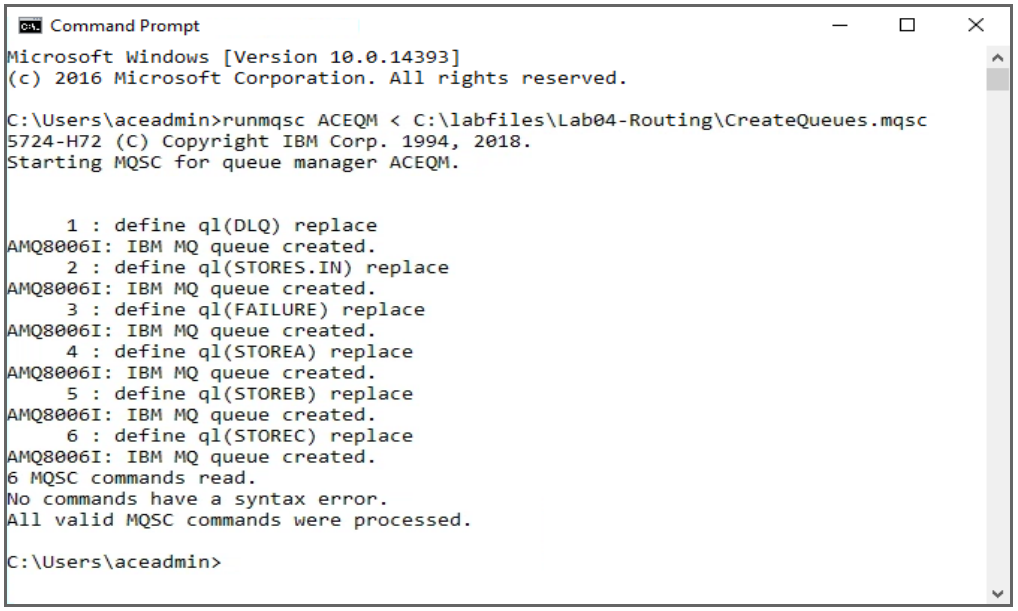
The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 4. 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, enter:

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

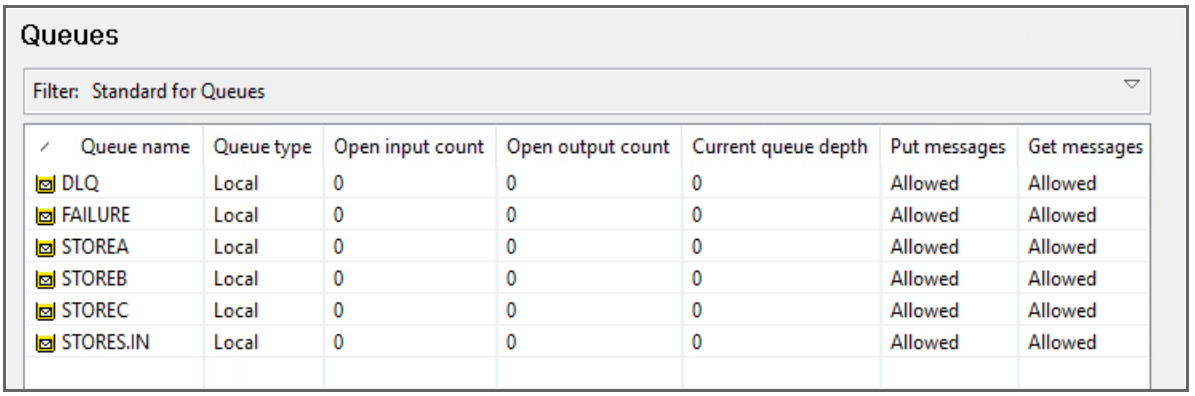


__ 5. Use IBM MQ Explorer to verify that the queues were created.

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



Queue name	Queue type	Open input count	Open output count	Current queue depth	Put messages	Get messages
DLQ	Local	0	0	0	Allowed	Allowed
FAILURE	Local	0	0	0	Allowed	Allowed
STOREA	Local	0	0	0	Allowed	Allowed
STOREB	Local	0	0	0	Allowed	Allowed
STOREC	Local	0	0	0	Allowed	Allowed
STORES.IN	Local	0	0	0	Allowed	Allowed

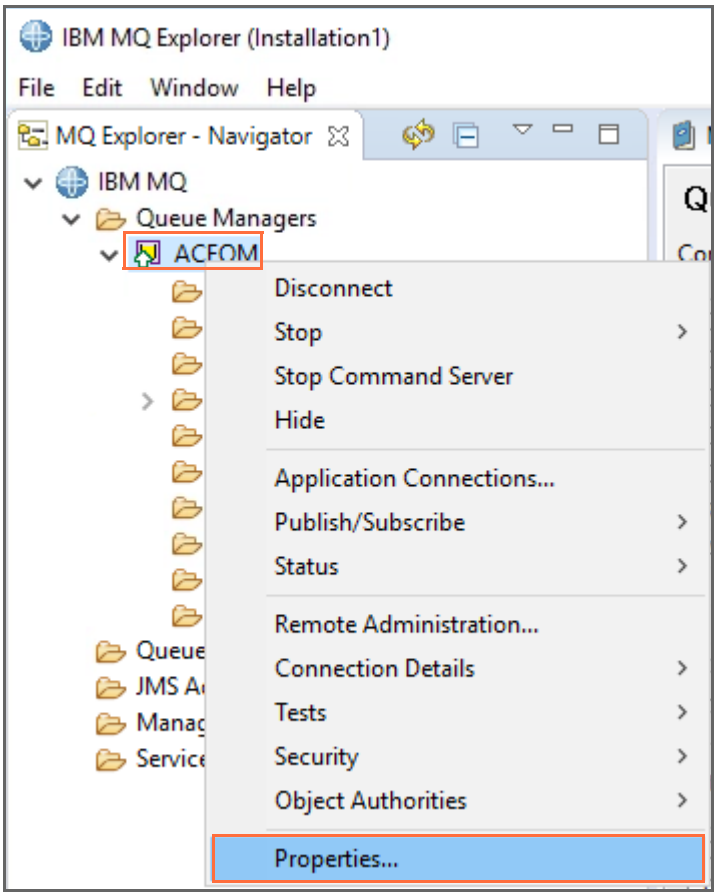
Information

You might have other queues for this queue manager from a previous exercise. Their presence does not effect the lab exercise.

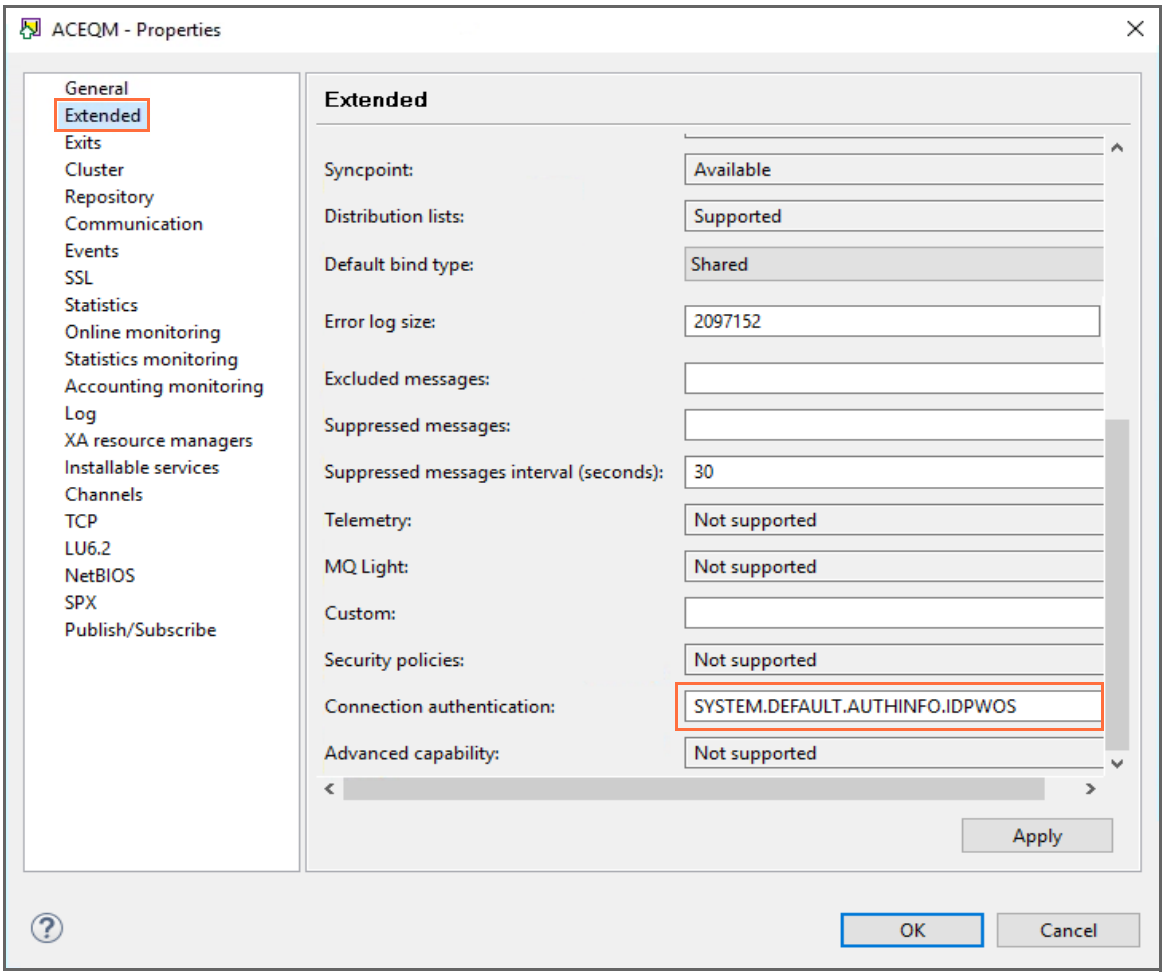
__ 6. Configure MQ security to enable Flow exerciser testing.

In order to allow communication to occur between IBM App Connect Enterprise and IBM MQ (to enable Flow exerciser testing), perform the following.

__ a. Right-click ACEQM in MQ Explorer and select **Properties**.



__ b. Select the Extended tab and delete the contents of the **Connection authentication** value.



__ c. Click OK

Next you update the connection authentication to allow **mquser** access on the **JAVA.CHANNEL**.

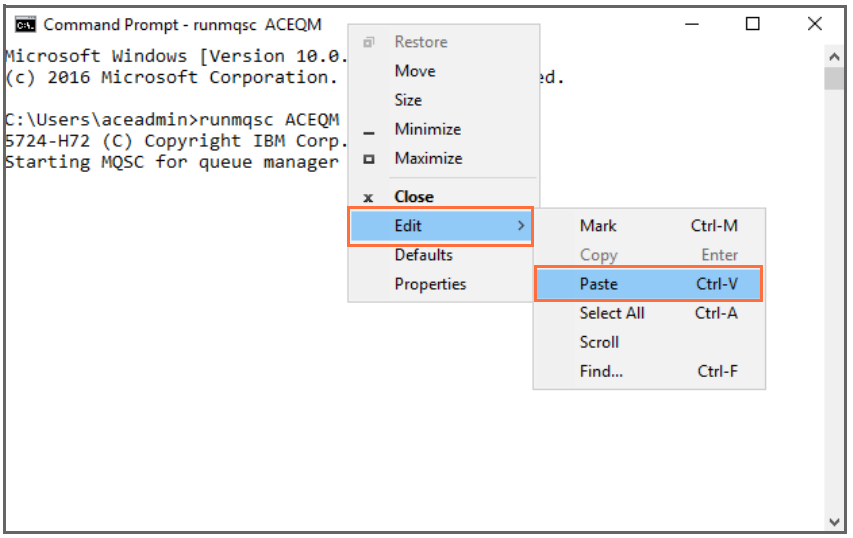
__ d. Start a Command Prompt.

A text file that is named `Cut&Paste.txt` in the `C:\labfiles\Lab04-Routing` directory contains the list of commands below. To ensure that the syntax is correct, you can copy the text from this file and paste it into the **Command Prompt** window.

Enter the commands one at a time. When you enter the first command, it starts the **mqsc** interface where you can enter commands to run against the queue manager. The window displays a blinking cursor.

Hint

To cut and paste into the Command Prompt window, first copy the command line from Notepad. Then right-click the Command Prompt header and select **Edit > Paste**.



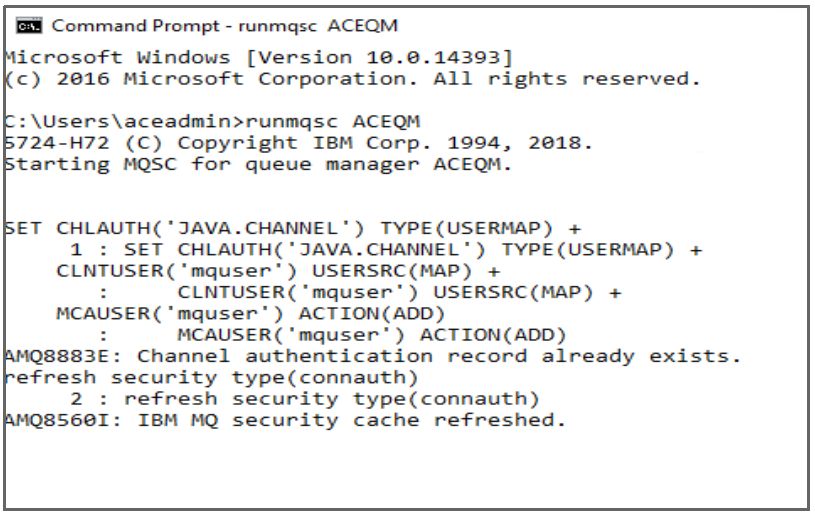
__ e. Start the MQSE interface by entering the following command: `runmqsc ACEQM`

__ f. Allow `mquser` access to the queue manager by entering the following:

```
SET CHLAUTH('JAVA.CHANNEL') TYPE(USERMAP) +
    CLNTUSER('mquser') USERSRC(MAP) +
    MCAUSER('mquser') ACTION(ADD)
```

__ g. Update security by entering the following command:

```
refresh security type(connauth)
```



__ h. Close the Command prompt window.

Information

Once you create the queue manager, you reuse it throughout the course. If you delete the queue manager, you need to rerun these commands again to use the Flow exerciser.

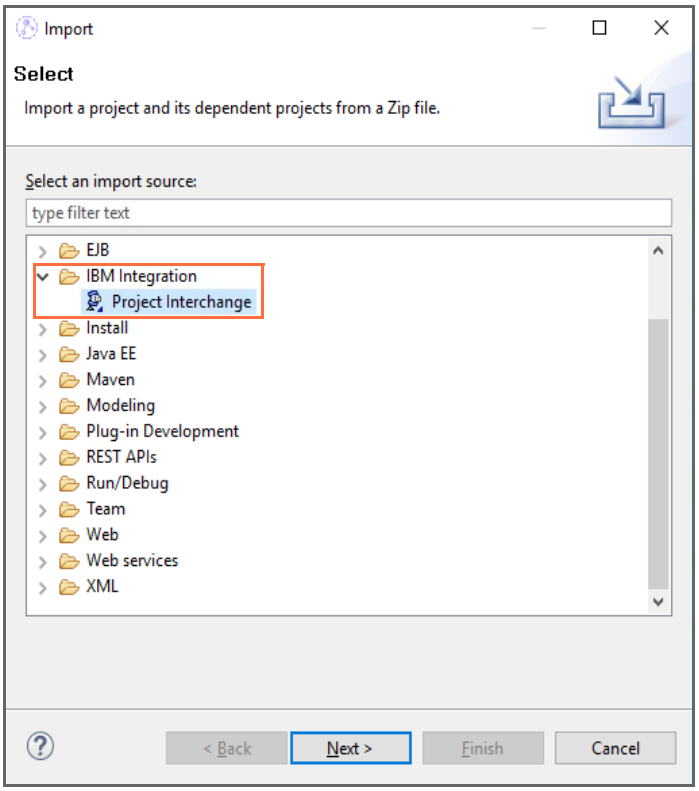
Part 2: Add routing to the message flow

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

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

__ a. From the IBM App Connect Enterprise Toolkit, click **File > Import**.

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

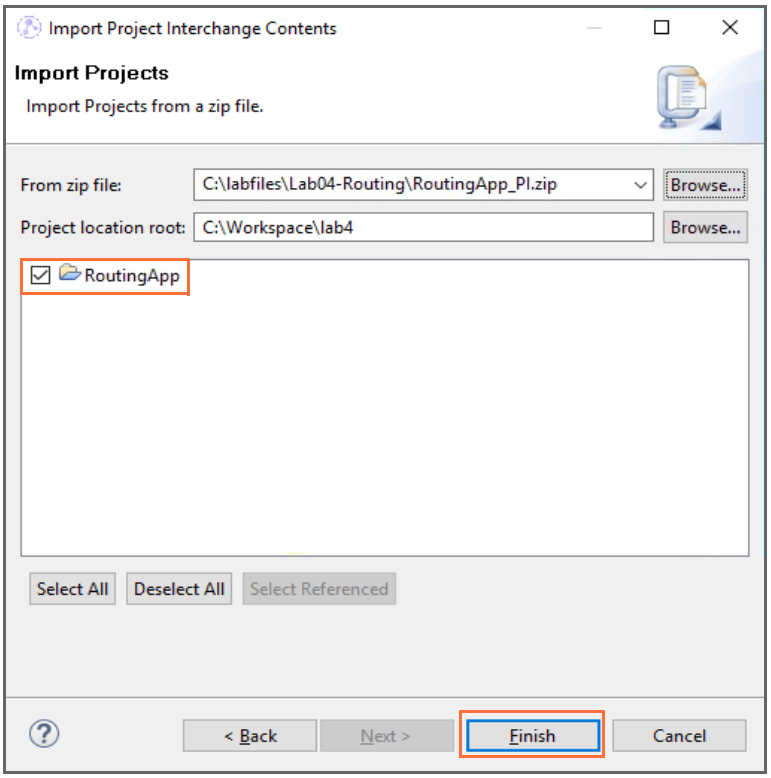


__ c. To the right of **From zip files**, click **Browse** to locate the project interchange file.

__ 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.`

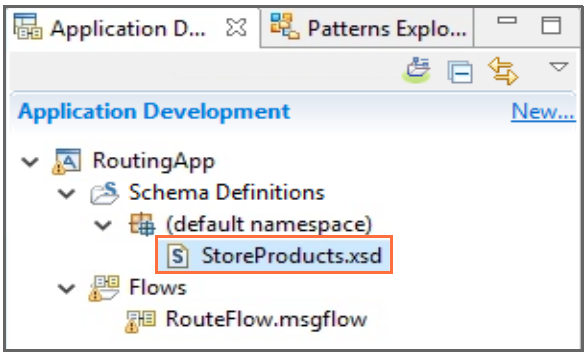
__ f. This project interchange file contains one application that is named **RoutingApp.** Ensure that it is selected and then click **Finish**.



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

__ 2. Examine the RoutingApp application.

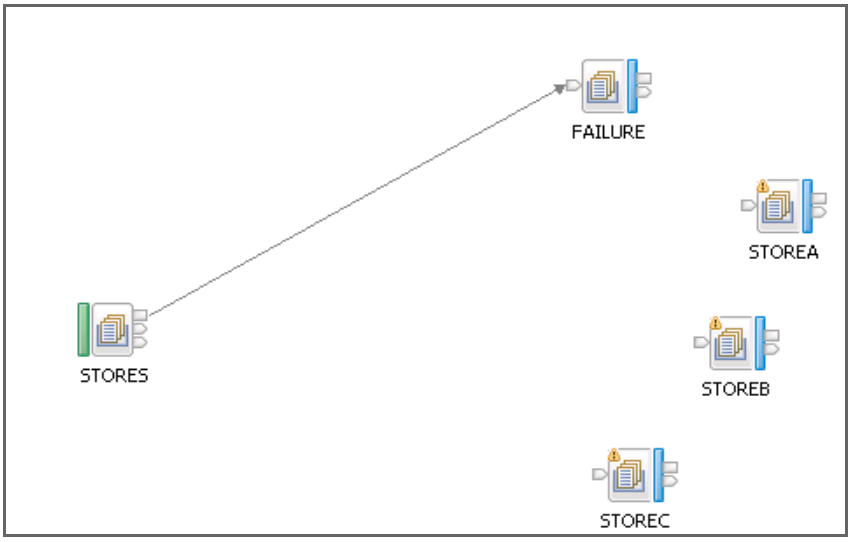
__ h. Expand the application to view the components in the **RoutingApp** application.



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

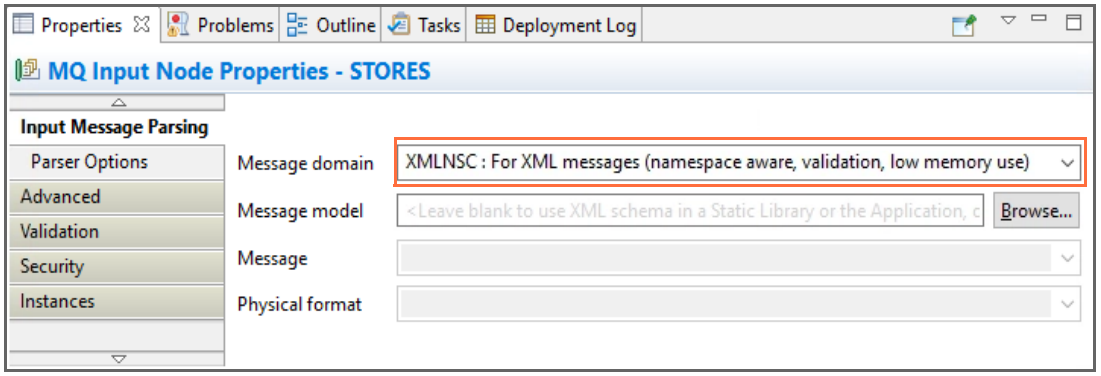
__ i. Double-click the message flow **RouteFlow.msgflow** in the Application Development view to open it in the Message Flow editor.

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.



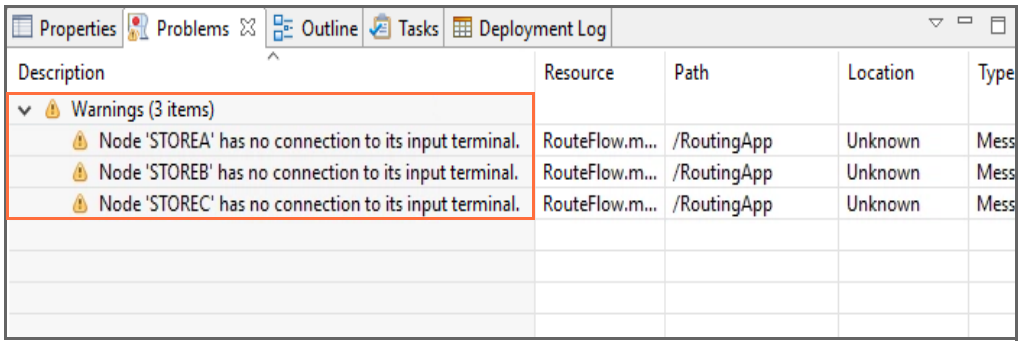
__ j. Examine the **Input Message Parsing** tab in the **Properties** view for the MQ Input node that is named STORES.

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**.



__ k. 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.

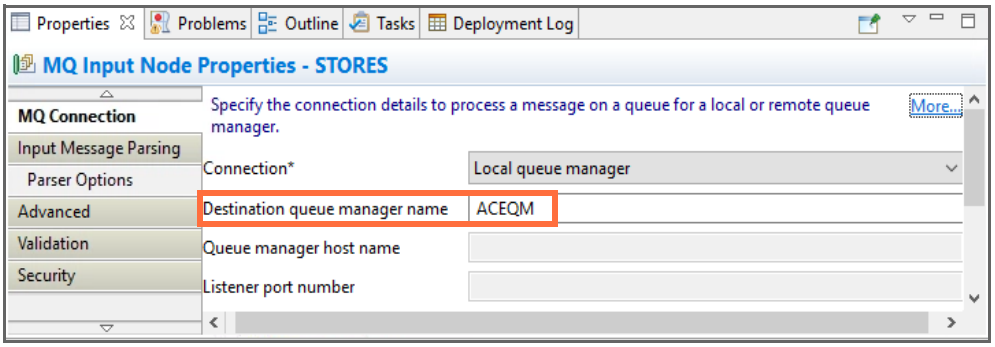


__ 3. Update the queue manager for each node.

__ a. 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 imported solution was built in a prior version using IBM Integration Bus V10. The solution used a different queue manager. The **MQ Connection** tab identifies the queue manager. The integration node that you use in this exercise, **ACENODE1**, does not have a specified default queue manager so the **MQ Connection** tab must specify ACEQM as the **Destination queue manager name**.

__ b. Change the **Destination queue manager name** from IIBQM to ACEQM for each node.

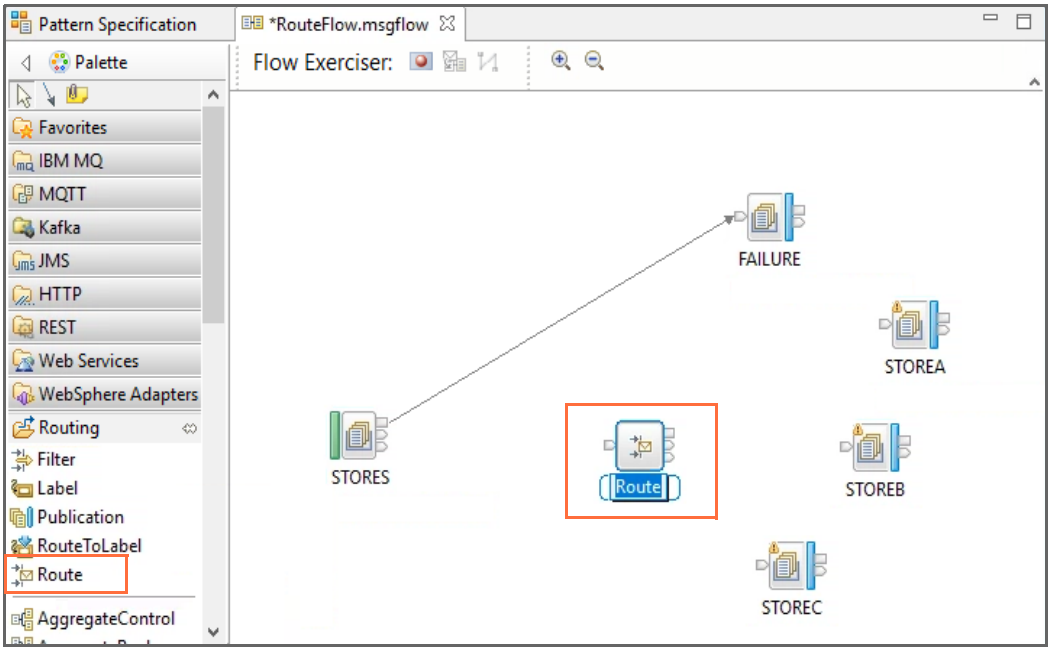


__ c. Save your work.

__ 4. Add the Route node to the message flow.

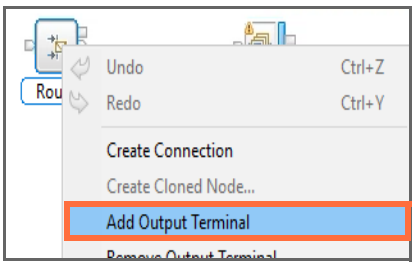
__ a. Expand the **Routing** drawer in the Message Flow editor Palette.

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

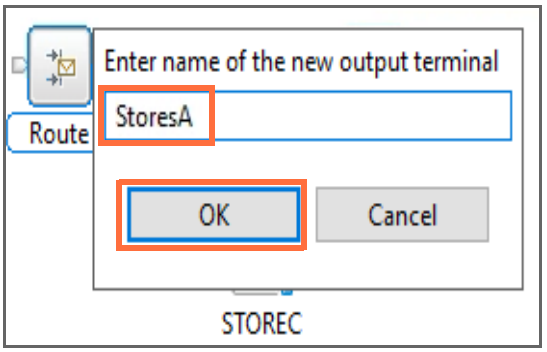


__ 5. Add output terminals to the Route node for each route.

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



__ b. For the terminal name, enter: `StoresA`



__ c. Click **OK**.

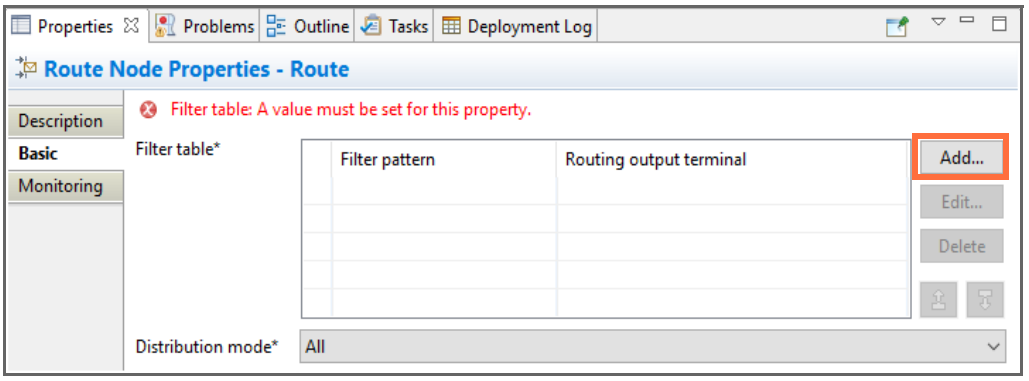
__ d. Using the same procedure, add two more output terminals that are named `StoresB` and `StoresC`.

Information

If a node has five or more terminals, they are displayed in a group.

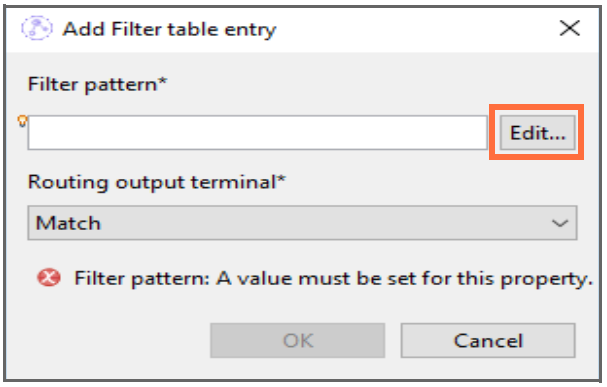
__ 6. Populate the Route node **Filter table** with the filter patterns and identify the routing output terminal.

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



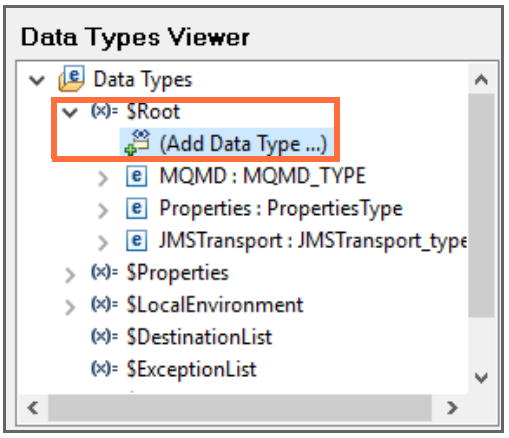
In this exercise, you use the XPath Expression Builder to create the filter pattern.

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

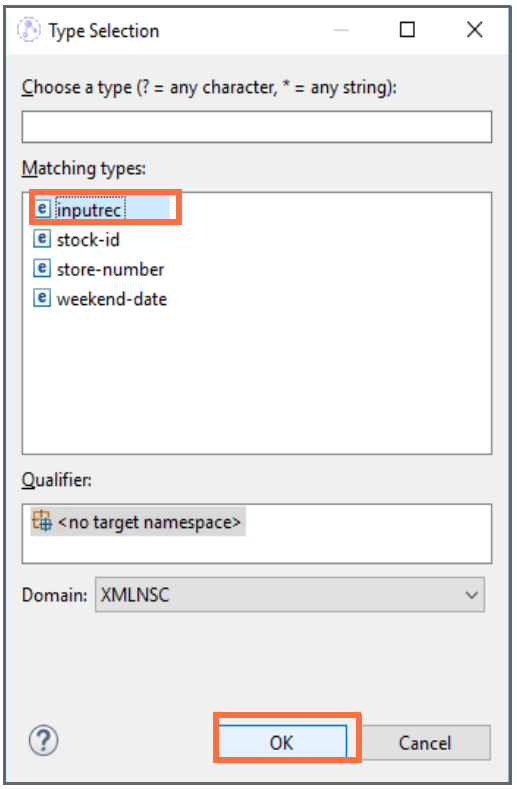


__ c. You need to add the XML schema structure and data types that define the XML message to the XPath Expression Builder.

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

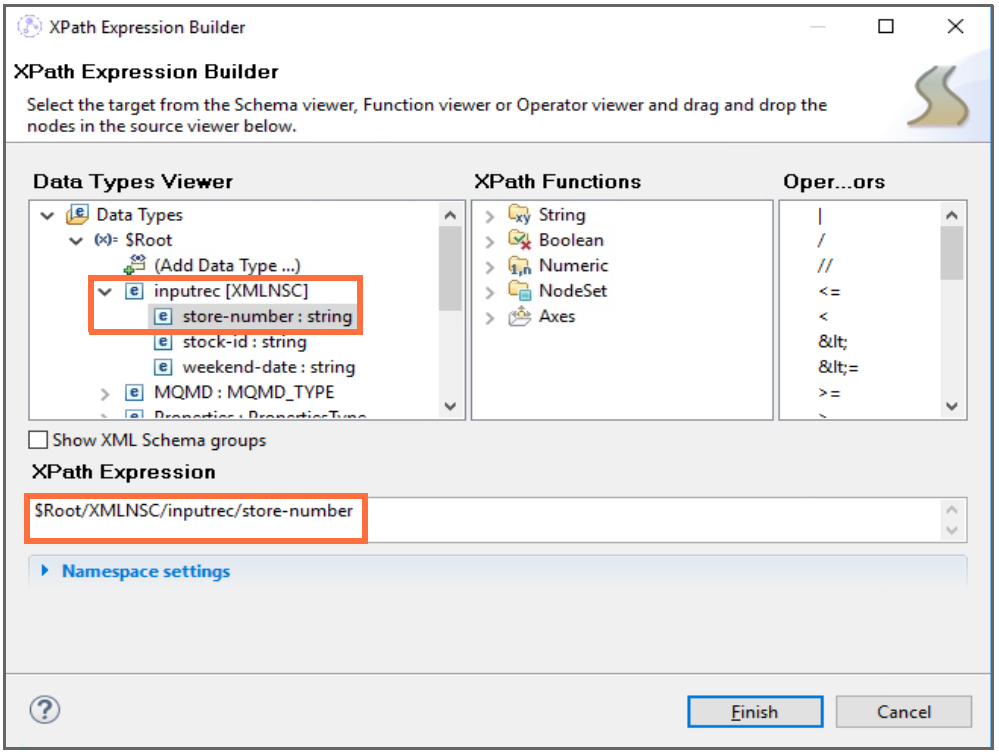


__ d. Select the top-level object in the `StoreProducts.xsd` schema, which is **inputrec.** Click **OK** to add it to the Data Types hierarchy.



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

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

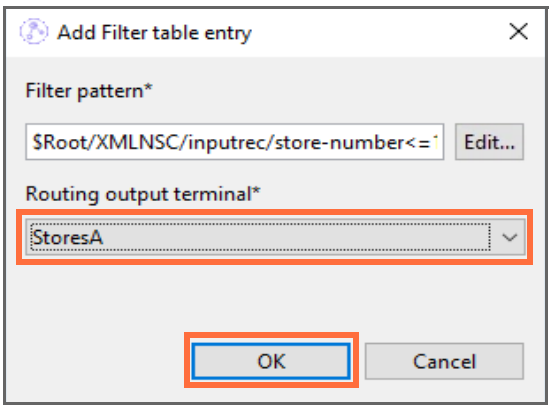


__ g. You can either type the remaining portion of the expression or select the operators (< and =) from the **Operators** column to complete the XPath Expression:

```
$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

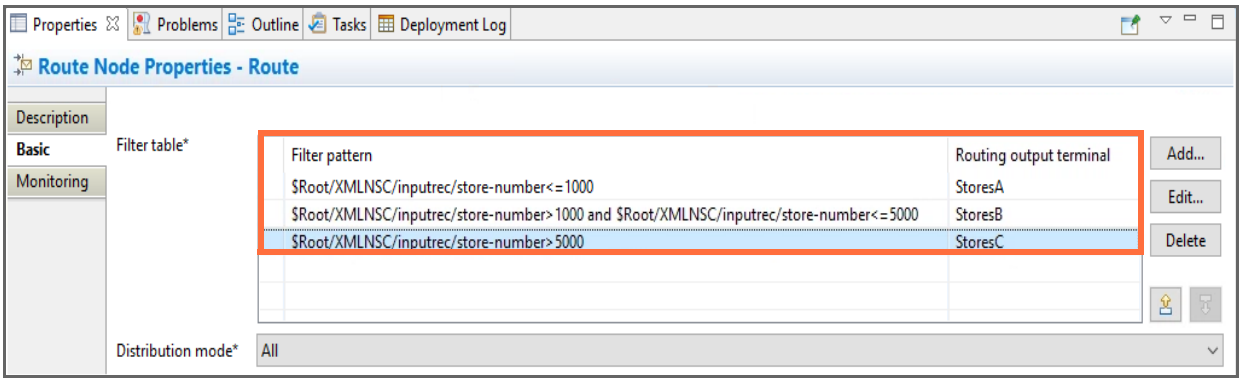
You do not need to repeat Step C to add the **inputrec** data type.

__ 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.

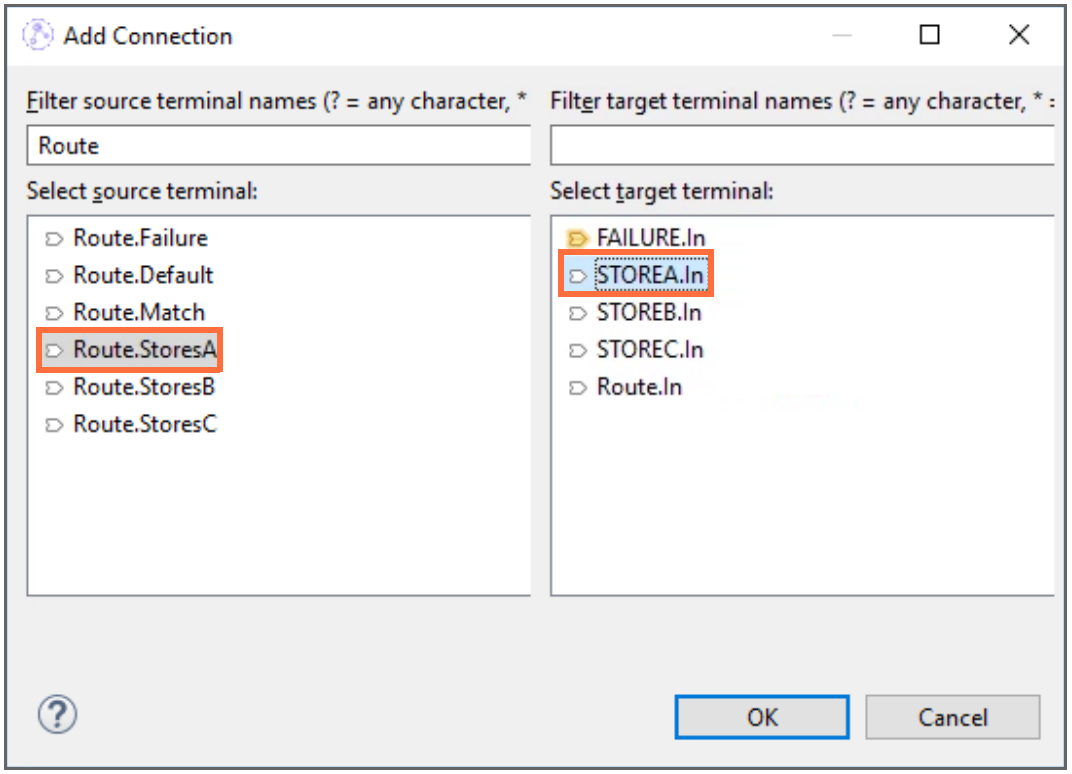


__ m. Save your work.

__ 7. 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.

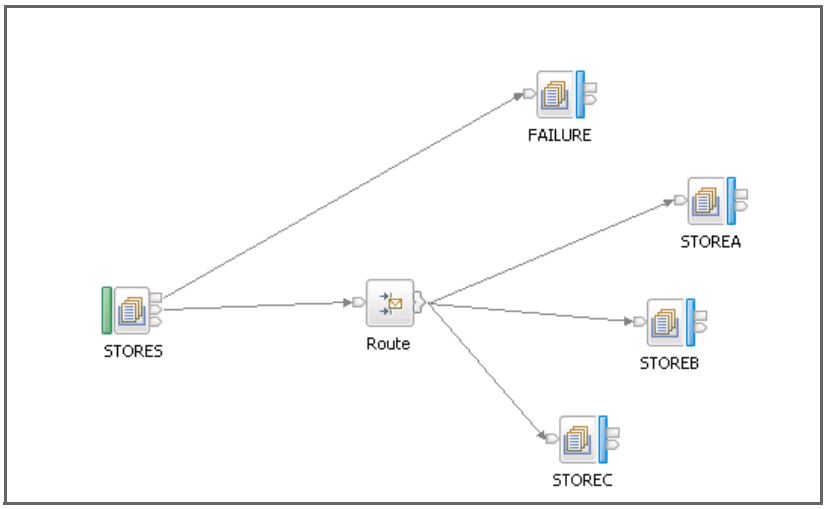


__ c. Click **OK**.

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

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

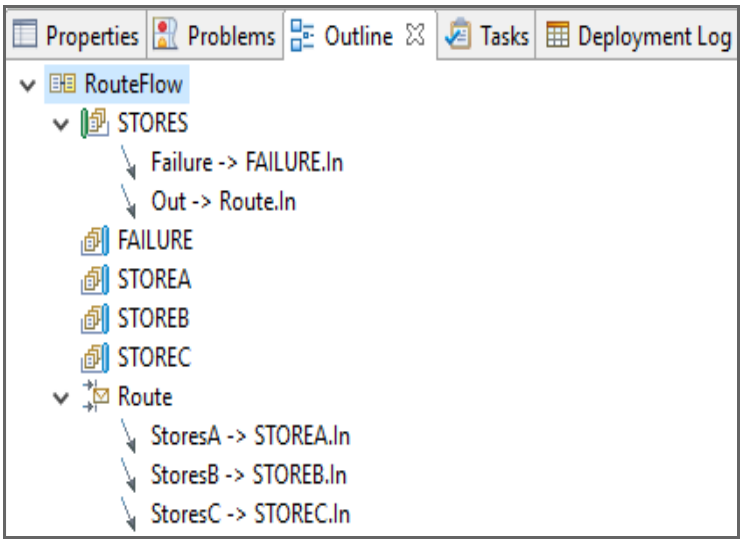
__ f. In the Message Flow editor, connect the STORES MQInput node **Out** terminal to the Route node **In** terminal.



__ g. Save your work.

Now that the input nodes are wired, all warnings should be cleared in the Problems pane.

__ h. All the nodes in the message flow should now be connected. Verify the message flow connections in the **Outline** view.



Part 3: Test the message flow

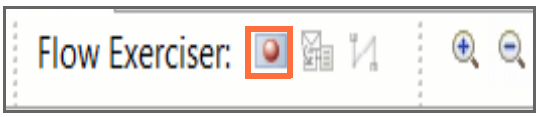
In this part of the exercise, you use the IBM App Connect Enterprise Toolkit Flow exerciser to test the message flow.

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 App Connect Enterprise Toolkit Message Flow editor, start the Flow exerciser to create the BAR file, deploy the application to **server1** on **ACENODE1**, and start recording.

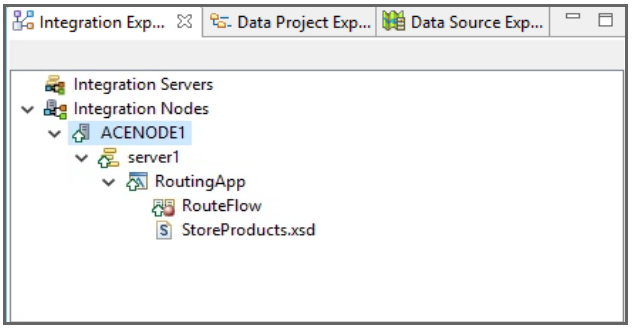
__ a. Click the **Start flow exerciser** icon at the top of the Message Flow editor.



__ b. Click **OK** on the Confirm dialog box.

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

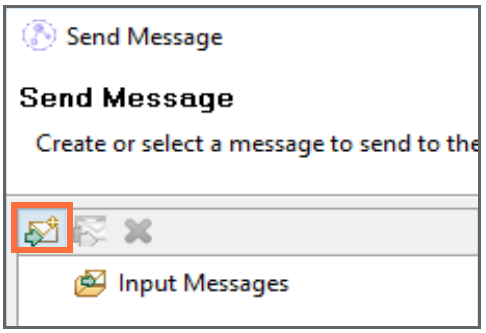
__ d. In the Integration Explorer view, expand the **ACENODE1** > **server1** to display the contents. Verify that the **RouteFlow** message flow is in record mode (signified by the Recording icon).



__ 2. 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.



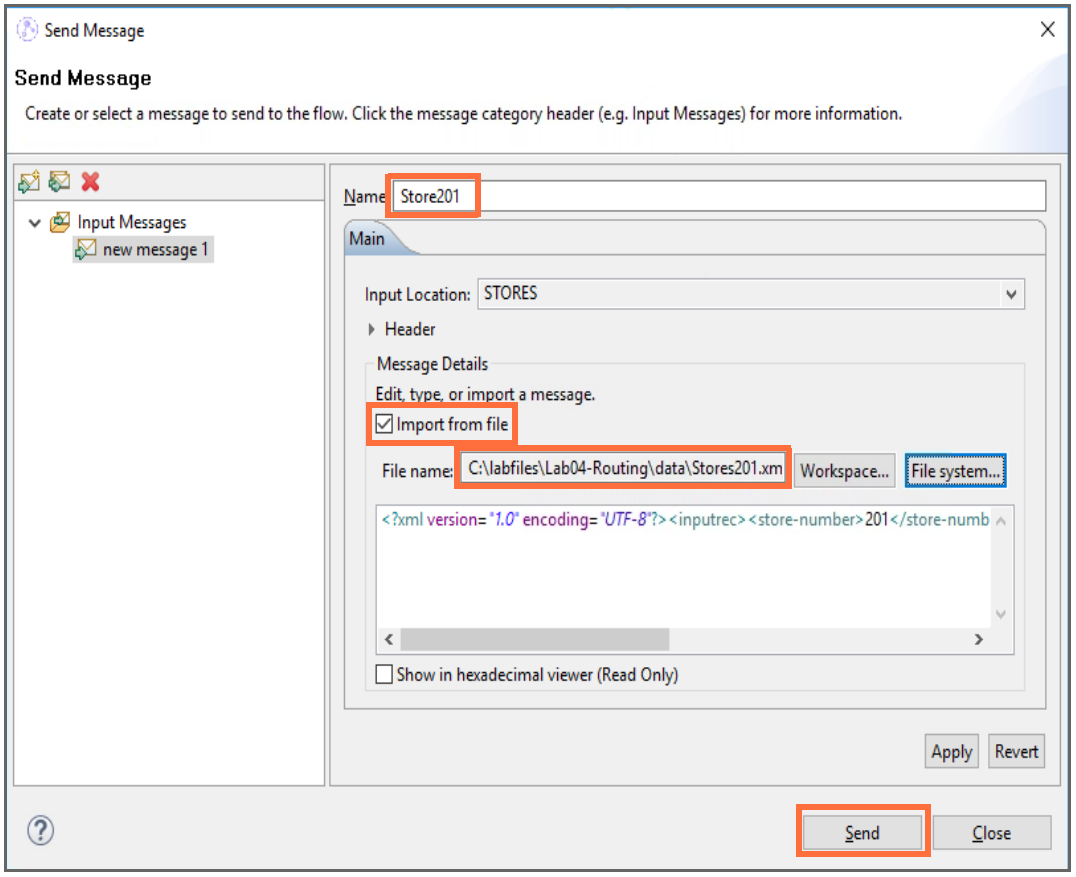
__ c. For the **Name**, type: **Store201**

__ 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**.

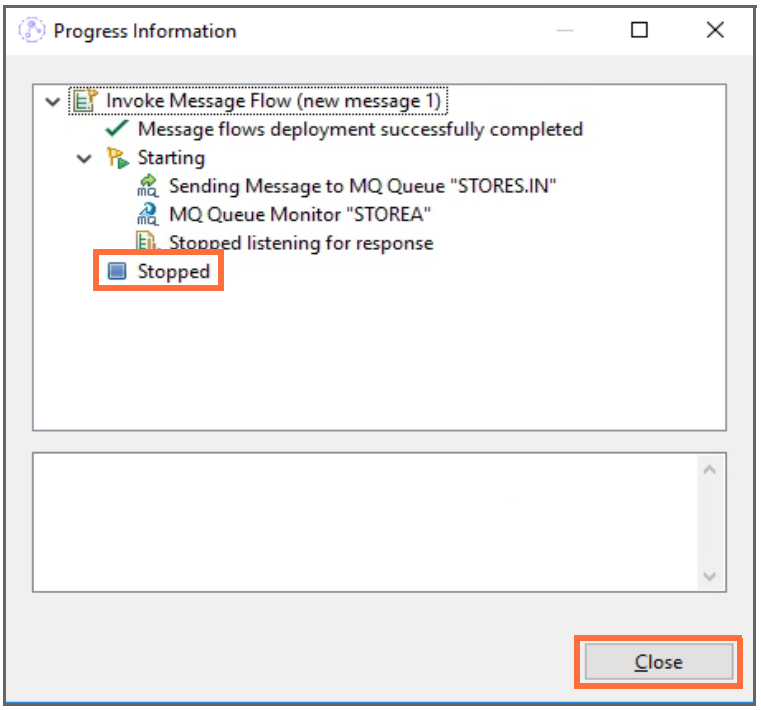
The file is imported into the Flow exerciser.

__ 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.

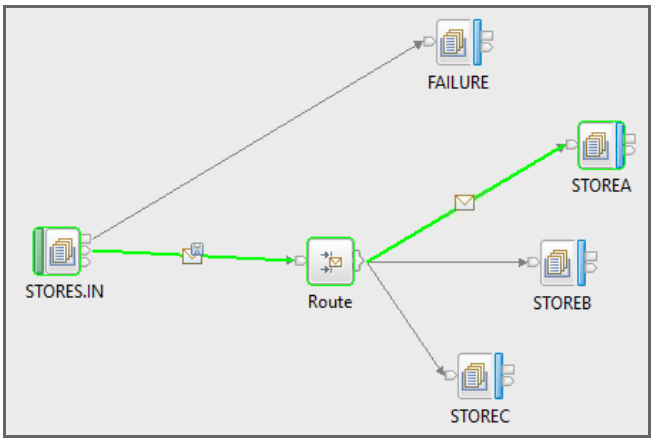
When the **Progress Information** window displays **Stopped**, the test is complete. Click **Close**.



Troubleshooting

If the message flow doesn't complete, verify that the filter patterns were entered correctly.

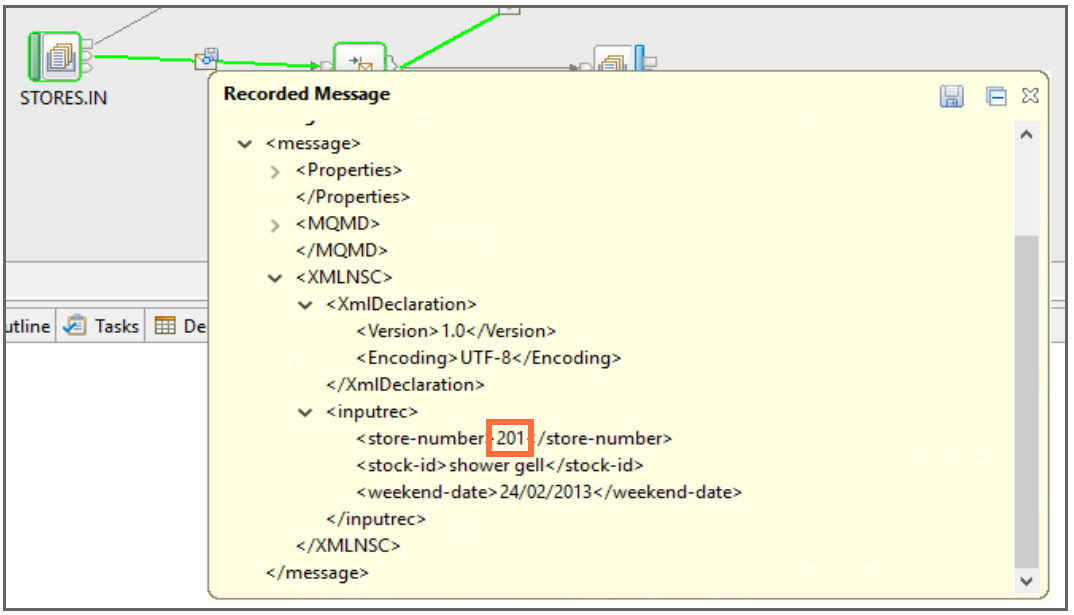
__ h. The message path is highlighted on the message flow.



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 application 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 ACEQM queue manager, enter the following command in a command prompt window:

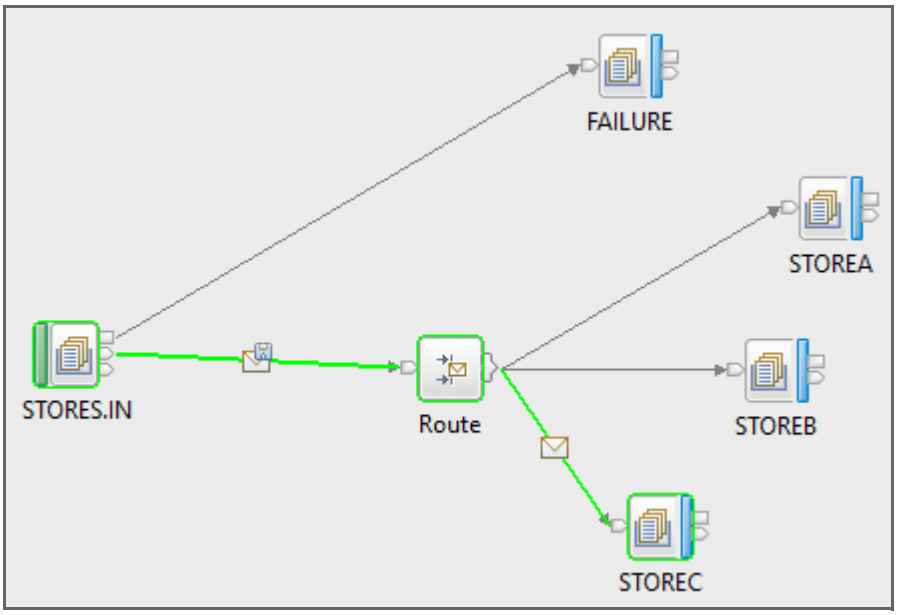
```
amqsput STORES.IN ACEQM < C:\labfiles\Lab04-Routing\data\Store201.xml
```

__ 3. 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.

__ 4. 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.



Part 4: Exercise clean-up

__ 1. In the IBM App Connect Enterprise Toolkit, stop recording on the message flow.

__ 2. In the **Integration Explorer view**, delete all resources from **server1** integration server on **ACENODE1**.

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

__ 4. In IBM MQ Explorer, delete the queues

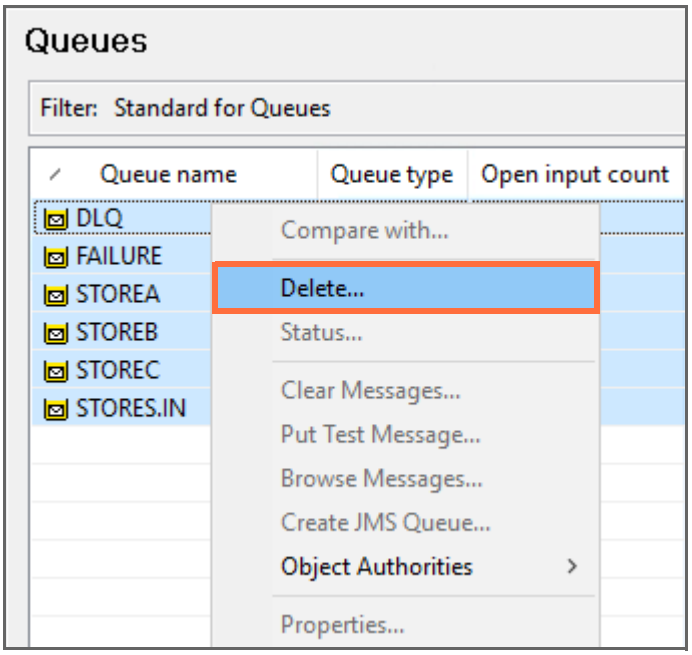
__ a. Open IBM MQ Explorer.

__ b. Expand the **ACEQM** folder.

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

__ d. Select all queues.

__ e. Right-click and select **Delete...**



__ f. When asked if you are sure that you want to delete the six selected objects, click **Delete**.

__ g. Click OK on each successful deletion dialogue for each object.

__ h. If messages still exist in any of the queues, you are asked to clear messages. Click the checkbox to **Clear all messages from the queue** if asked.

All queues are deleted.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you imported an IBM App Connect Enterprise 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 App Connect Enterprise Toolkit Flow exerciser

Exercise 5. Creating a DFDL model

Estimated time

01:00

Overview

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.

Objectives

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.

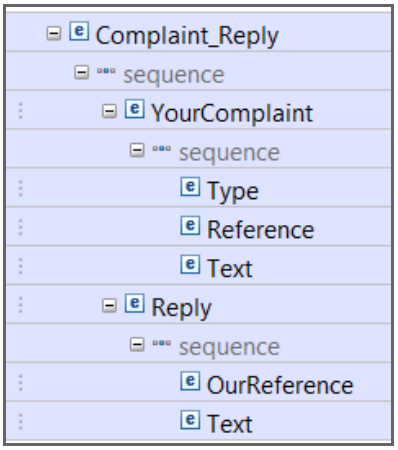
The reply file consists of two records that are delimited by a '|'. In each record, the string '+++' delimits each field in the record.

```
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 App Connect Enterprise V11 App Connect Enterprise Toolkit
- The lab files in the `C:\labfiles\Lab05-DFDL` directory

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab05`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, enter `C:\Workspace\Lab05`

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

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

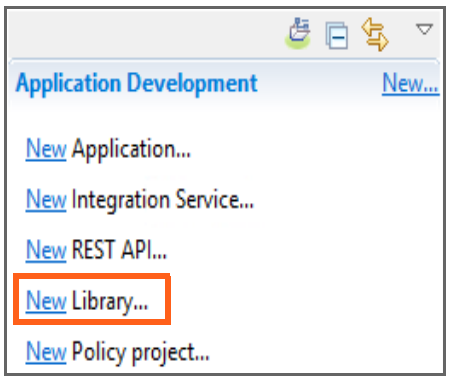
Part 2: Create the DFDL model that defines the reply message

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 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.

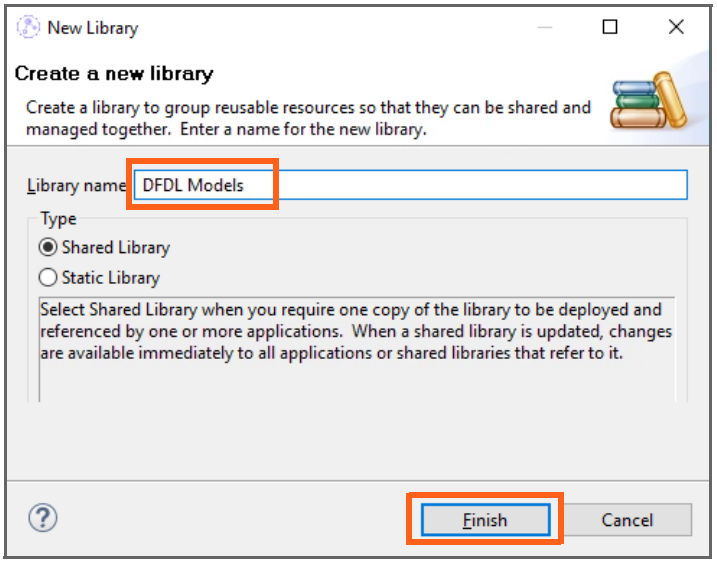
__ a. In **Application Development** view, click **New Library**.



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

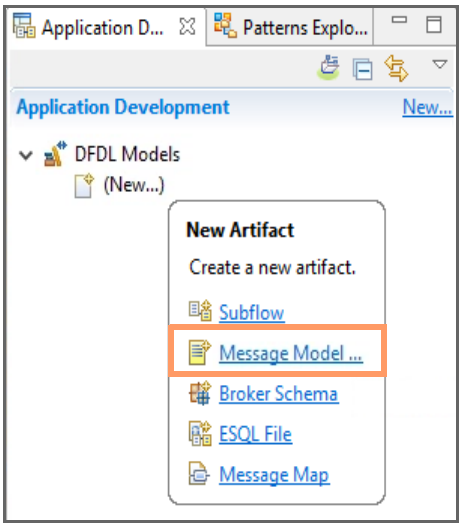
__ c. Ensure that the **Shared Library** option is selected.

__ d. Click **Finish**.

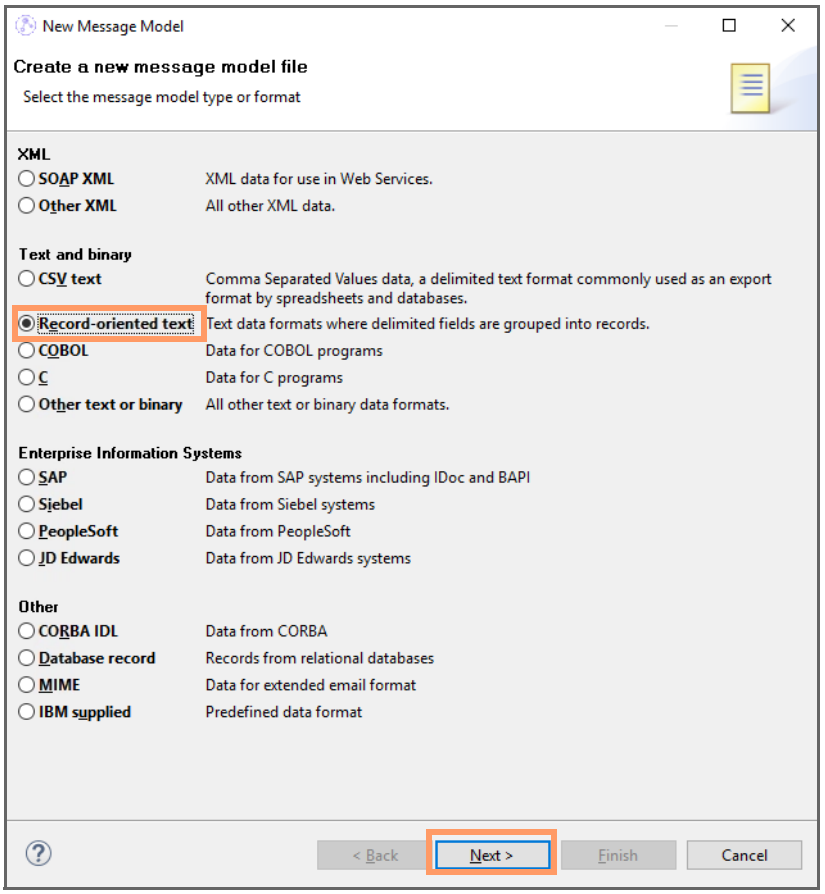


__ 2. Use the New Message Model wizard to create a DFDL schema definition file for the reply file that is described in the Introduction for this exercise.

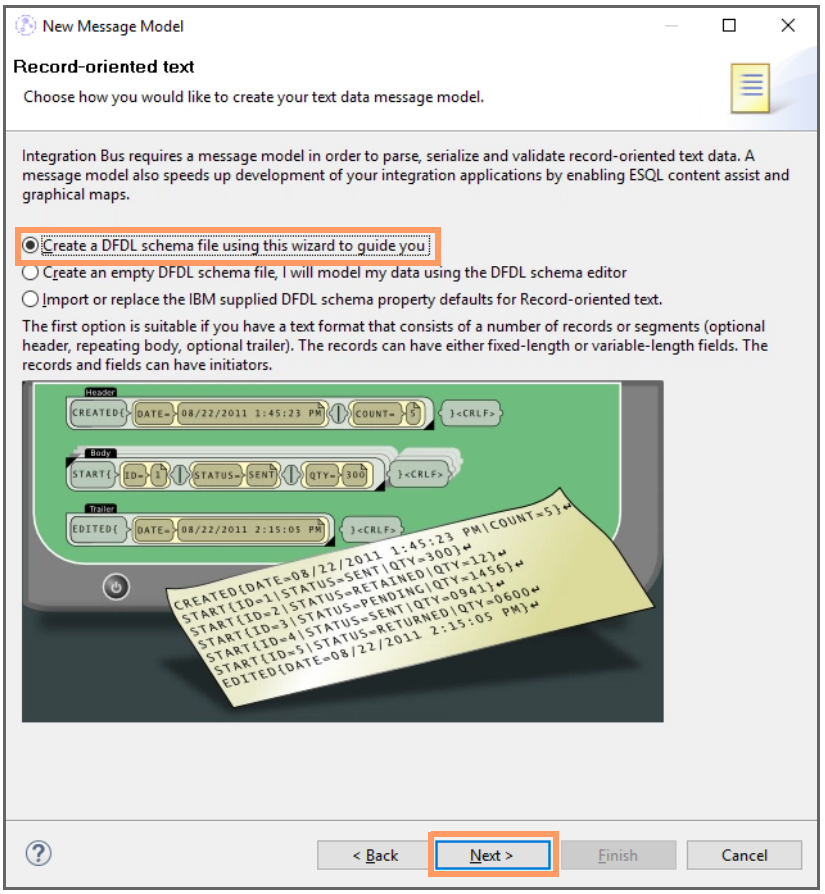
__ a. Click **New** under the **DFDLModels** library folder in the **Application Development** view and then click **Message Model**.



__ 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**.

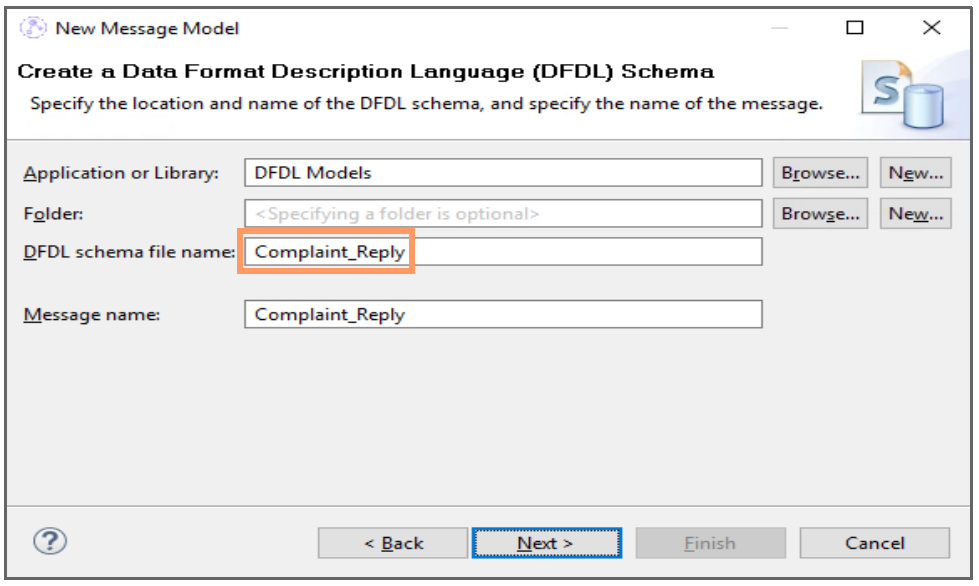


__ c. On the **Record-oriented text** page of the wizard, ensure that the option to **Create a DFDL schema file using this wizard to guide you** is selected and then click **Next**.



__ 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.

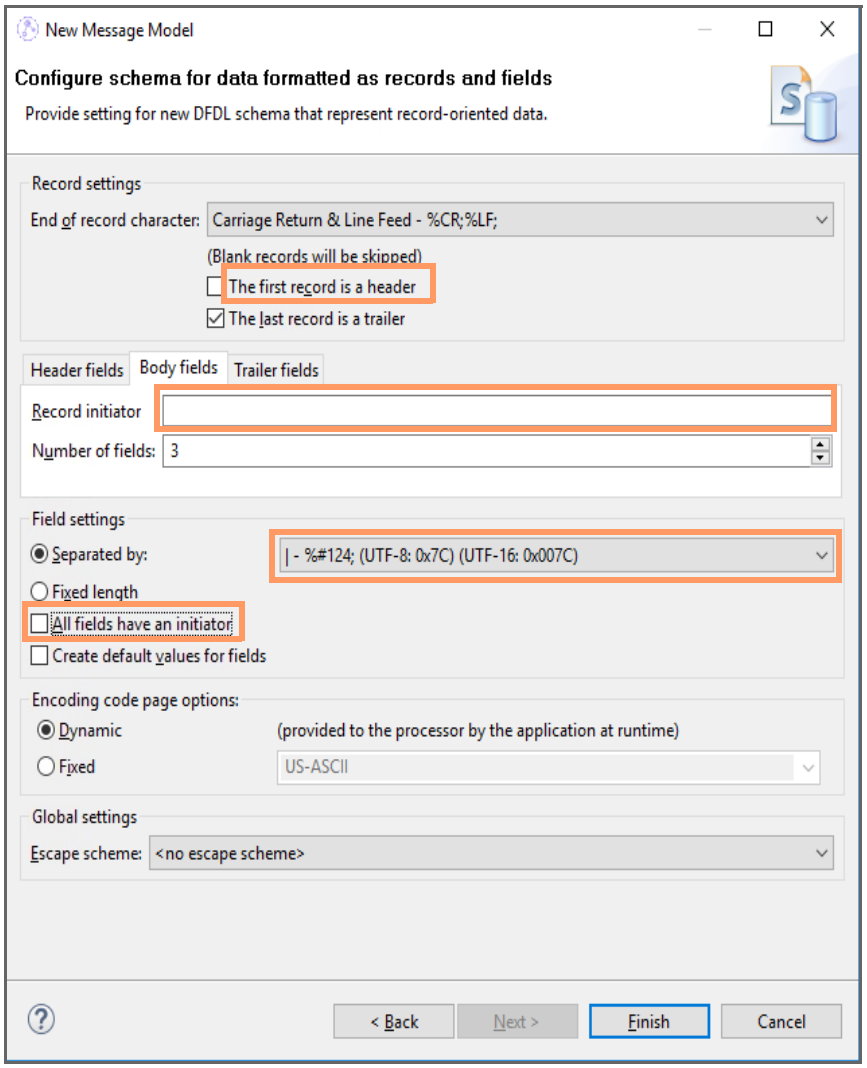


Because you started the wizard by clicking **New** under the DFDLModels library, the wizard automatically selects **DFDLModels** for the **Application or Library** field.

__ 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 fields** page, complete the following actions:

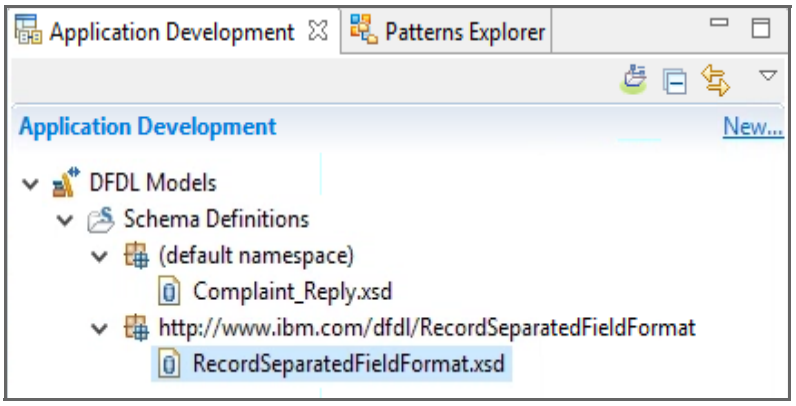
- Clear **The first record is a header** option under **Record settings**.
- On the **Body fields** tab, clear the **Record initiator** field and set **Number of fields** to 3
- On the **Trailer fields** tab, clear the **Trailer initiator** field and set **Number of fields** to 2
- Under **Field settings**,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

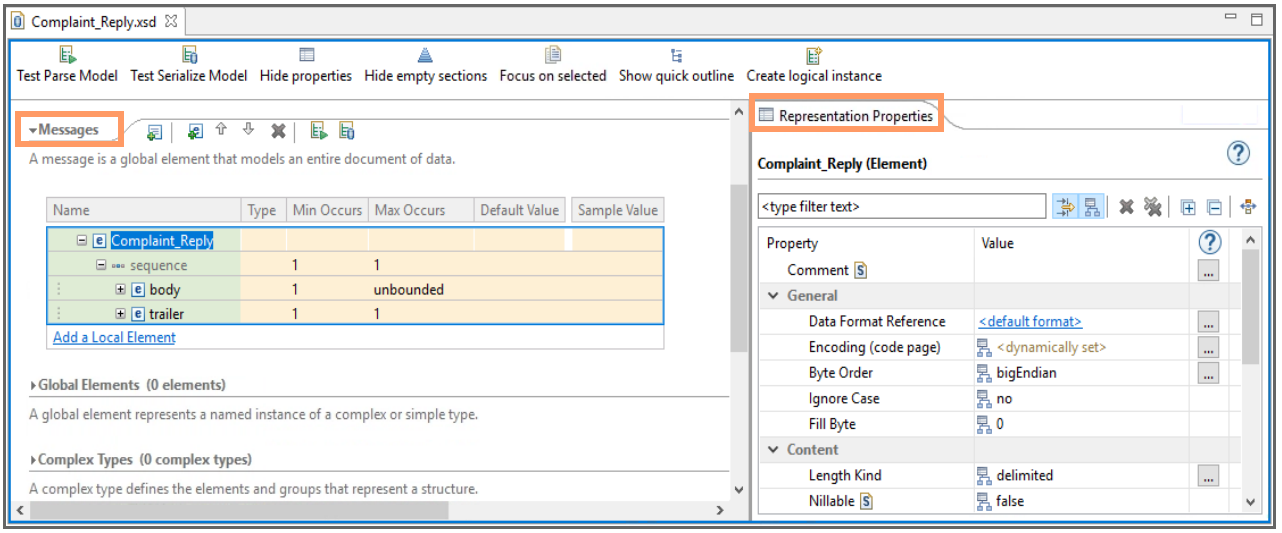
The wizard creates a DFDL schema definition file that is named `Complaint_Reply.xsd` and opens the DFDL schema editor.

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

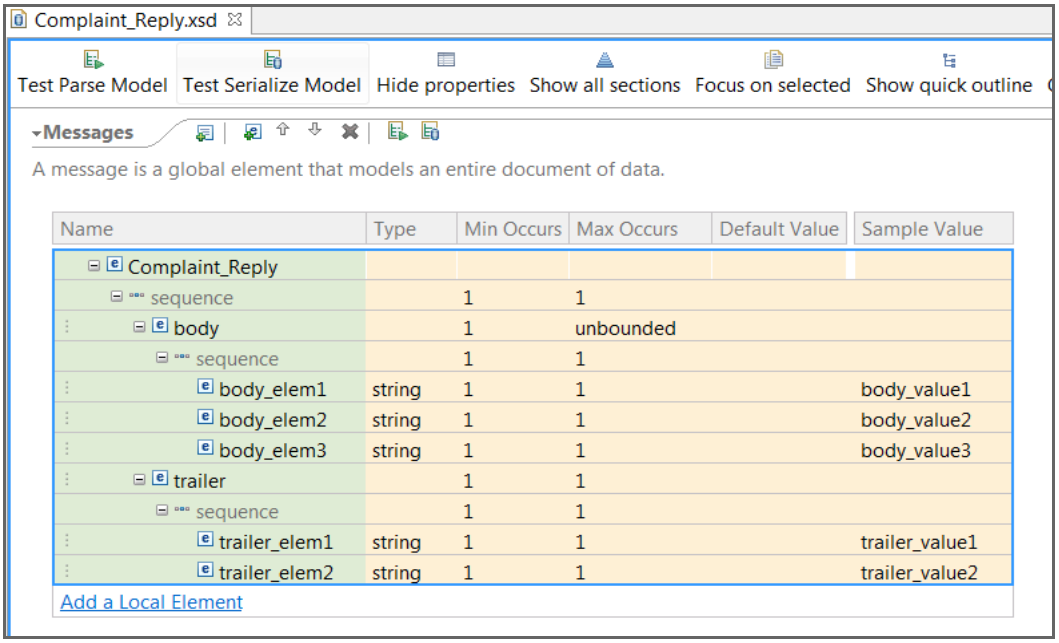


__ 3. Model the logical structure and physical properties of the complaint reply message.

__ a. 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.



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



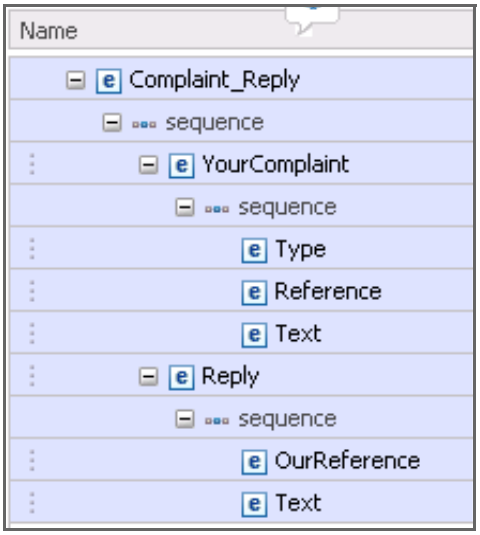
Reminder

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.

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

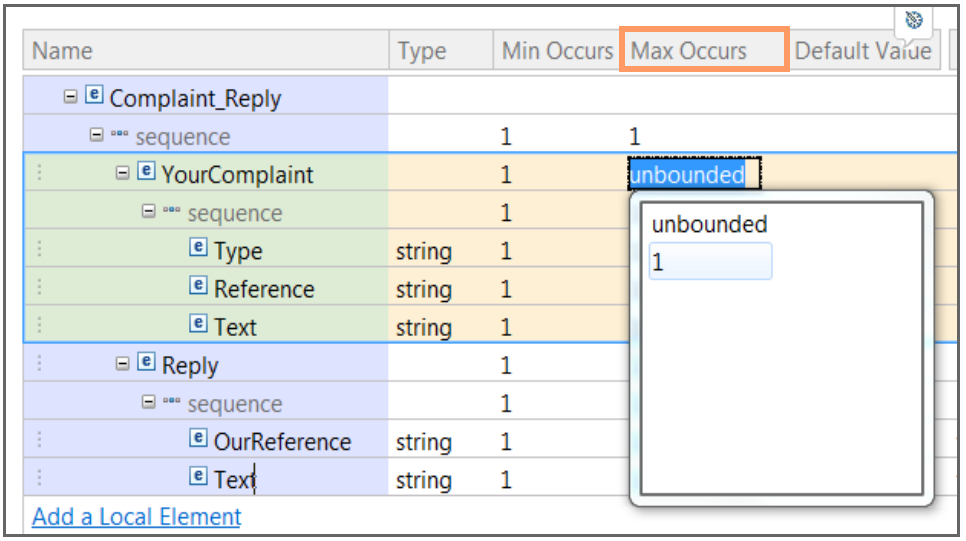
- Rename **body** to **YourComplaint**
- Rename **body_elem1** to **Type**
- Rename **body_elem2** to **Reference**
- Rename **body_elem3** to **Text**
- Rename **trailer** to **Reply**
- Rename **trailer_elem1** to **OurReference**

- Rename **trailer_elem2** to **Text**



__ d. The Complaint_Reply message contains only one complaint.

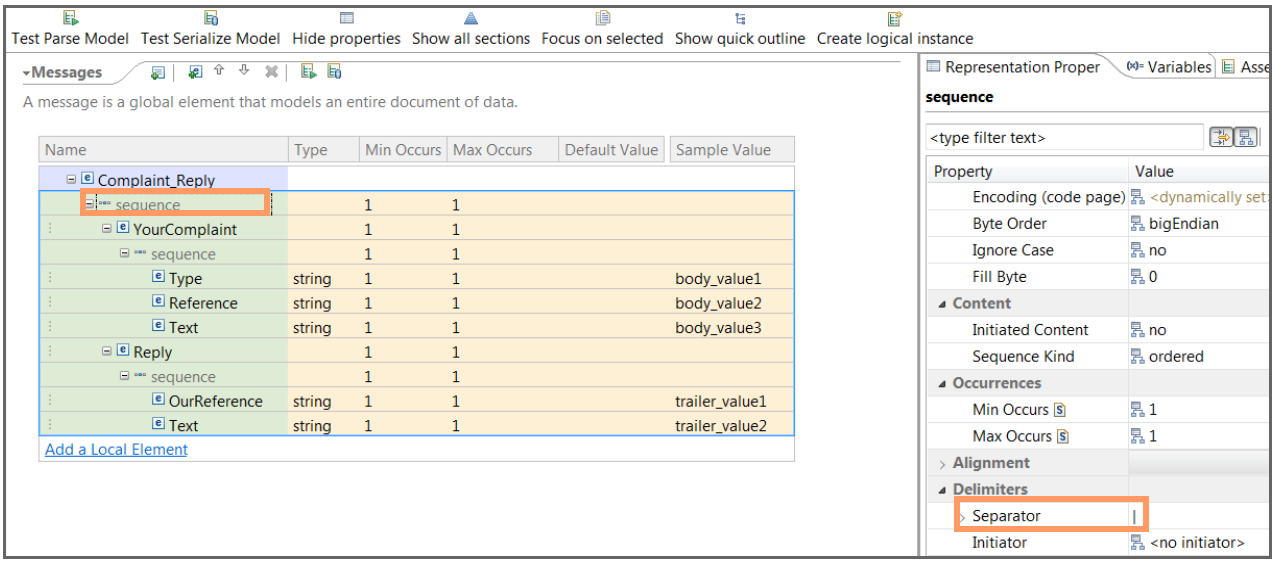
Change the **Max Occurs** value for **YourComplaint** from **unbounded** to **1** by clicking the **Max Occurs** field and selecting **1**.



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

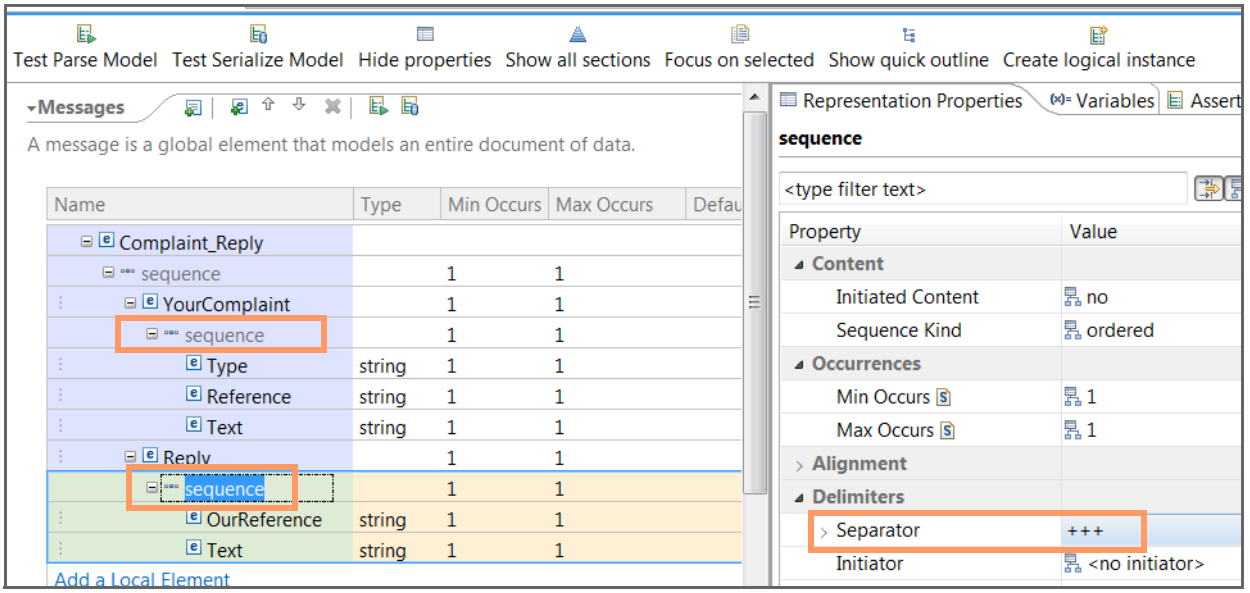
Configure the separator between **YourComplaint** and **Reply** by selecting **sequence** element under **Complaint_Reply** and changing the **Delimiters > Separator** property to **|** (vertical pipe symbol).

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



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

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



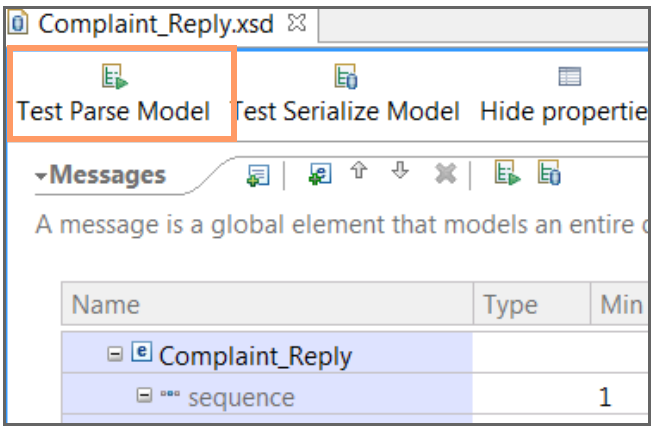
__ g. Save the `Complaint_Reply.xsd` DFDL schema file.

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

Part 3: 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**.



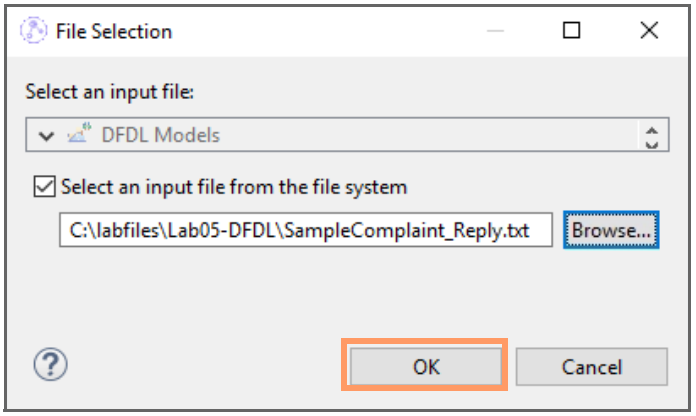
__ b. In the **Test Parse Model** window, select **Content from data file**.

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

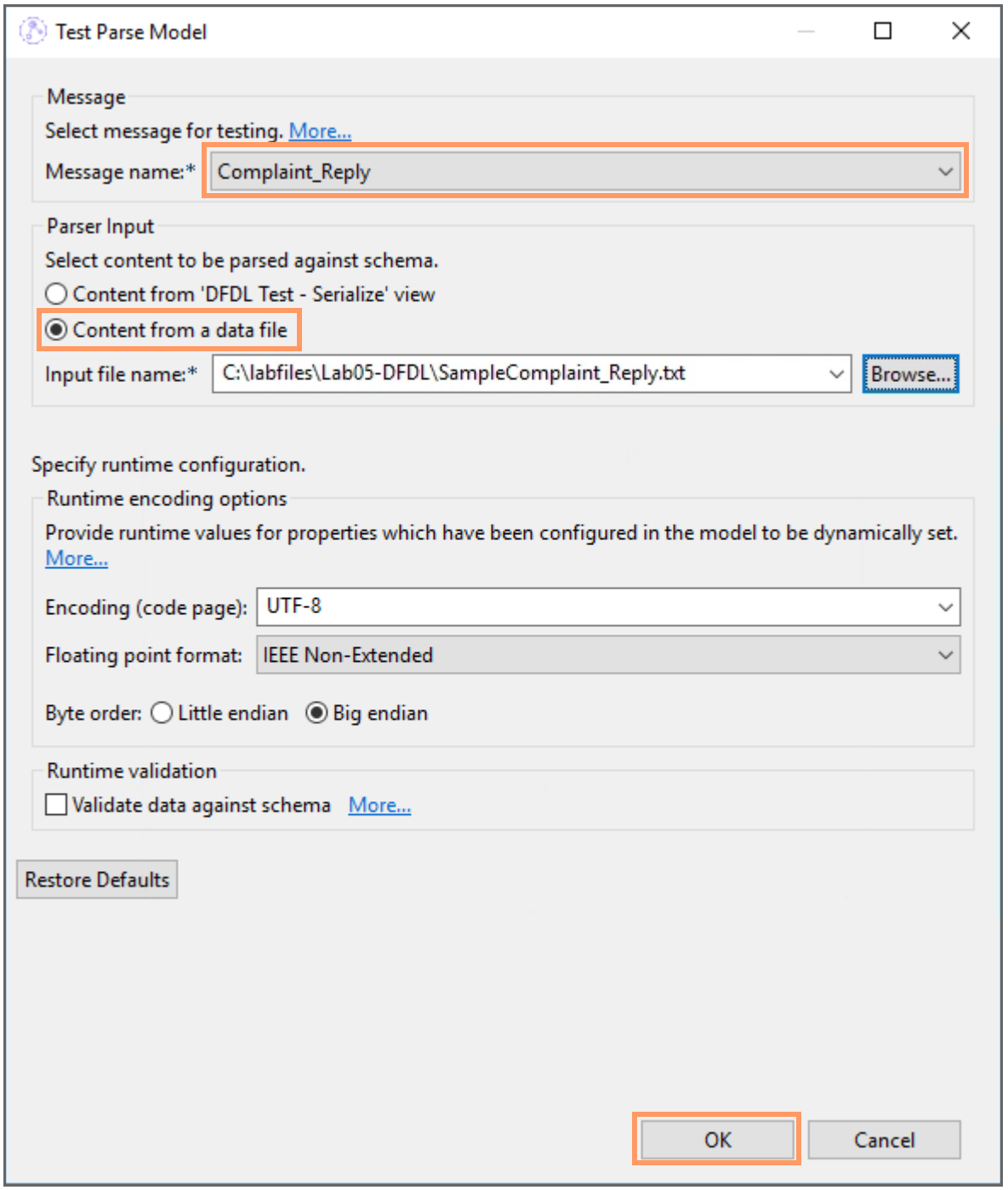
__ d. In the **File Selection** window, click **Select an input file from the file system** and then click **Browse**.

__ 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.



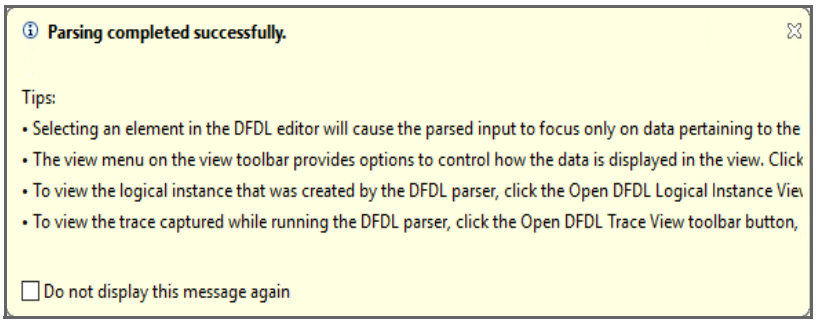
__ g. Click **OK** in the **Test Parse Model** window.



__ 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.

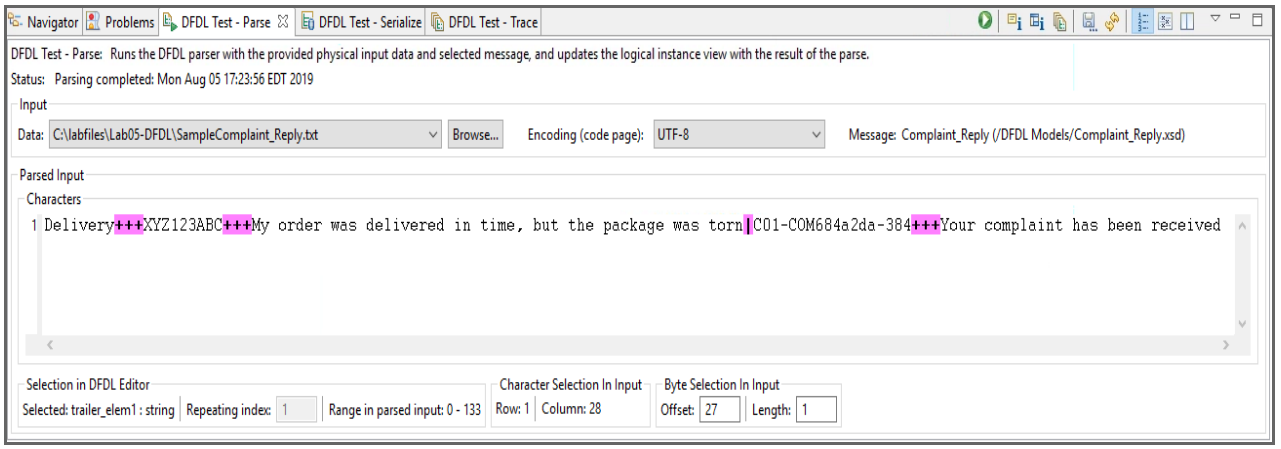
The message “Parsing completed successfully” is displayed if the DFDL parser can parse the sample data file.



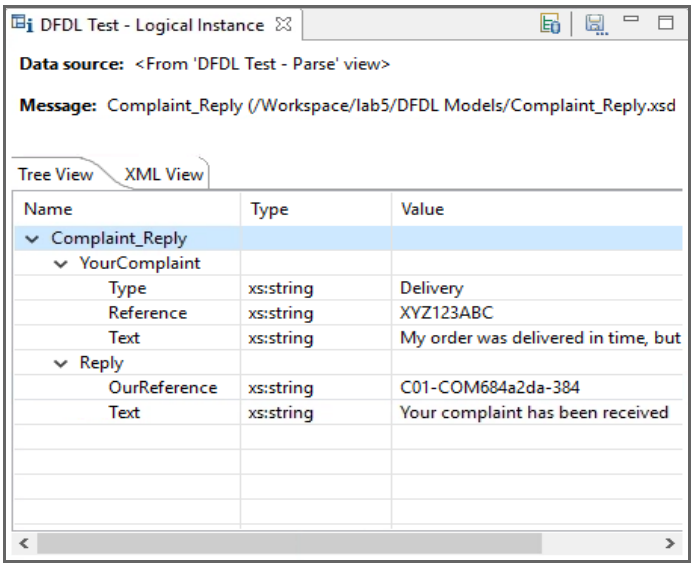
If parsing fails, an error message that describes the cause of the failure is displayed. Review the error message. If necessary, modify the DFDL Properties, save the model, and rerun the test until the Test Parse succeeds.

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

__ k. 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.



__ l. 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.



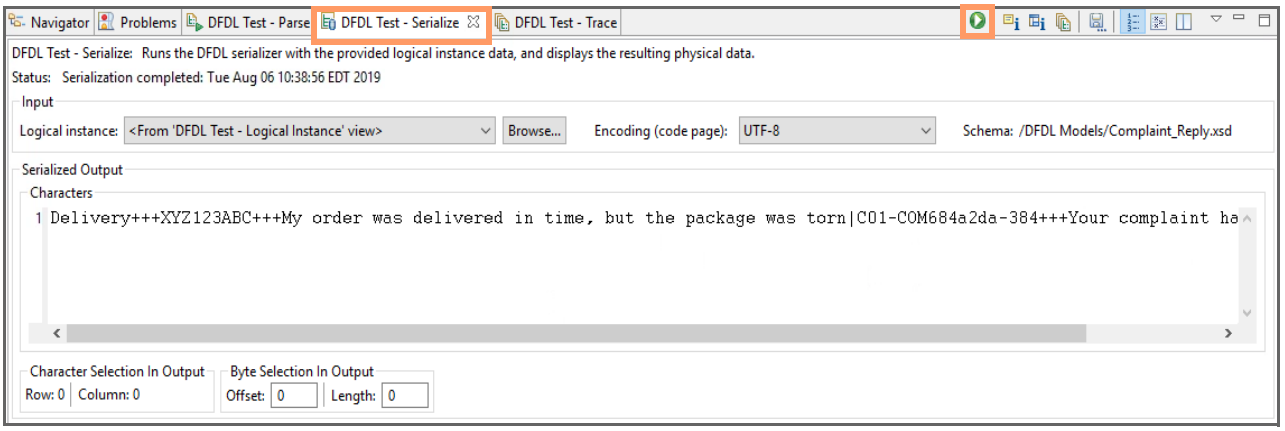
Compare this result to the data description in the lab exercise Introduction to ensure that the message was parsed correctly.

__ 2. 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).

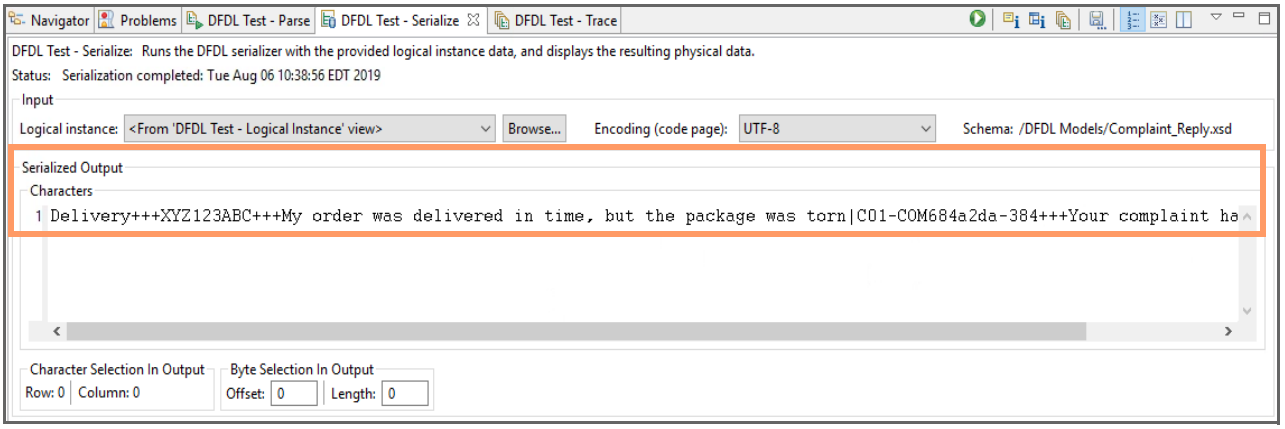
__ b. For the **Logical instance**, select **<From 'DFDL Test - Logical Instance View'>**

__ 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.

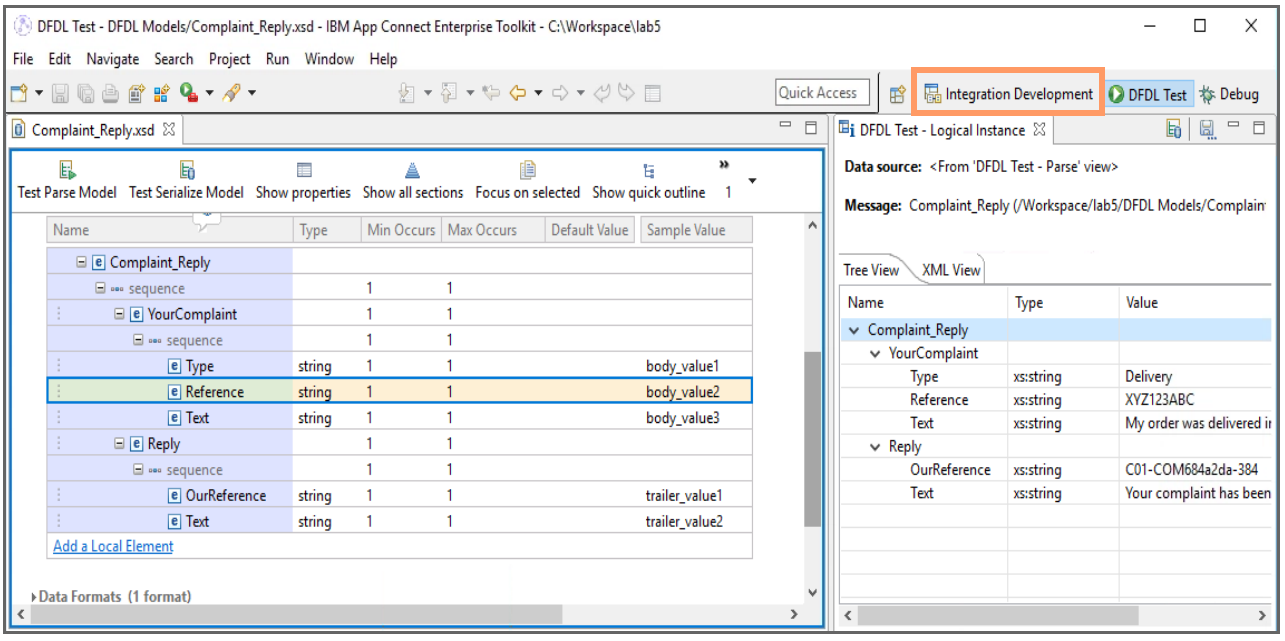
__ d. Verify the output data that the DFDL Serializer generated. It should match the original sample input file `SampleComplaint_Reply.txt`.



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.

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



__ f. 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 then 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

Estimated time

01:00

Overview

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.

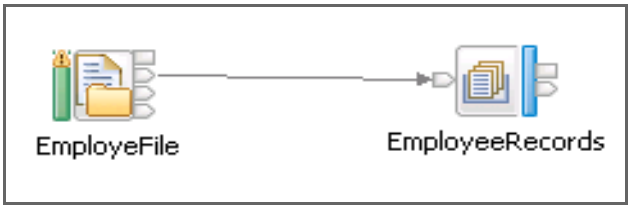
Objectives

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 App Connect Enterprise 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 App Connect Enterprise V11 Toolkit and IBM MQ V9

- A local IBM MQ queue manager that is named ACEQM with the following local queues: DLQ, RECORDS
- Lab files in the `C:\labfiles\Lab06-Files` directory
- The user `aceadmin` is a member of the “mqm” group

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab06`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

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

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

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

__ 2. Verify the node and server are running.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 3. This exercise requires an IBM MQ queue manager.

If you completed Exercise 3, you created a queue manager that is named ACEQM. You can use that queue manager in this exercise. Proceed to the next step.

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

__ a. Start IBM MQ Explorer.

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

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

__ 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.

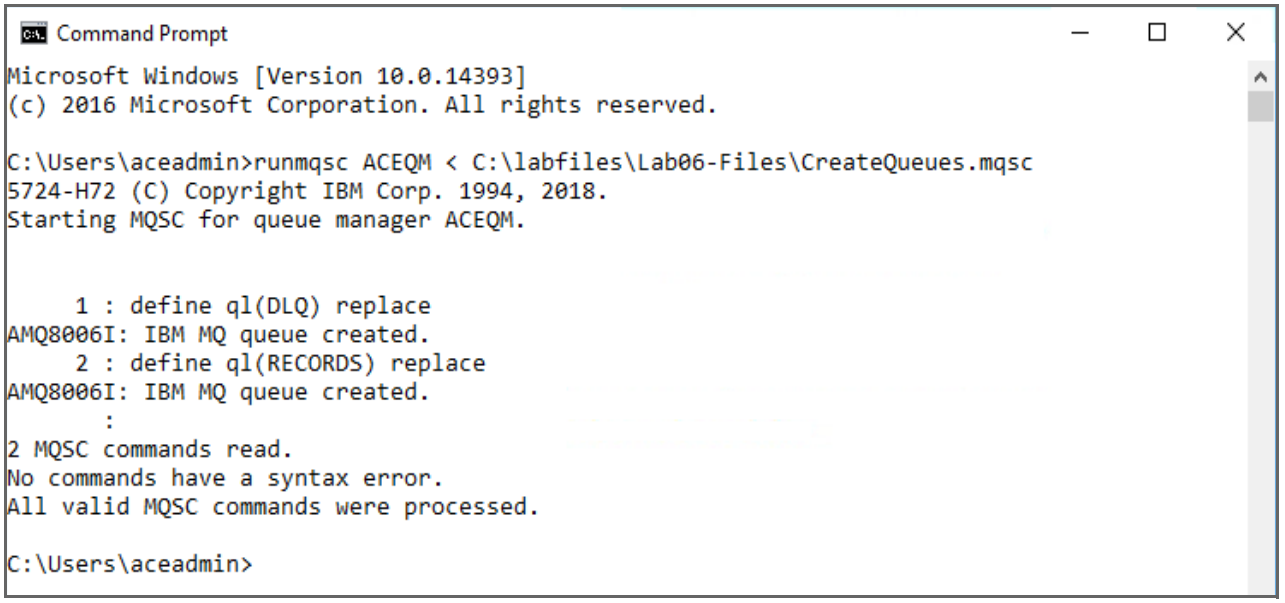
The queue manager is shown under the **Queue Managers** folder in the **MQ Explorer - Navigator** view.

__ 4. 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, enter:

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



__ 5. Use IBM MQ Explorer to verify that the queues were created.

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

Information

You might have other queues for this queue manager from a previous exercise. Their presence does not affect the lab exercise.

Part 2: 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 App Connect Enterprise Toolkit **Application Development** pane, click **New Application**.

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

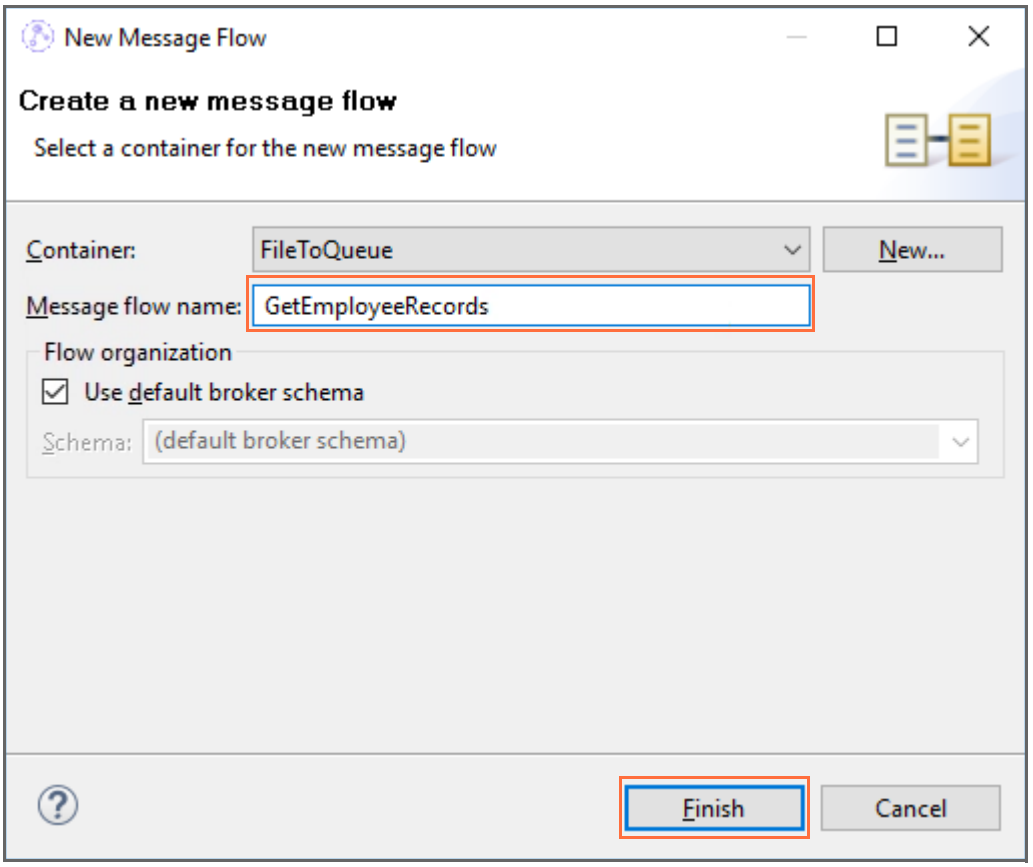
__ c. Click Finish

__ 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 nodes to the message flow.

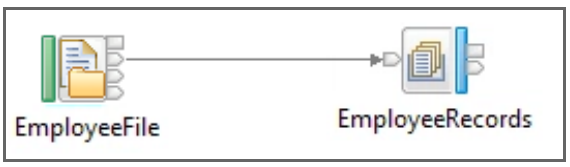
__ a. Add a FileInput node that is named **EmployeeFile** to the message flow. Click the **FileInput** node in the Message Flow Editor Palette **File** drawer and then click in the message flow editor canvas.

__ b. Rename the node to `EmployeeFile`

__ c. Add an MQOutput node that is named **EmployeeRecords** to the message flow. Click the **MQOutput** node in the Message flow Editor Palette **IBM MQ** drawer and then click in the message flow editor canvas.

__ d. Rename the node to `EmployeeRecords`.

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



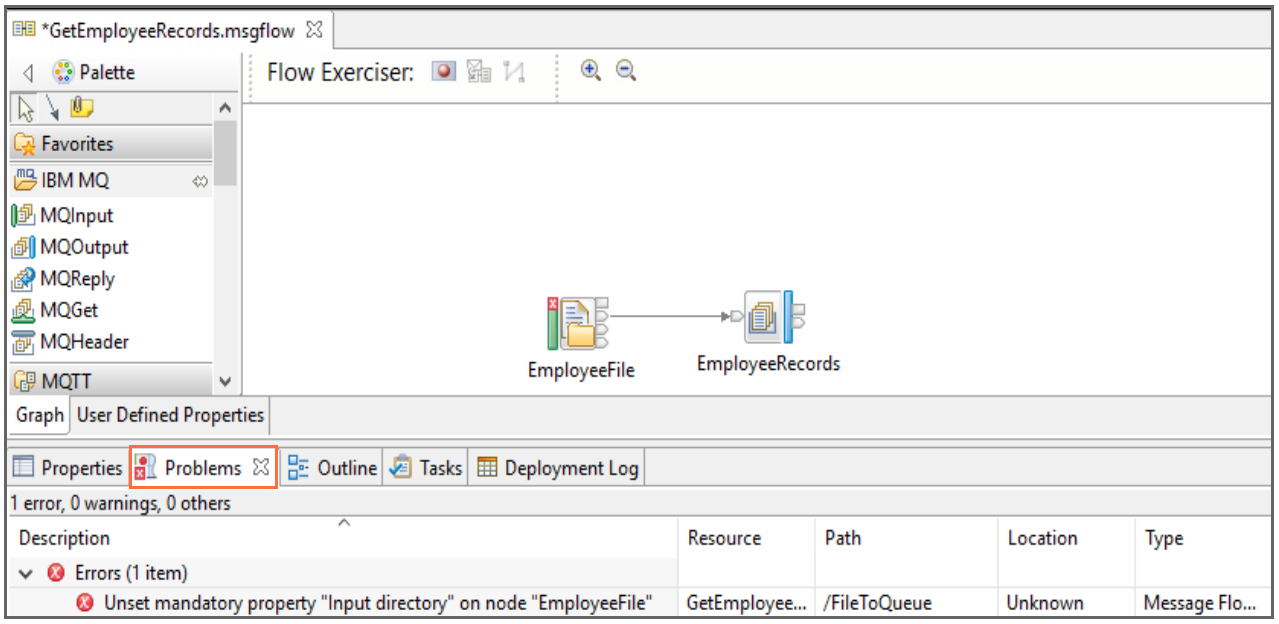
__ 4. Define properties for the MQOutput node that is named **EmployeeRecords**.

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

__ b. For the **Destination queue manager name** on the **MQ Connections** tab, enter `ACEQM`

__ c. Save the message flow.

You should notice an error 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 of this exercise.

Part 3: 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 App Connect Enterprise Toolkit, click **File > Import**.

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

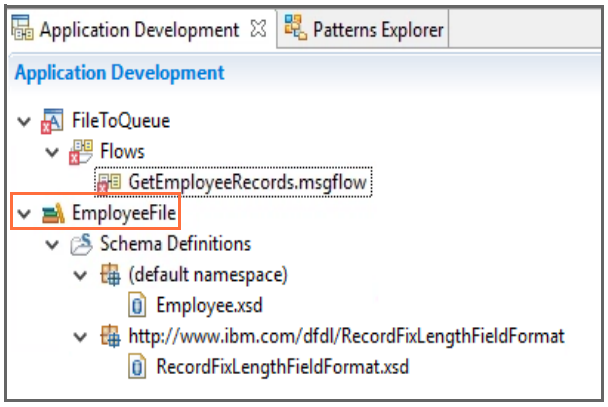
__ c. Click **Browse** next to **From .zip file**.

__ 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**.

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

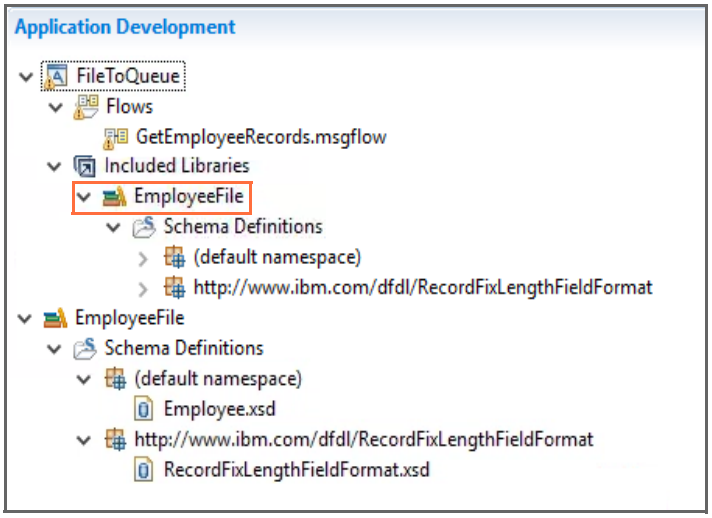


__ 2. Change the **FileToQueue** application to reference the **EmployeeFile** library.

__ 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 4: Configure the FileInput node

__ 1. Set the **Basic** properties.

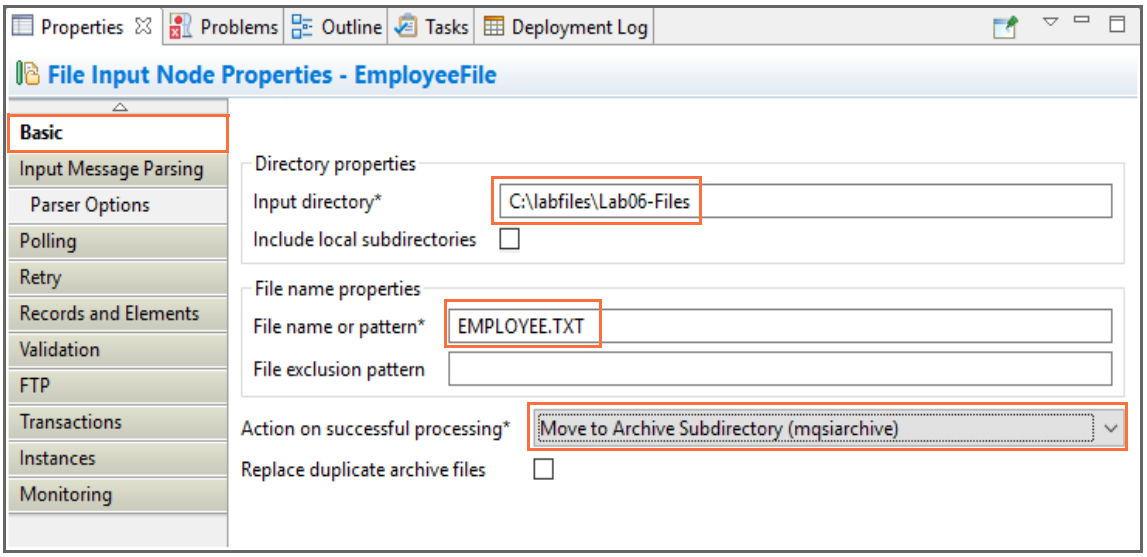
__ a. Click the FileInput node (named **EmployeeFile**) in the Message Flow editor to display the node properties.

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

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

If you get a warning, you can ignore it for now.

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

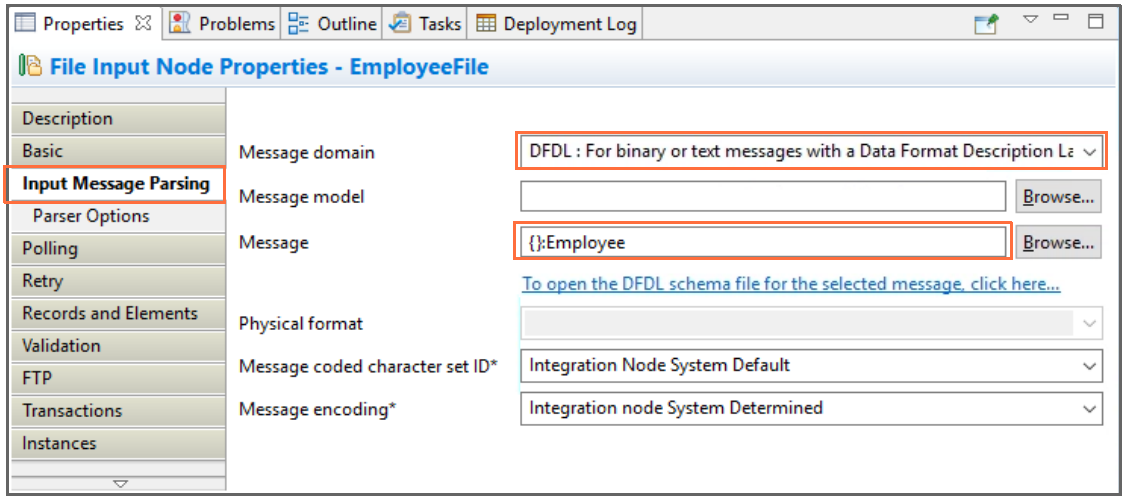


__ e. Save the message flow. The Problems pane should show no errors now that the input directory is configured for the input node.

__ 2. Set the **Input Message Parsing** properties to use the Employee message model.

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

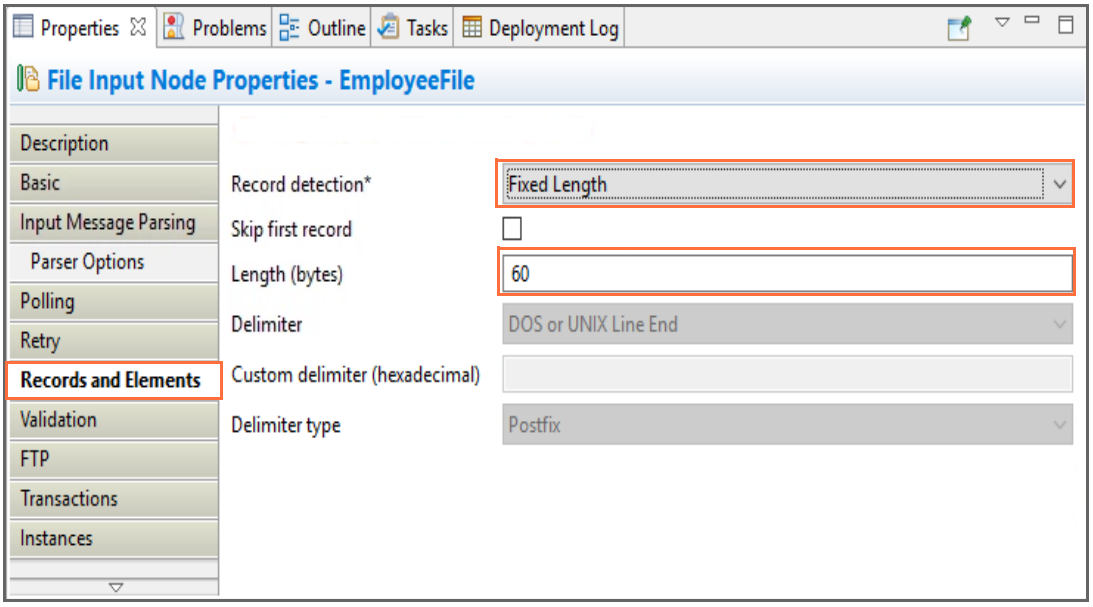
__ b. For **Message** property, click **Browse**, click **Employee {}**, and then click **OK**.



__ 3. Set the **Records and Elements** properties.

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

__ b. For the **Length (bytes)** property, change the value to **60**.



__ c. Save the changes to the message flow.

Information

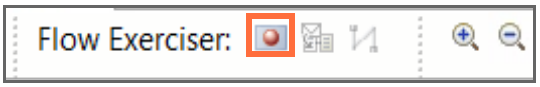
If you see a Warning related the EmployeeFile configuration in the Problems view, you can ignore it.

Part 5: Test the message flow

In this part of the exercise, you use the IBM App Connect Enterprise 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 App Connect Enterprise Toolkit Message Flow editor, start the Flow exerciser to create the BAR file, deploy the application to **server1** on **ACENODE1**, and start recording.

__ a. Click the **Start flow exerciser** icon at the top of the Message Flow editor.



__ b. Click **OK** on the Confirm dialog box.

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

__ d. In the Integration Explorer view, expand the **ACENODE1** > **server1** to display the contents. Verify that the **GetEmployeeRecords** message flow is in record mode (signified by the Recording icon).

__ e. 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.

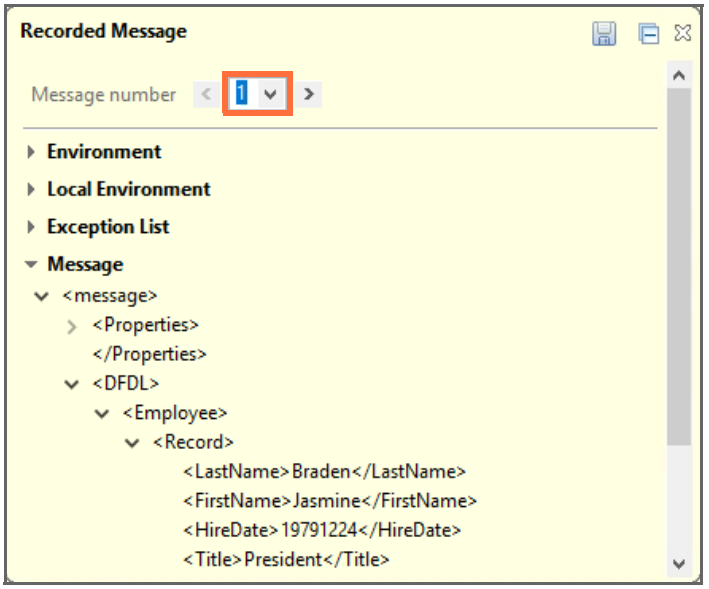
If the flow failed, a copy of the file is written to the `C:\labfiles\Lab06-Files\mqsibackout` directory.

__ 2. Verify the data in the Flow exerciser

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

__ b. 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.

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



__ d. Close the message window.

__ 3. Verify the data in MQ Explorer

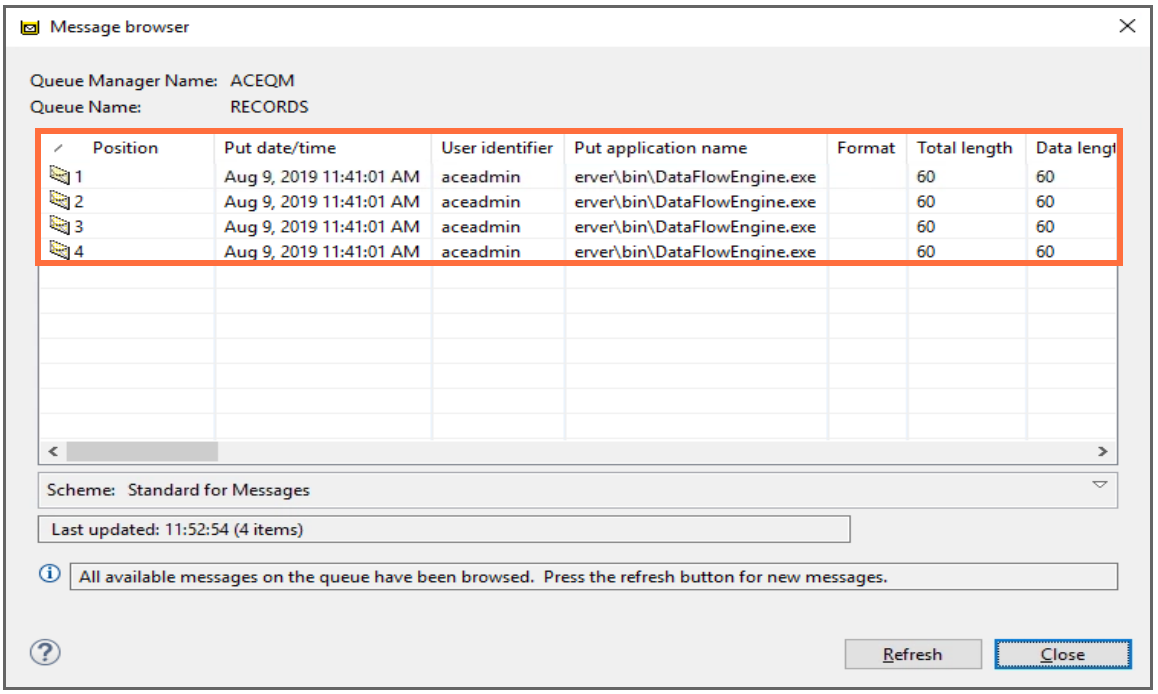
__ a. Open IBM MQ Explorer to check the RECORDS queue for messages.

__ b. Expand the **ACEQM** queue manager folder.

__ c. Double-click the **Queues** folder.

__ d. Right-click the RECORDS queue and select **Browse Messages...**

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



__ e. Click **Close**.

Part 6: Exercise clean-up

__ 1. In the IBM App Connect Enterprise Toolkit **Integration Explorer view**, stop recording on the message flow.

__ 2. Delete all resources from **server1** integration server on **ACENODE1**.

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

__ 4. In IBM MQ Explorer, delete the queues

__ a. Select all queues.

__ b. Right-click and select **Delete...**

__ c. When asked if you are sure that you want to delete the two selected objects, click **Delete**.

__ d. When asked to clear messages, click the checkbox to **Clear all messages from the queue**. Click **Delete**.

All queues are deleted.

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 App Connect Enterprise 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 App Connect Enterprise Toolkit Flow exerciser to view multiple output messages

Exercise 7. Using problem determination tools

Estimated time

01:30

Overview

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.

Objectives

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 App Connect Enterprise Toolkit Test Client and message flow debugger view to step through a message flow application
- Examine the IBM App Connect Enterprise logs and system logs to diagnose problems

Introduction

In some cases, the IBM App Connect Enterprise 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 file 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 App Connect Enterprise 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 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 App Connect Enterprise 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 App Connect Enterprise V11 App Connect Enterprise Toolkit and IBM MQ V9
- A local IBM MQ queue manager that is named ACEQM with the following local queues: DLQ, COBOL_IN, and XML_OUT
- Lab files in the `C:\labfiles\Lab07-PDTools` directory
- rfhutil.exe in the `C:\mq-rfhutil-master\bin\Release` directory
- User `aceadmin` that is a member of the “mqm” group

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab07`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

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

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

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

__ 2. Verify that the node and server are running.

You may need to refresh the view to see the ACENODE1 node.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 3. This exercise requires an IBM MQ queue manager.

If you completed Exercise 3, you created a queue manager that is named ACEQM. You can use that queue manager in this exercise. Proceed to the next step.

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

__ a. Start IBM MQ Explorer.

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

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

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

Important

This exercise requires the ACEQM queue manager with a dead-letter queue named 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.

__ 4. 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:

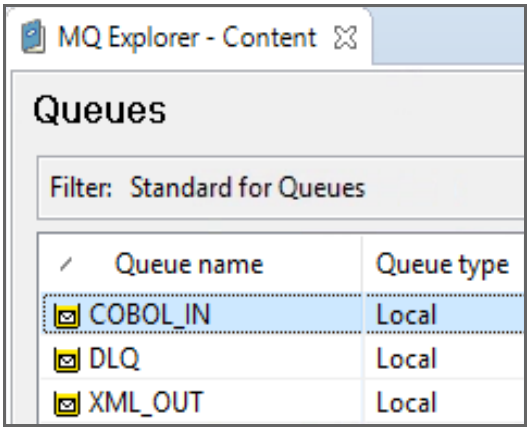
```
runmqsc ACEQM < C:\labfiles\Lab07-PDTools\CreateQueues.mqsc
```

__ 5. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the **MQ Explorer - Navigator** view, expand the **Queue Managers > ACEQM** 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 2: 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.

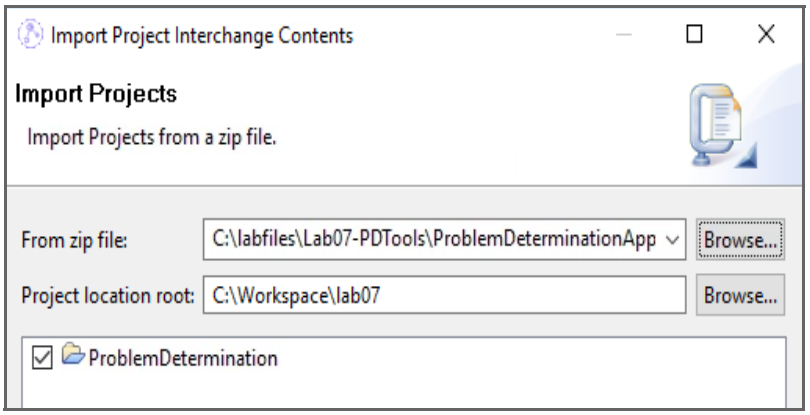
__ a. From the IBM App Connect Enterprise Toolkit, click **File > Import**.

__ 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.

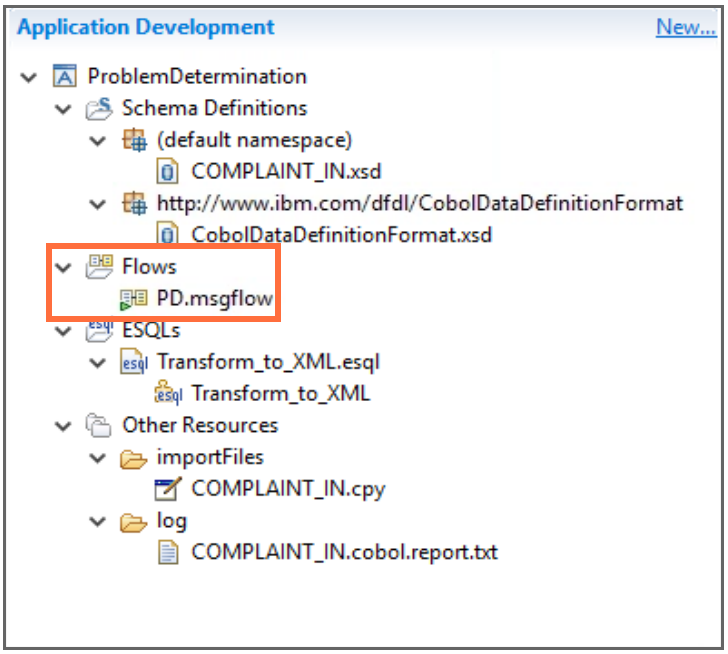
__ e. Select `ProblemDeterminationApp_PI.zip`, and then click **Open**. The **Import Project Interchange Contents** window is displayed.



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

__ 2. Add a Trace node between the COBOL_IN MQInput node and the Transform_to_XML Compute node.

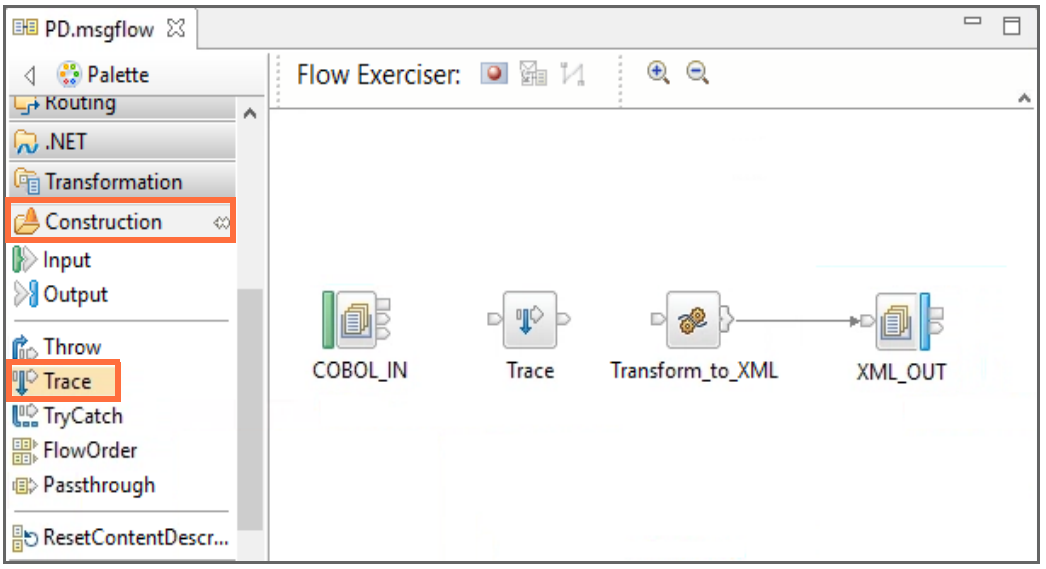
__ a. In the **Application Development** view, expand **Problem Determination > Flows**.



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

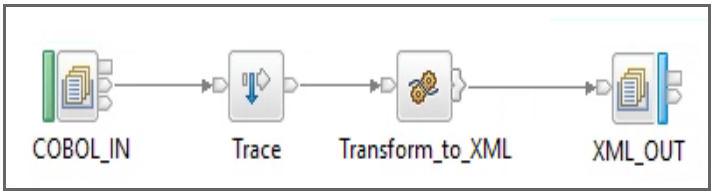
__ c. Delete the connection between the COBOL_IN MQInput node and the Transform_to_XML Compute node by selecting the connection and clicking **Delete**.

__ d. 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.



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

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



__ 3. Configure the Trace node properties to output a time stamp and the entire root tree to a file.

__ a. 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`

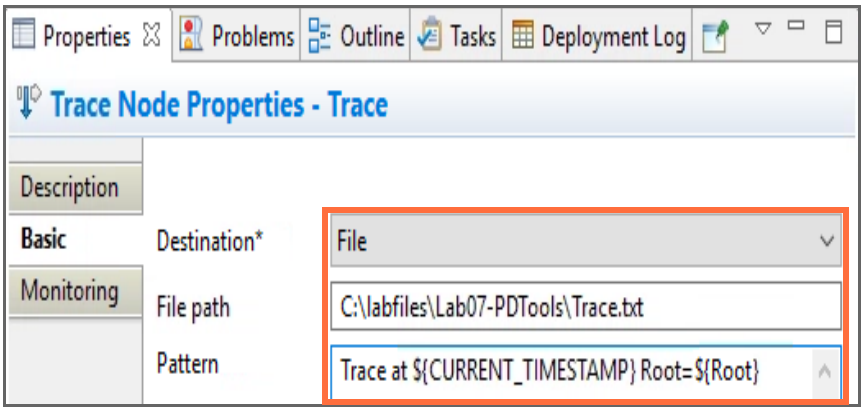
Important

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.



__ 4. 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: `ACEQM`

__ 5. 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: `ACEQM`

__ c. Save the message flow.

__ 6. 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.

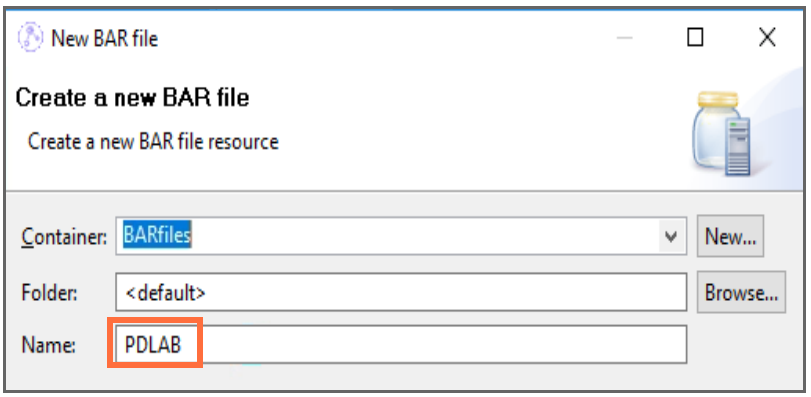
If any errors are indicated, you can review them in the **Problems** view. Correct any errors and then save and verify that no more errors are indicated in the project.

In most cases, warning messages can be ignored.

__ 7. Create a BAR file that is named **`PDLAB.bar`**.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > New > BAR file**.

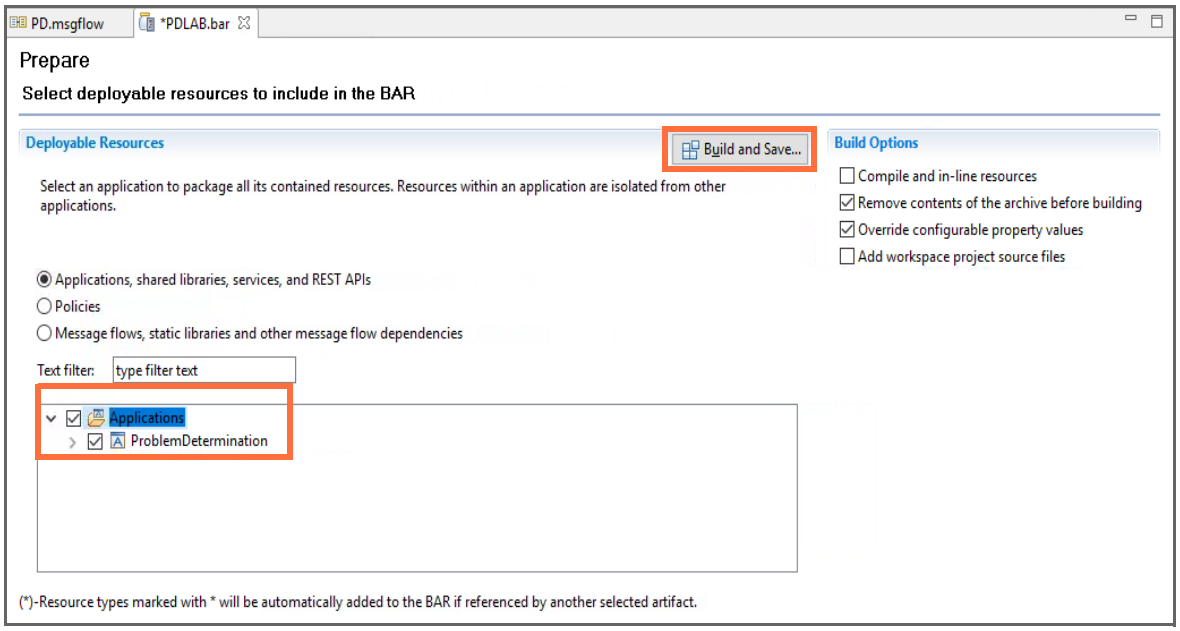
__ b. For the **Name** of the BAR file, type: `PDLAB`



__ 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.

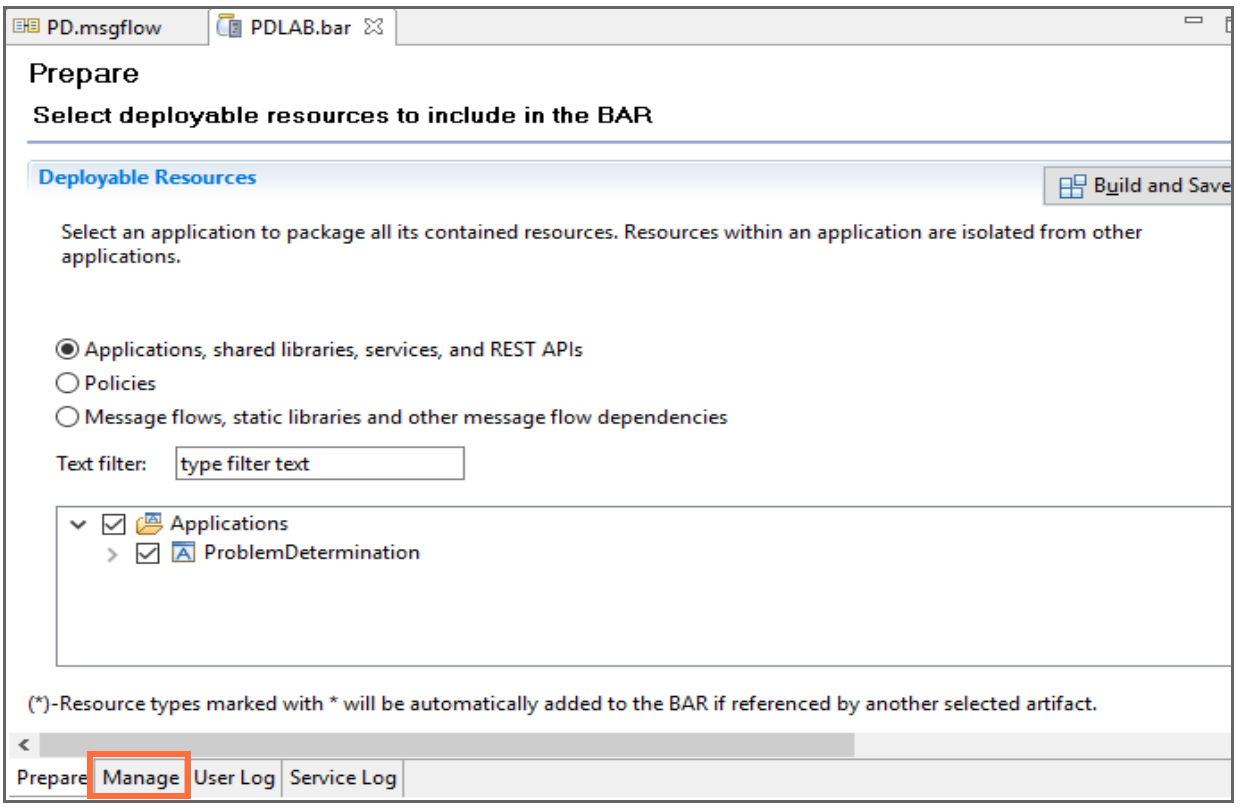
__ e. Click **Build and Save**. The BAR file is built.



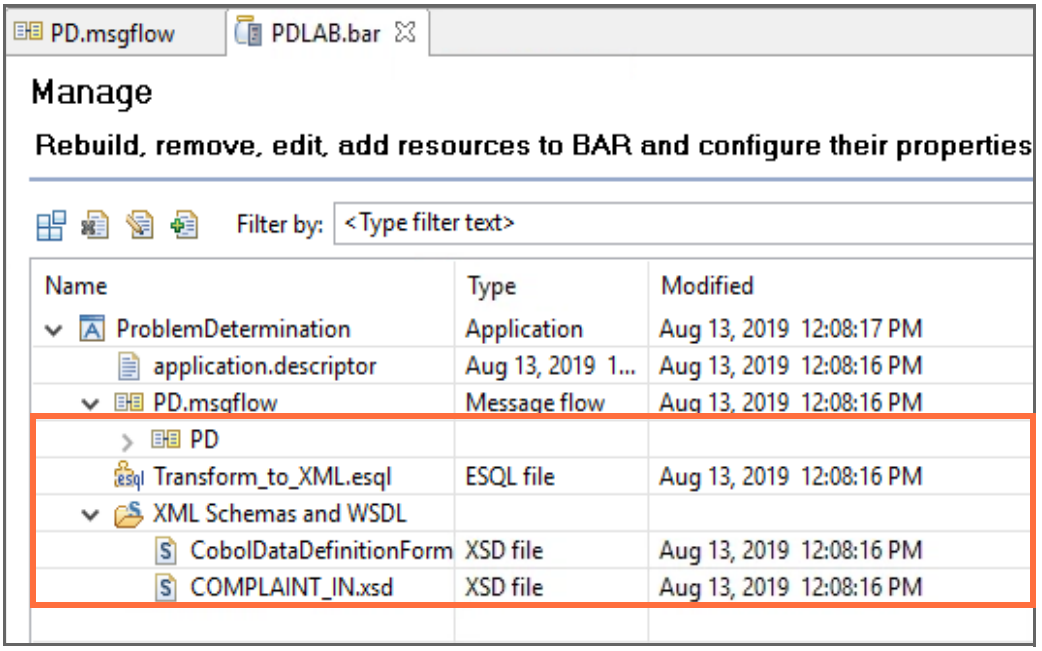
__ f. Click **OK** on the **Operation completed successfully** message.

__ 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.

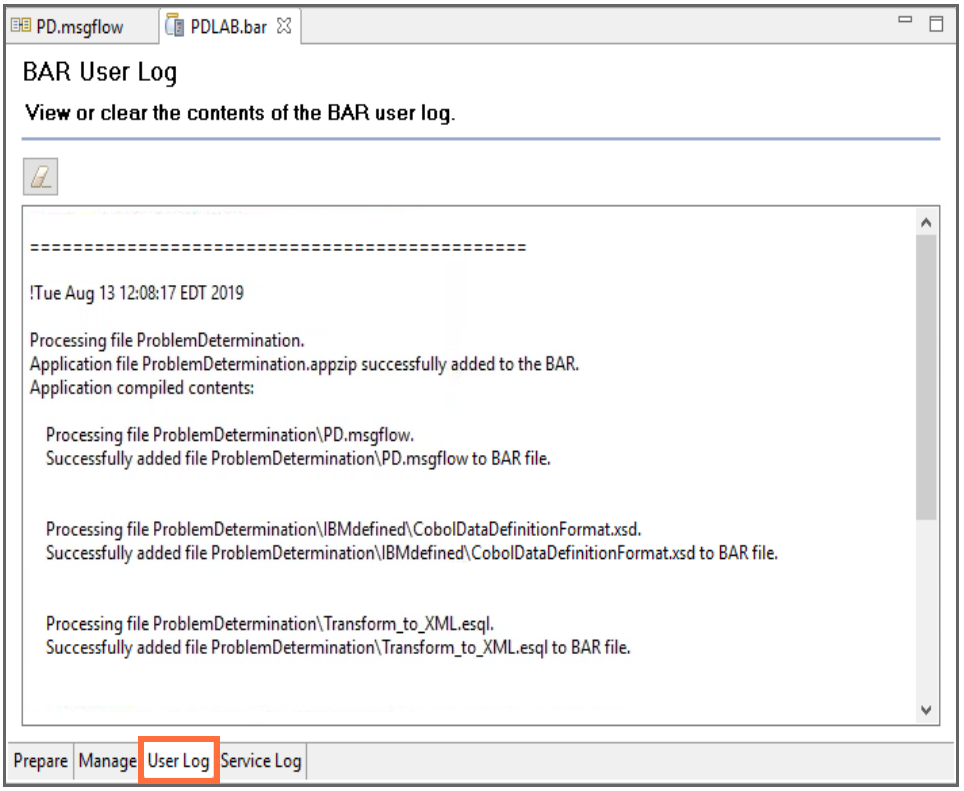


__ i. Verify that the `PD.msgflow`, `Transform_to_XML.esql`, `CobolDataDefinitionFormat.xsd`, and `COMPLAINT_IN.xsd` files are included in the BAR file and that they show current time stamps.



__ 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.

__ 8. Deploy the BAR file to **server1** integration server on ACENODE1.

__ a. In the Integration Explorer view, verify that the integration node **ACENODE1** is connected and running.

__ b. Verify that **server1** is running on ACENODE1.

__ c. If the integration node or server are not running, right-click them and then click **Start**.

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

__ e. Right-click the PDLAB.bar file under the Independent Resources folder, and then click **Deploy**.

The Deploy window is displayed with **server1** selected.

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 Explorer view**.

__ f. Click Finish.

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

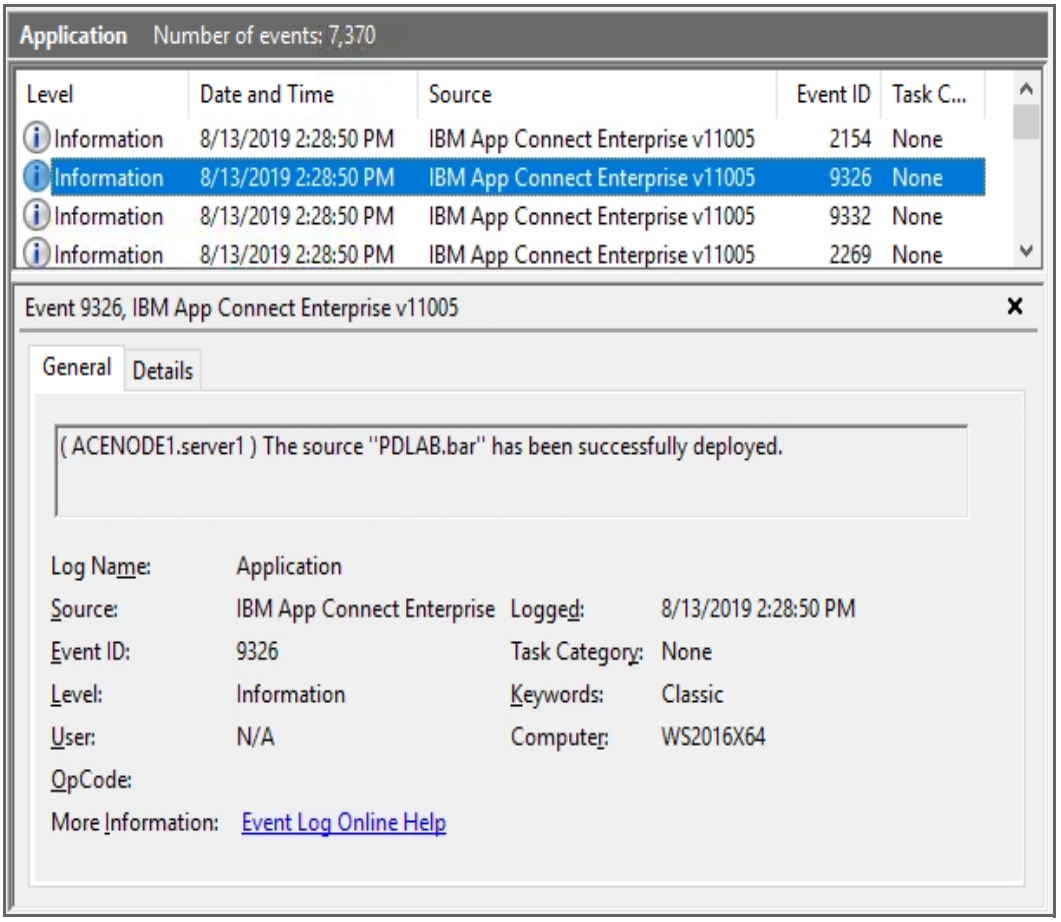
__ 9. Check the local error log by using the Windows Event Viewer.

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

__ b. Double-click **Event Viewer**.

__ c. Expand **Windows Logs**.

__ d. Select **Application**. Step through the messages and look at the deployment messages for IBM App Connect Enterprise. Verify that the level for the deployment messages is **Information** and not **Error**.



__ e. Minimize the Event Viewer.

__ 10. 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 directory: `C:\mq-rfhutil-master\bin\Release` and then double-clicking `rfhutil.exe`.

RfhUtil opens on the **Main** tab.

__ b. Click **Run** in the **Open File - Security Warning** message.

__ c. For **Queue Manager Name**, enter **ACEQM**.

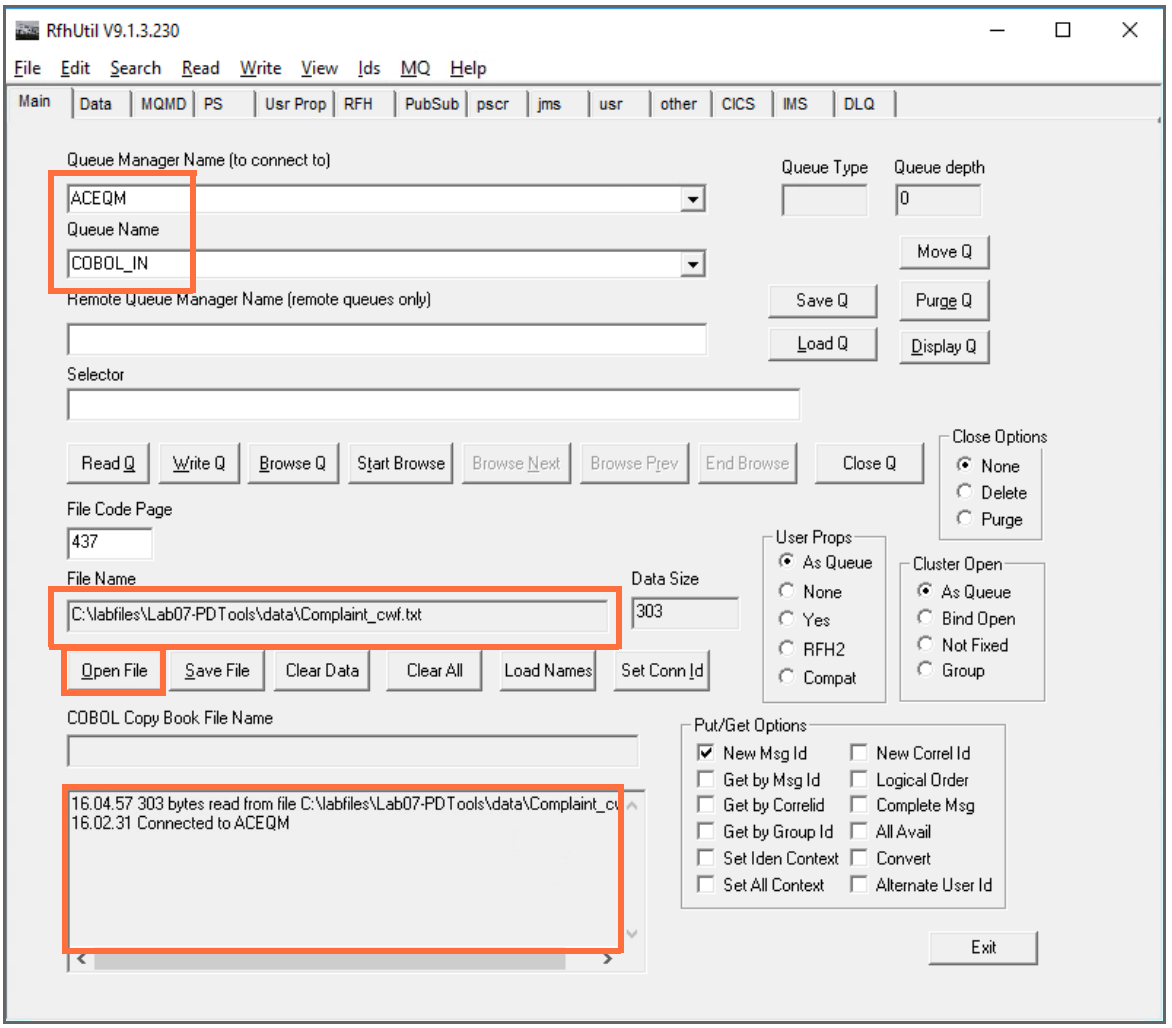
__ d. For **Queue Name**, enter **COBOL_IN**.

__ e. Connect to MQ by navigating to the menu item: **MQ > MQConn**

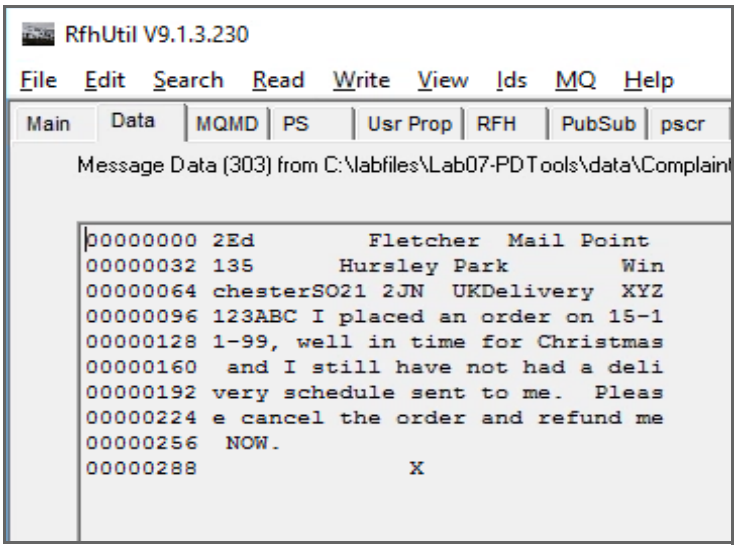
A message appears with a time stamp in the RfhUtil message window: **Connected to ACEQM**.

__ f. Click **Open File** under the File Name and select `C:\labfiles\Lab07-PDTools\data\Complaint_cwf.txt` and click **Open**.

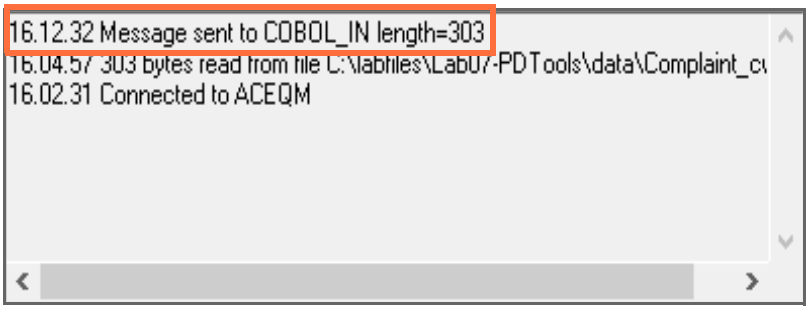
The message window displays a time stamp with information about the data read from the file.



__ g. Use RfhUtil to review the test data. Click the **Data** tab. The data is presented in character format by default.



__ h. 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.



__ 11. Review the output message on queue XML_OUT by using RfhUtil.

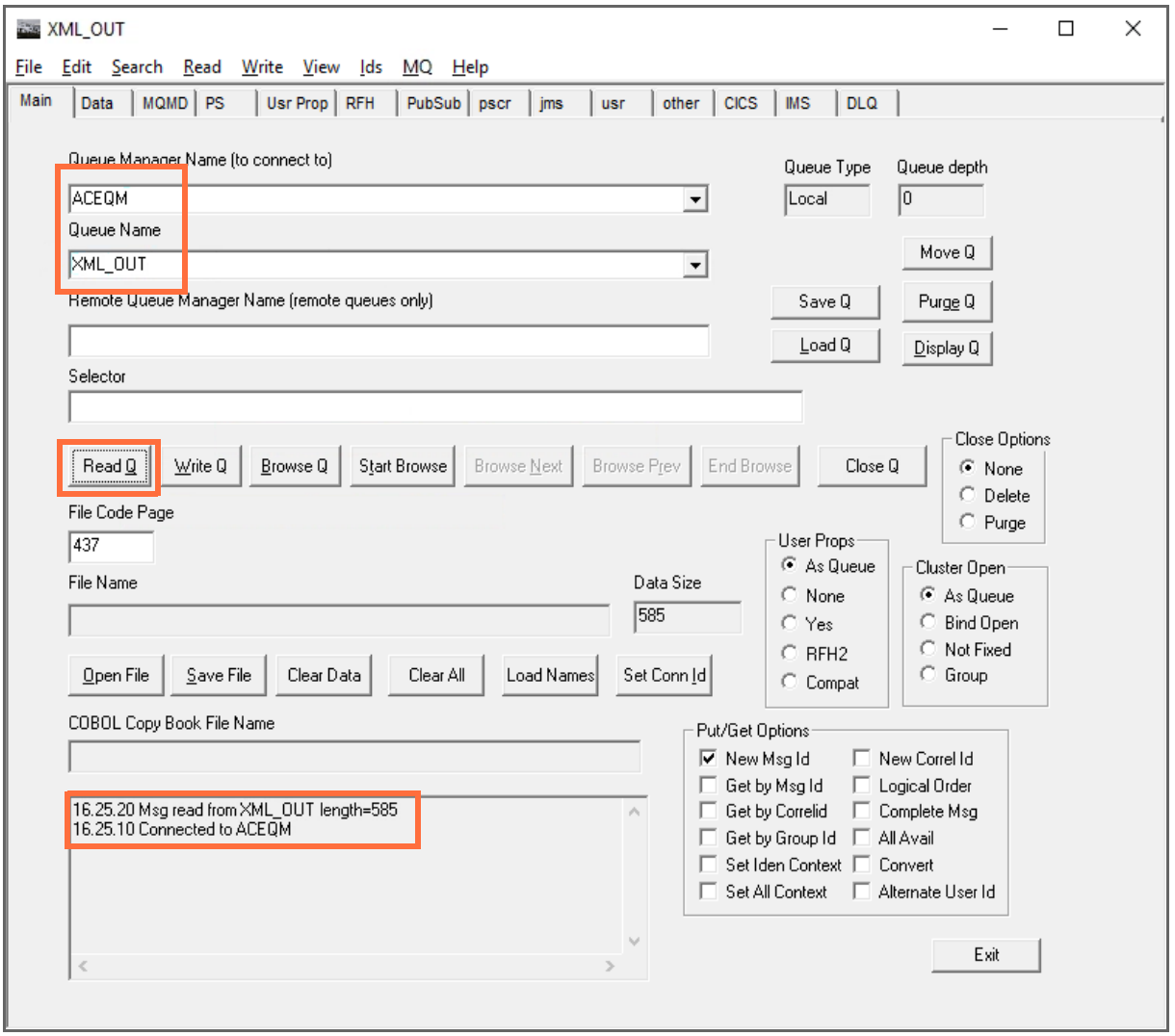
__ a. Start a second instance of RfhUtil.

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

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

__ d. Connect to MQ by navigating to the menu item: **MQ > MQConn**

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

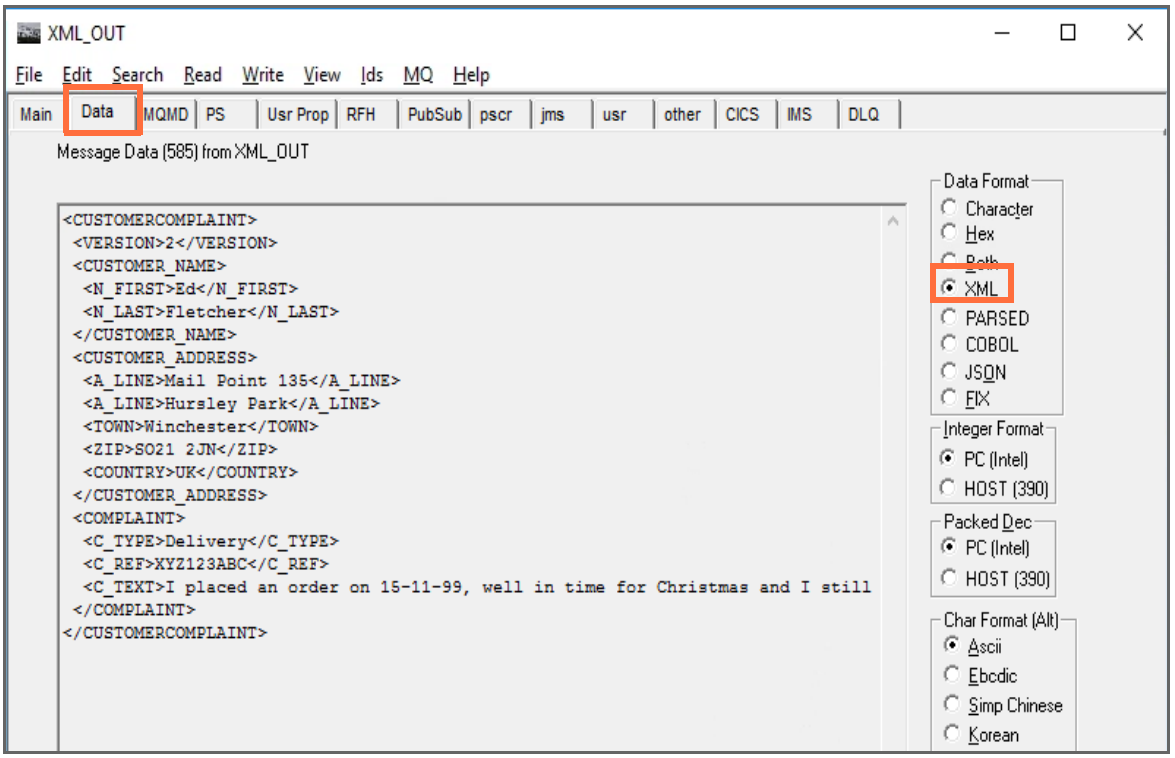


Information

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.

__ f. 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.



__ 12. Review the output from the 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 2019-08-13 15:36:18.428721 Root=( ['MQROOT' : 0x19de07b4ce0]
(0x01000000:Name):Properties = ( ['MQPROPERTYPARSER' : 0x19ddc21ff60]
(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 '2019-08-13
19:36:18.460' (GMTTIMESTAMP)
(0x03000000:NameValue):ExpirationTime          = -1 (INTEGER)
(0x03000000:NameValue):Priority                = 0 (INTEGER)
(0x03000000:NameValue):ReplyIdentifier         =
X'000000000000000000000000000000000000000000000000' (BLOB)
(0x03000000:NameValue):ReplyProtocol           = 'MQ' (CHARACTER)
```

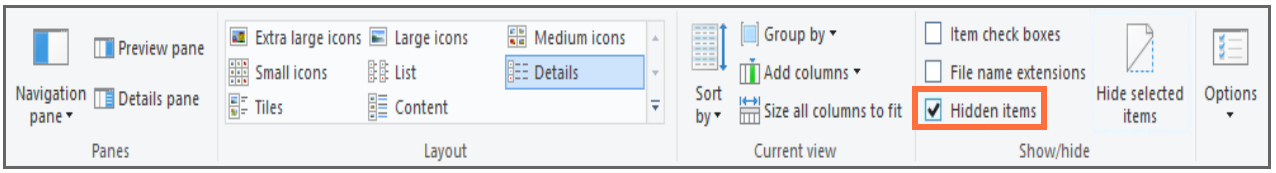
__ 13. The Trace node requires extra system resources. Now that you are finished using the Trace node, turn it off.

You can enable and disable Trace nodes by setting properties in the `server.conf.yaml` configuration file. The location of the `server.conf.yaml` file that you need to modify depends on whether you are configuring an independent integration server or an integration server that is managed by an integration node.

__ a. Using Windows Explorer, navigate to the following directory:
`C:\ProgramData\IBM\MQSI\components\ACENODE1\servers\server1`

Information

ProgramData is a hidden folder. You might have to configure Windows to display hidden items under the View tab if you do not see the folder.



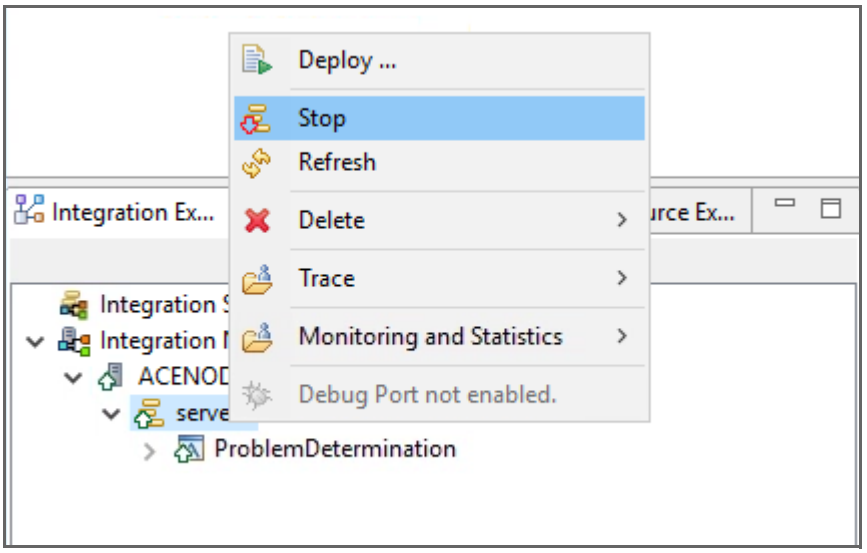
__ b. Open the `server.conf.yaml` file by using **Notepad**.

__ c. Enable the **traceNodeLevel** line by deleting the # character.

__ d. Change the **traceNodeLevel** from `true` to `false`.

__ e. Save the file.

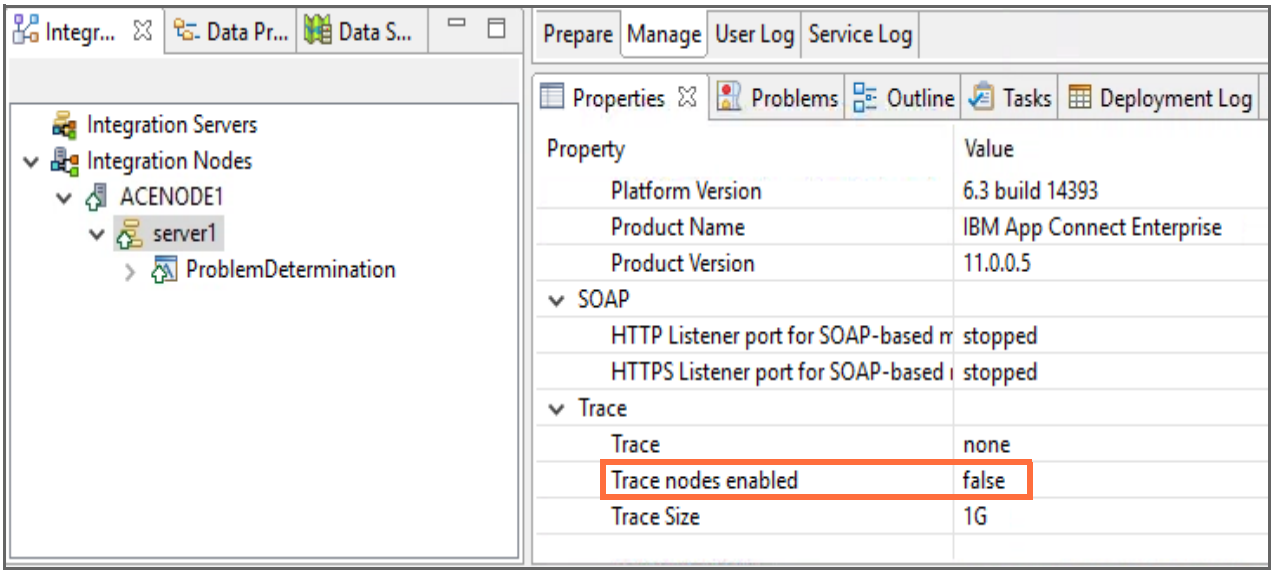
__ f. In the App Connect Enterprise Toolkit, right-click server1 and select **Stop**.



__ g. Right-click server1 and select **Start**.

__ h. Right-click server1 and select **Refresh**.

__ i. Verify that trace nodes are not enabled by viewing the properties of **server1**.



Part 3: Run the flow with User Trace

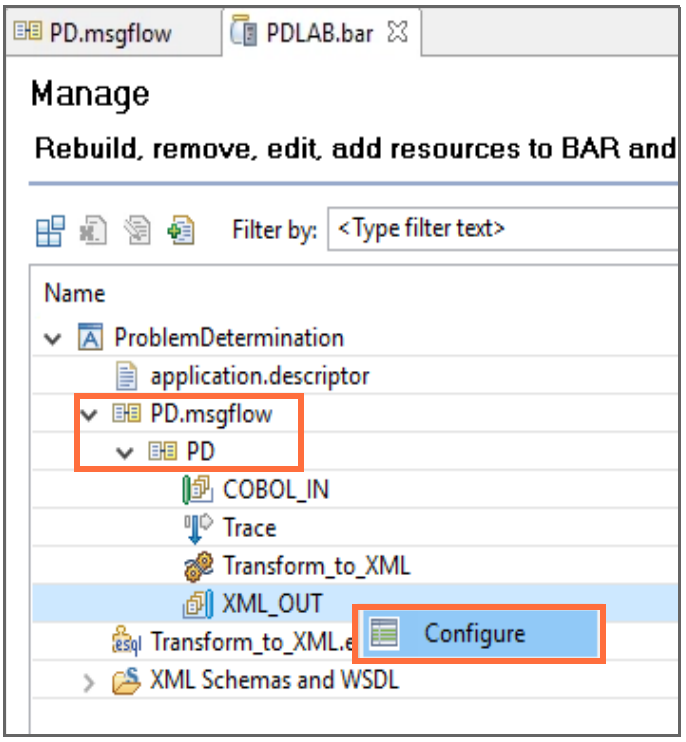
In this part of the exercise, you run the message flow with an invalid queue name for the MQOutput node that is named XML_OUT. This action causes a runtime error, which you diagnose by using a User Trace and the Event Viewer.

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

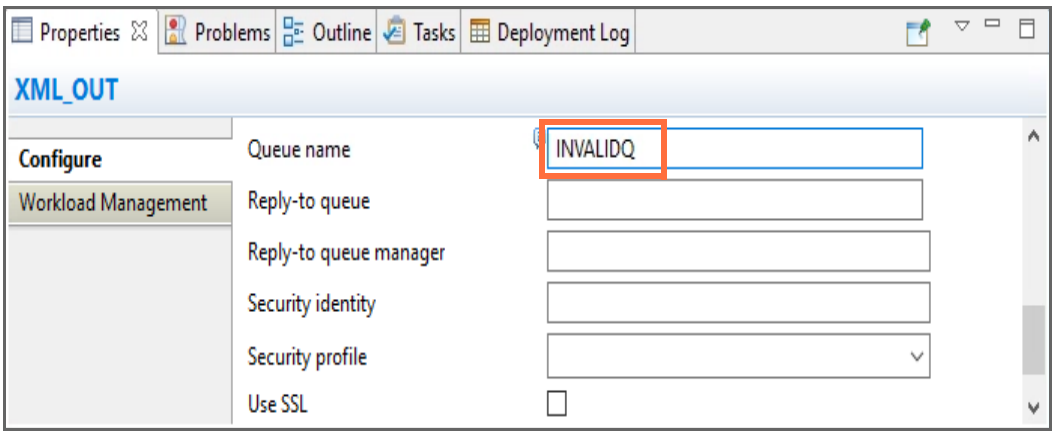
__ a. If the **PDLAB.bar** file is not already open, double-click the **PDLAB.bar** file in the **Application Development** view to open it in the BAR File editor.

__ 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.

__ f. Redeploy the modified BAR file to the **server1** integration server.

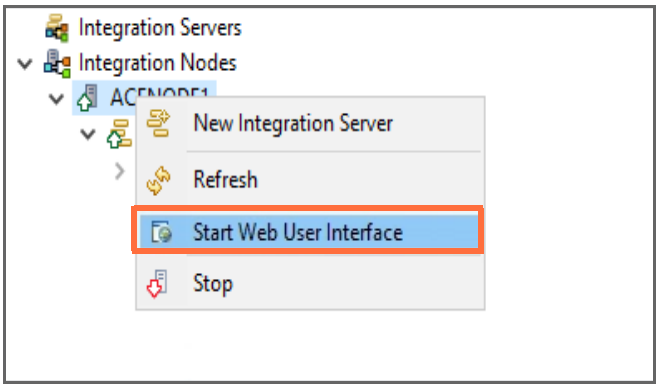
__ 2. Enable a user trace at the debug level on the **server1** integration server on ACENODE1.

__ a. In the IBM App Connect Enterprise Console window, type:

```
mqsichangetrace ACENODE1 -u -e server1 -l debug
```

You should see a message that the command was successful.

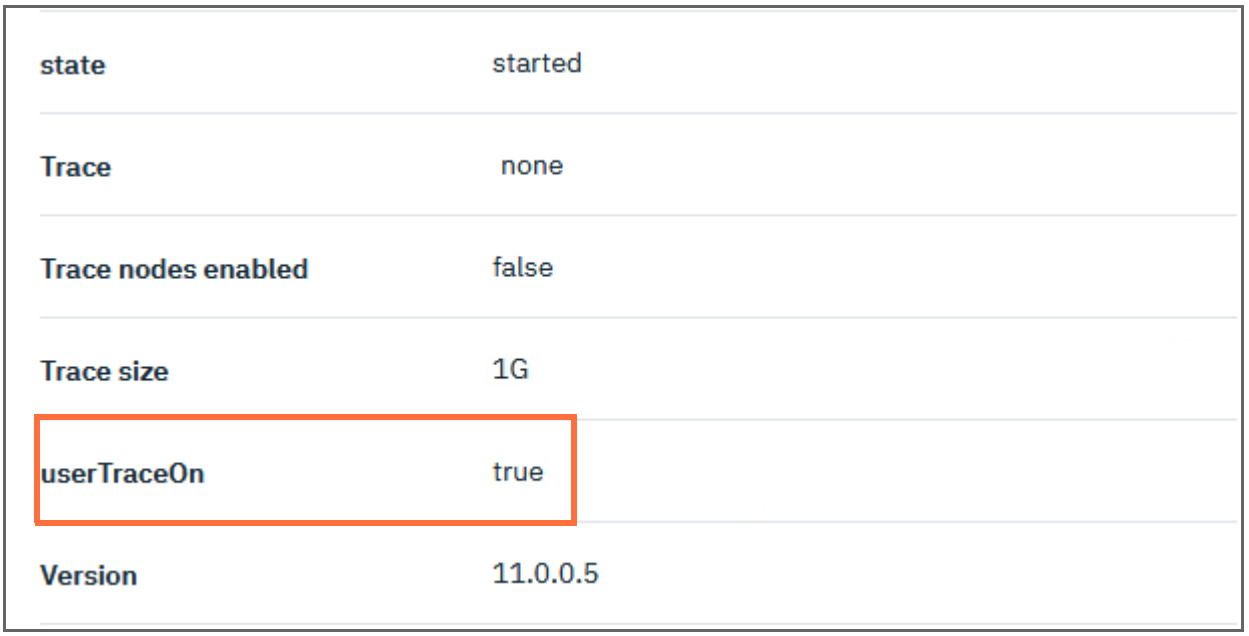
__ b. Verify that the user trace is active. In the App Connect Enterprise Toolkit, navigate to the Integration Explorer view, right-click **ACENODE1** and select **Start Web User Interface**.



__ c. Navigate to **ACENODE1 > server1**.

__ d. Open up **Properties** for server1.

__ e. Scroll down to the property **userTraceOn** and verify it is set to `true`.



__ 3. 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, App Connect Enterprise attempts to rollback the message for the following reasons:

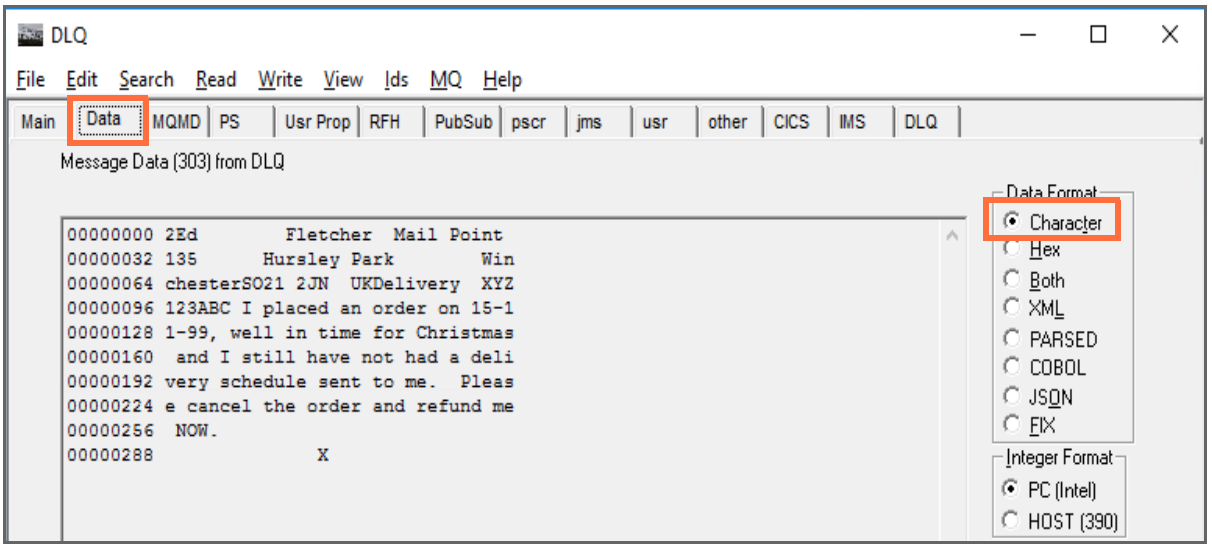
- The flow is transactional (default mode for the MQInput node).
- The **Catch** and **Failure** terminals are not wired on the COBOL_IN MQinput node.
- No back-out queue is configured for the queue COBOL_IN.

__ 4. 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.

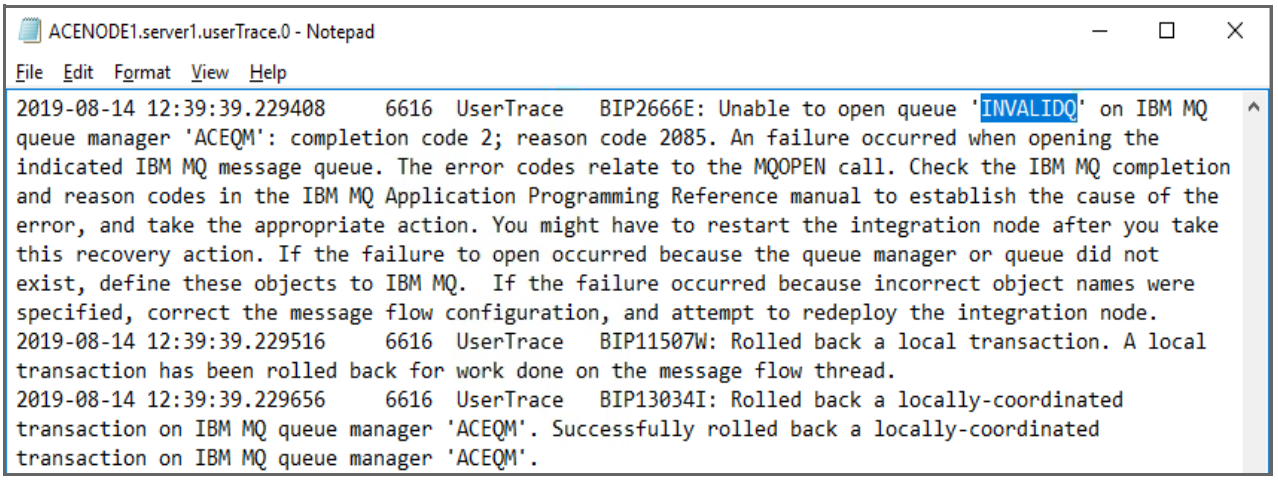
__ 5. Review the user trace.

__ a. In Windows Explorer, navigate to `C:\ProgramData\IBM\MQSI\common\log` and open the file: **ACENODE1.server1.userTrace.0** in Notepad.

Depending on the size, it can take a moment to load.

__ b. Scroll to the bottom.

__ c. Review the trace log and perform a search for `INVALIDQ`



__ d. When you are done reviewing the messages, close Notepad.

__ 6. To find out what that IBM MQ reason code means, type the following command in the IBM App Connect Enterprise Console window:

```
mqrc 2085
```

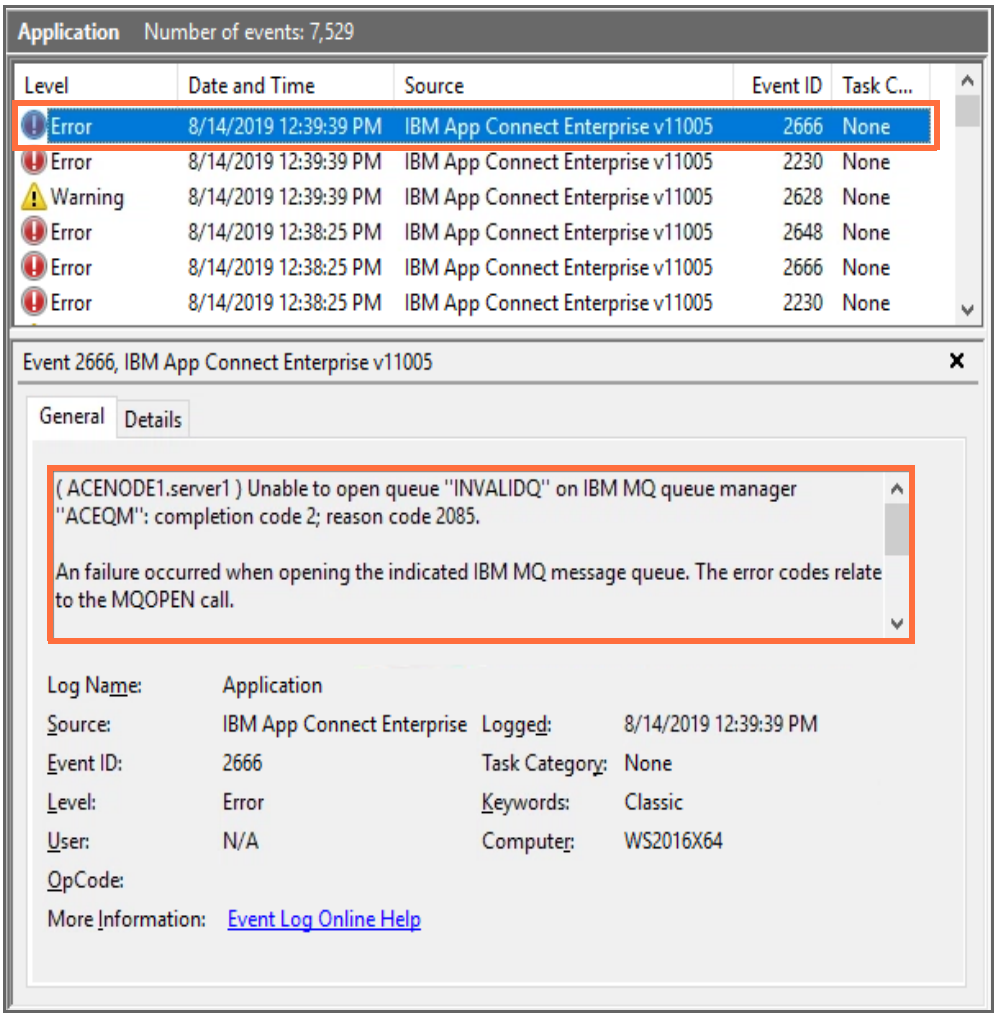
The reason code `MQRC_UNKNOWN_OBJECT_NAME` indicates that a non-existent output queue name (INVALIDQ) was specified in the message flow.

__ 7. 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 **Action > Refresh** to update the event log.

__ b. On the **Windows Logs > Application Logs** view, you should see some entries for IBM App Connect Enterprise that are marked with **Error**.



The errors that you see might not be in the same order.

__ c. View one of the errors listed.

__ d. Review the description to determine the cause of the error.

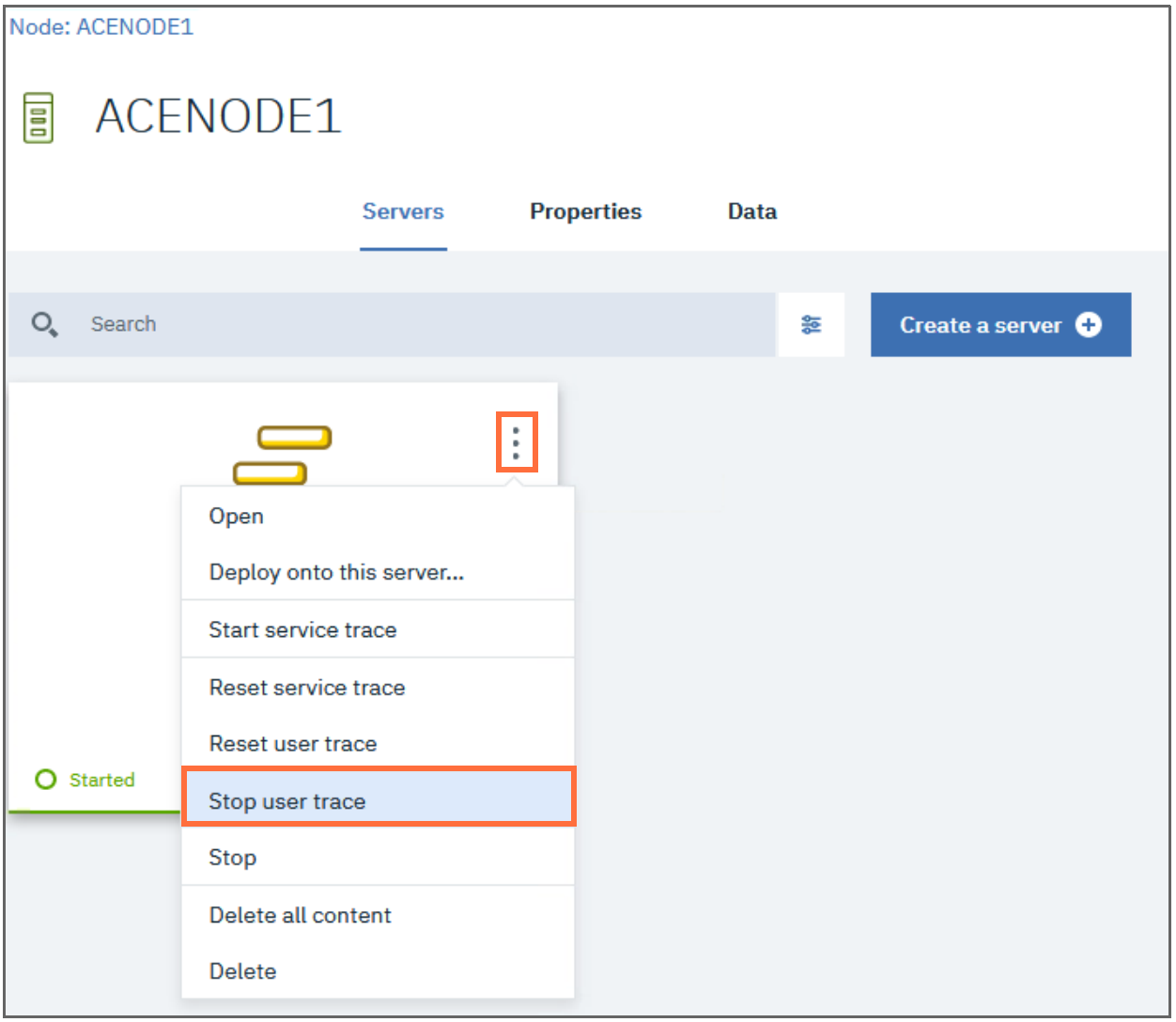
__ e. Subsequent error messages often provide more information about the error, or about processing that took place in response to the error.

Review the other messages that are marked as **Error**.

__ f. Close the Event Viewer.

__ 8. Turn off user trace.

__ a. In the IBM App Connect Enterprise web user interface, navigate to **server1**. Click the three dots in the upper right corner of the server and select **Stop user trace**.



__ b. Click **server1** and select **Properties**.

__ c. Scroll to the bottom

__ d. Verify **userTraceOn** is set to `false`.

Part 4: Use the Message Flow Debugger

In this part of the exercise, you configure the Message Flow Debugger in the IBM App Connect Enterprise Toolkit and use it for problem determination.

__ 1. Attach the debugger to the integration server by setting the Java debug port in the `server.conf.yaml` configuration file.

__ a. Using Windows Explorer, navigate to the following directory: `C:\ProgramData\IBM\MQSI\components\ACENODE1\servers\server1`

__ b. Open the `server.conf.yaml` file by using **Notepad**.

__ c. Search for `jvmDebugPort`.

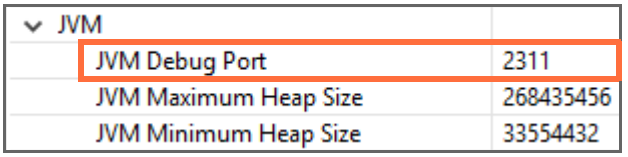
__ d. Update the setting by deleting the # character and changing the port number from `0` to 2311.

__ e. Save the file.

__ f. In the App Connect Enterprise Toolkit, stop the server, then start it.

__ g. Right-click server1 and select **Refresh**.

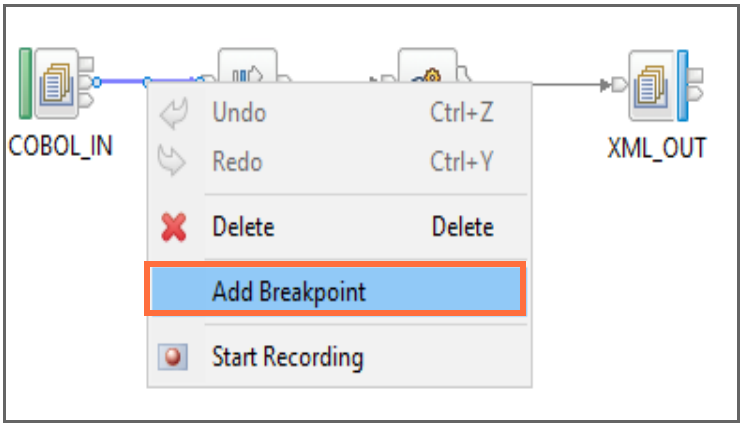
__ h. Verify the JVM Debug Port under the properties for **server1** is set to 2311.



__ 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**.

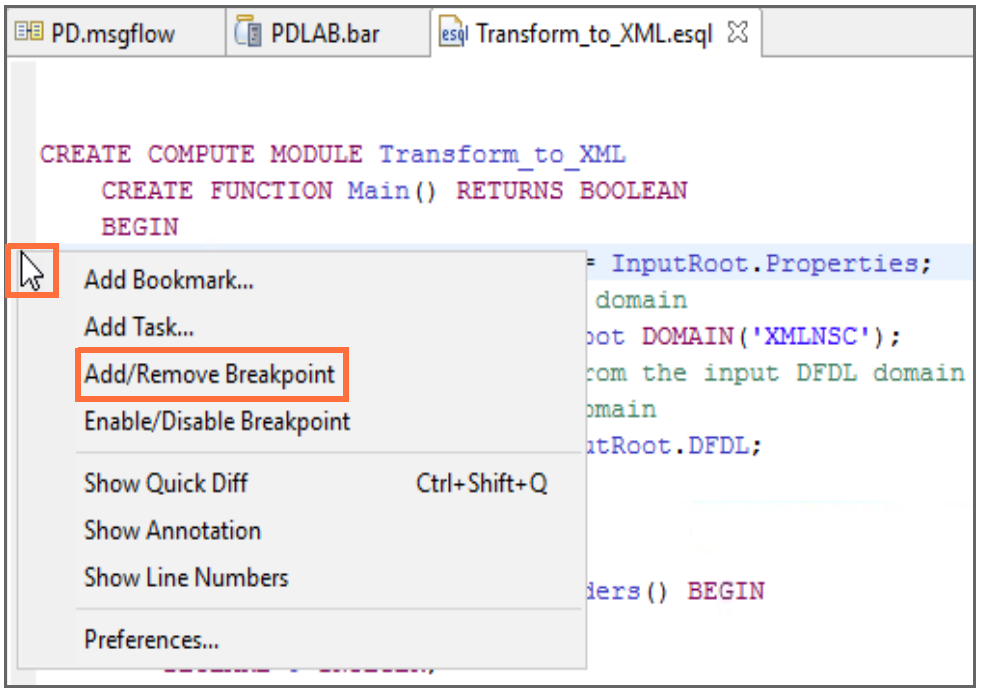


A blue circle is displayed on the wire, indicating that a breakpoint is set.

__ c. Double-click the Transform_to_XML Compute node to open the ESQL editor.

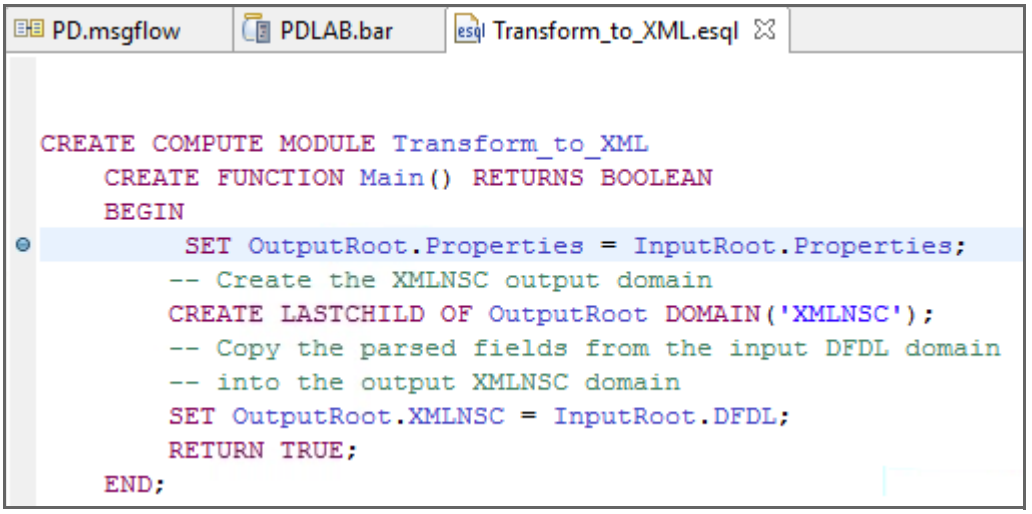
__ d. Click the line `SET OutputRoot.Properties = InputRoot.Properties;` to highlight it.

__ e. Right-click in the gray margin to the left of the line of code, and then click **Add Breakpoint**.



__ f. Click **OK** on the Add Bookmark message.

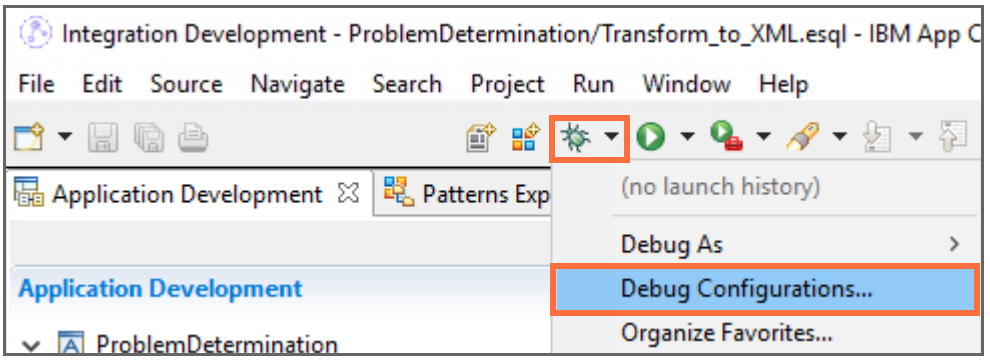
A blue marker is displayed in the margin, indicating that a breakpoint is set.



__ g. Save your work.

__ 3. Configure the debugger.

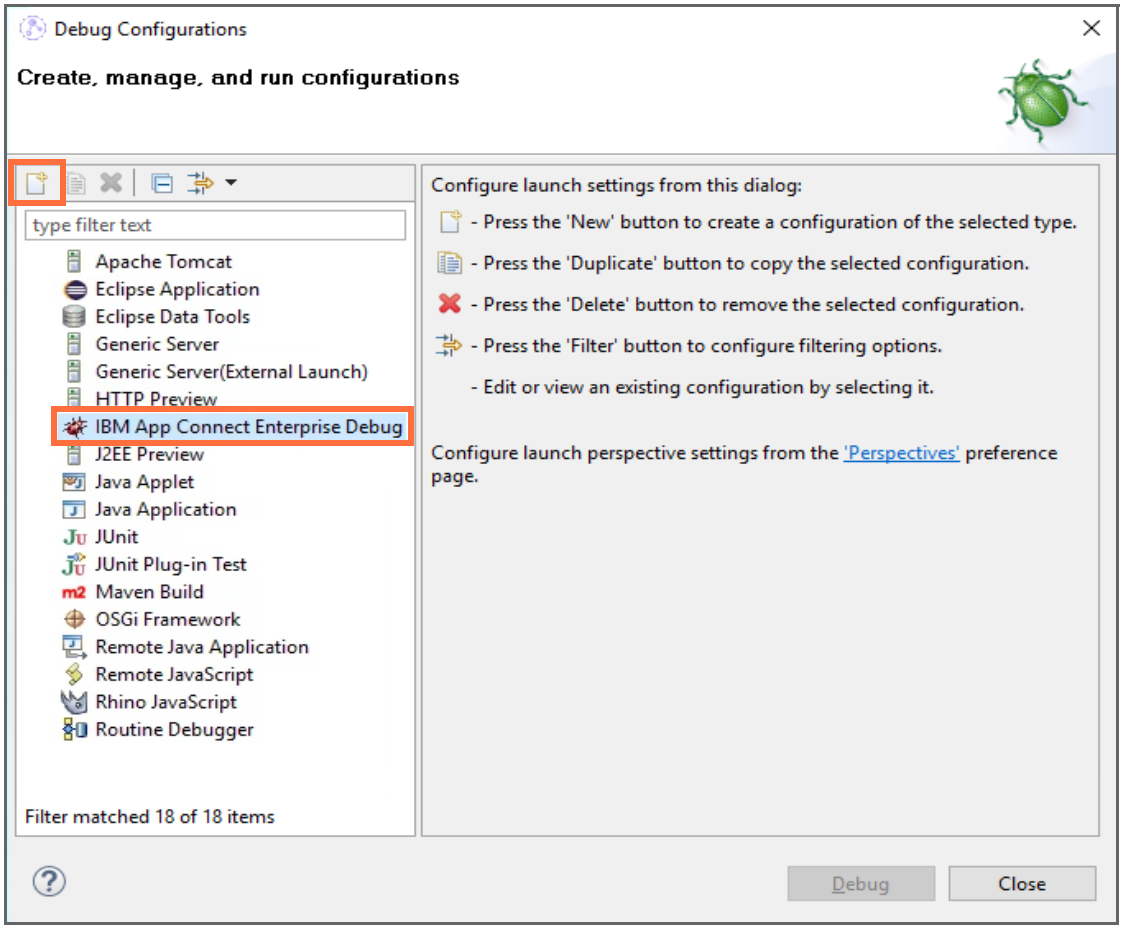
__ a. In the IBM App Connect Enterprise Toolkit, click the down-arrow to the right of the Debug button in the action bar, and then click the **Debug configurations** menu item.



Because you are using the Debugger for the first time, you must create an IBM App Connect Enterprise debug configuration.

__ b. Select **IBM App Connect Enterprise Debug**.

__ c. Click **New Launch Configuration**..

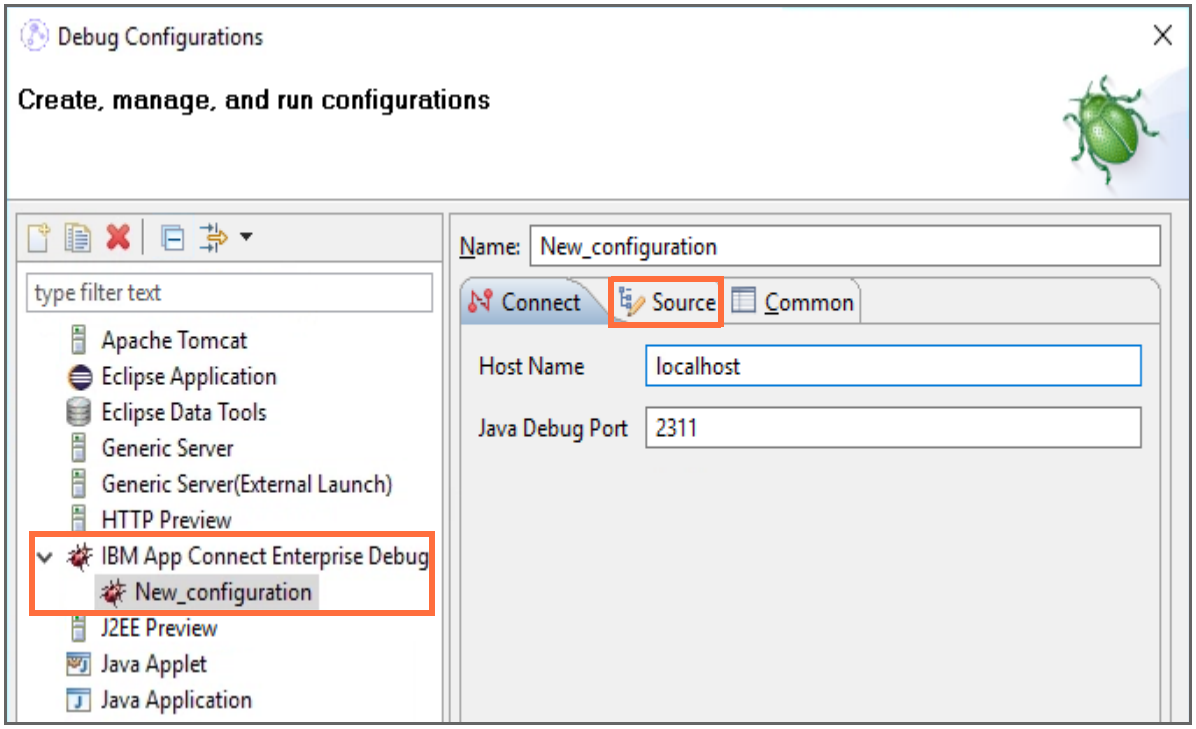


The **Debug Configurations** window is displayed.

__ 4. To display the source code during debugging, you must identify the location of the source code to the debugger.

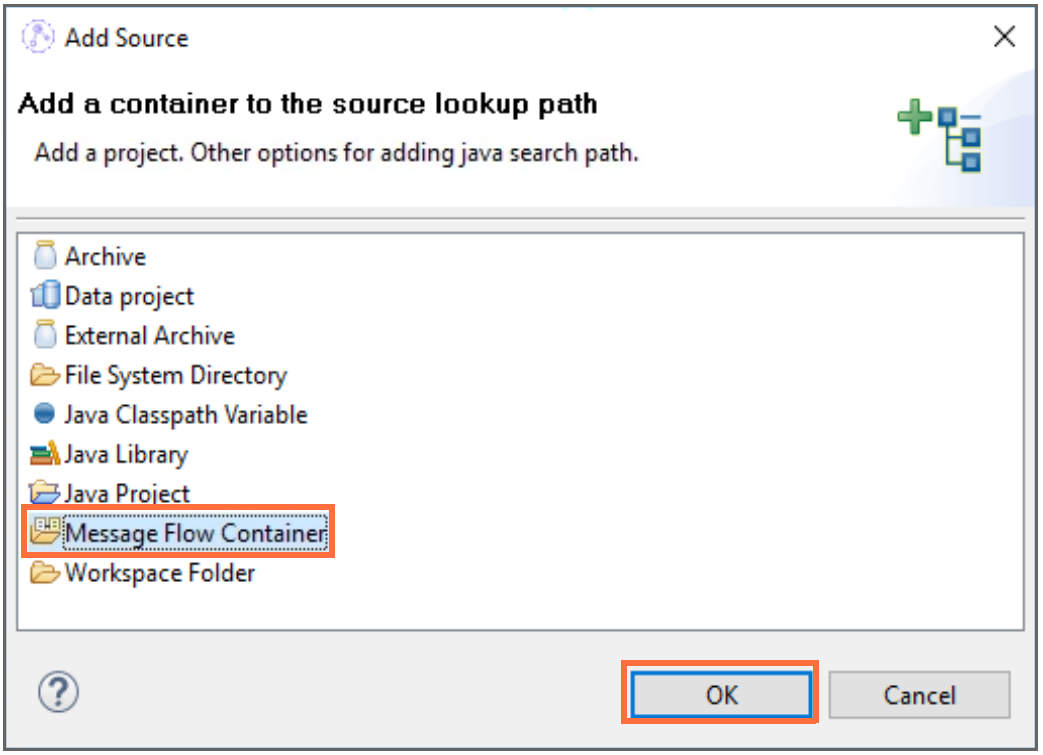
__ a. Change the Java Debug Port to 2311

__ b. Click the **Source** tab.

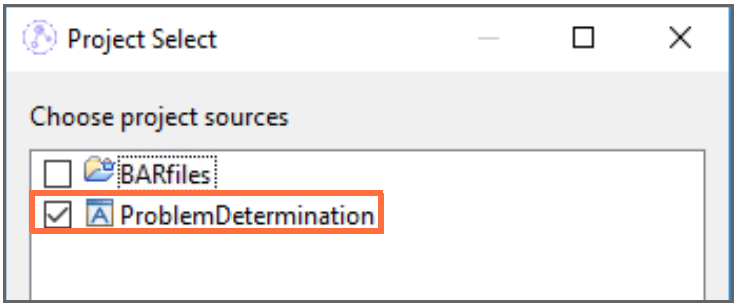


__ c. Click **Add**.

__ d. On the **Add Source** window, click **Message Flow Container**, and then click **OK**.



__ e. On the **Project Select** window, select **ProblemDetermination**, and then click **OK**.



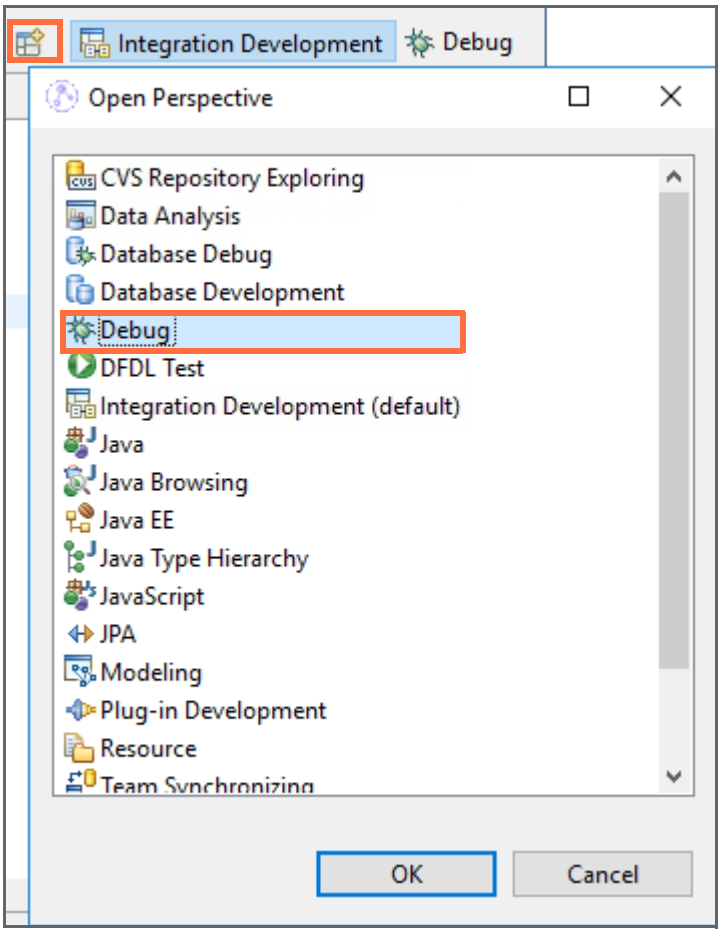
__ f. Click **Apply**.

__ 5. Start the Debug session.

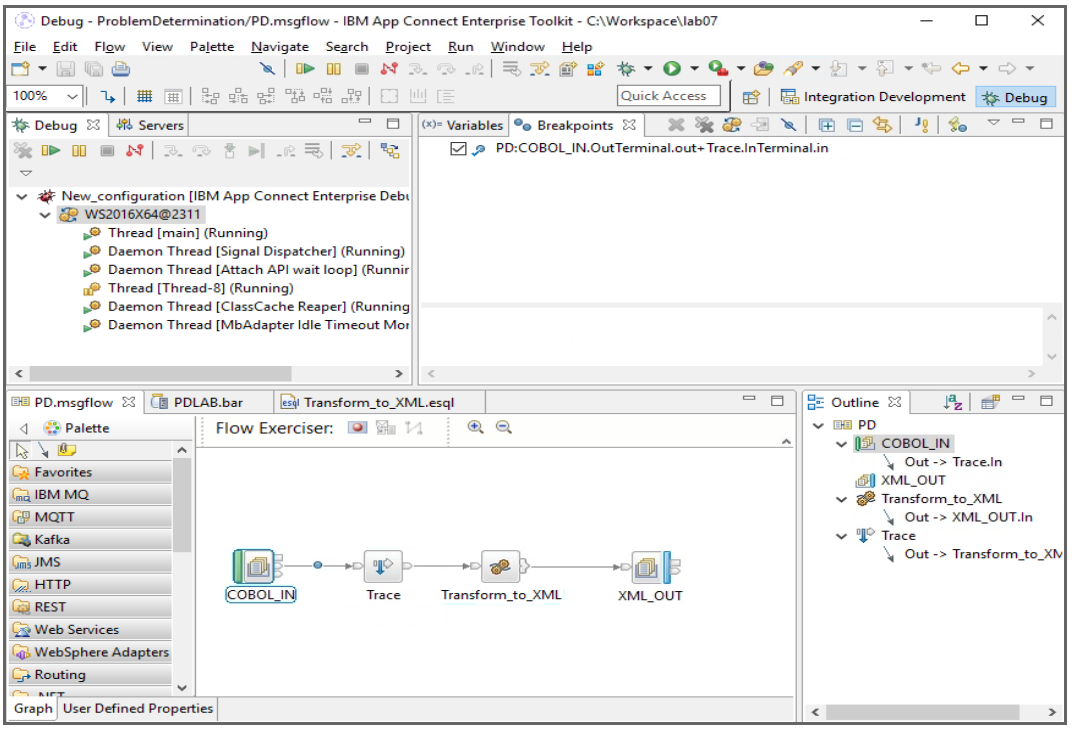
__ a. Click **Debug**.

__ b. A Progress tab is opened and the debugger is started.

__ c. Change to the Debug perspective.



__ d. The Debug tab is opened. The name of the debug session is shown in the **Debug** view with the hostname (WS2016X64) and debug port (2311).



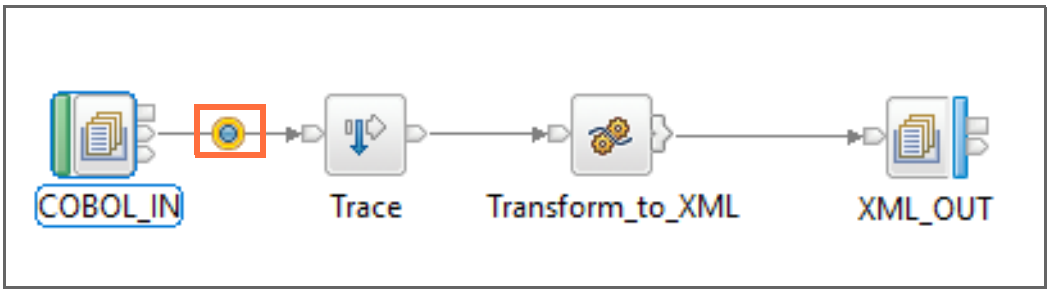
Your screen might not exactly match the screen capture that is shown in this exercise.

__ 6. Using the Debug session, send a test message.

__ a. Using the RfhUtil session that is already running for queue COBOL_IN, send a test message by clicking **Write Q**.

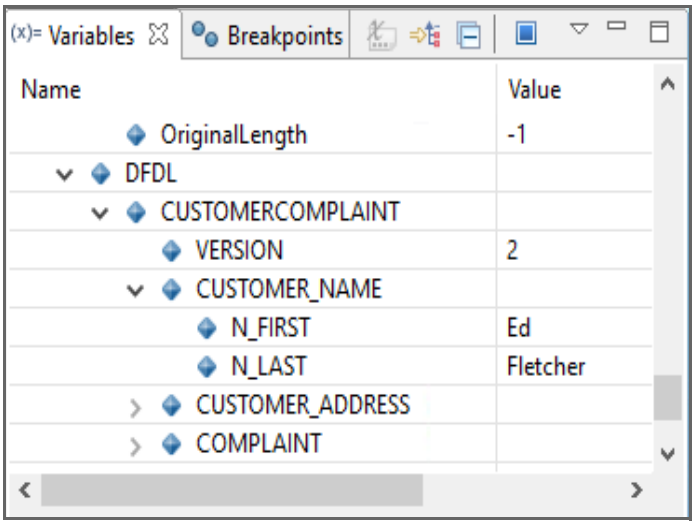
__ b. In the IBM App Connect Enterprise Toolkit, the **Debug** view shows that the message flow is suspended.

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 the flow is halted.



The **Variables** pane now shows the elements of the message assembly: Message, Local Environment, Environment, and the Exception List.

__ c. In the Variables pane, expand **Message > DFDL** and a few of the subordinate components such as CUSTOMER_NAME and CUSTOMER_ADDRESS.



__ 7. 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.

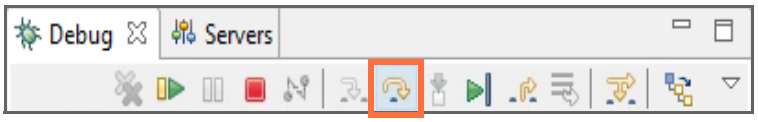
__ b. Click the VERSION **Value** column and type over its current value (2) with another value.

Note

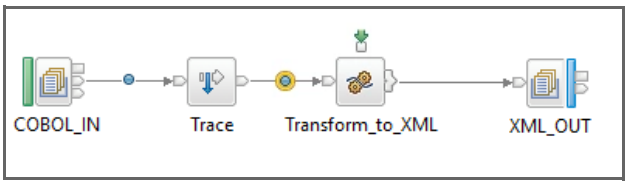
In this exercise, modifying this field does not change the runtime results. This step in the exercise is intended to show that you can change a value during debugging when necessary.

__ 8. 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.

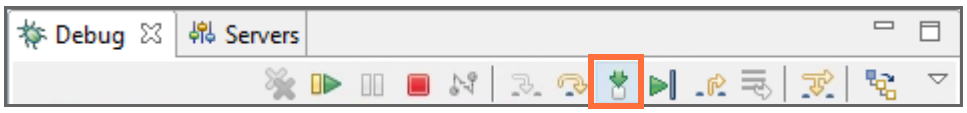


Step over pauses the debugger after the next node (the Trace node) runs. You see that the blue circle is highlighted on the wire between the Trace node and the Compute node, which acts as a temporary breakpoint.



You also see a **Step into source** option above the Transform_to_XML node.

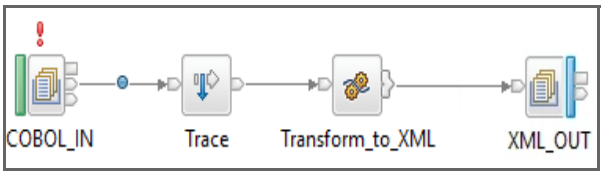
__ b. Click **Step into source** to view the next breakpoint in the code.



The next breakpoint is highlighted in the **Transform_to_XML.esql** tab where you set the breakpoint in the ESQL code.

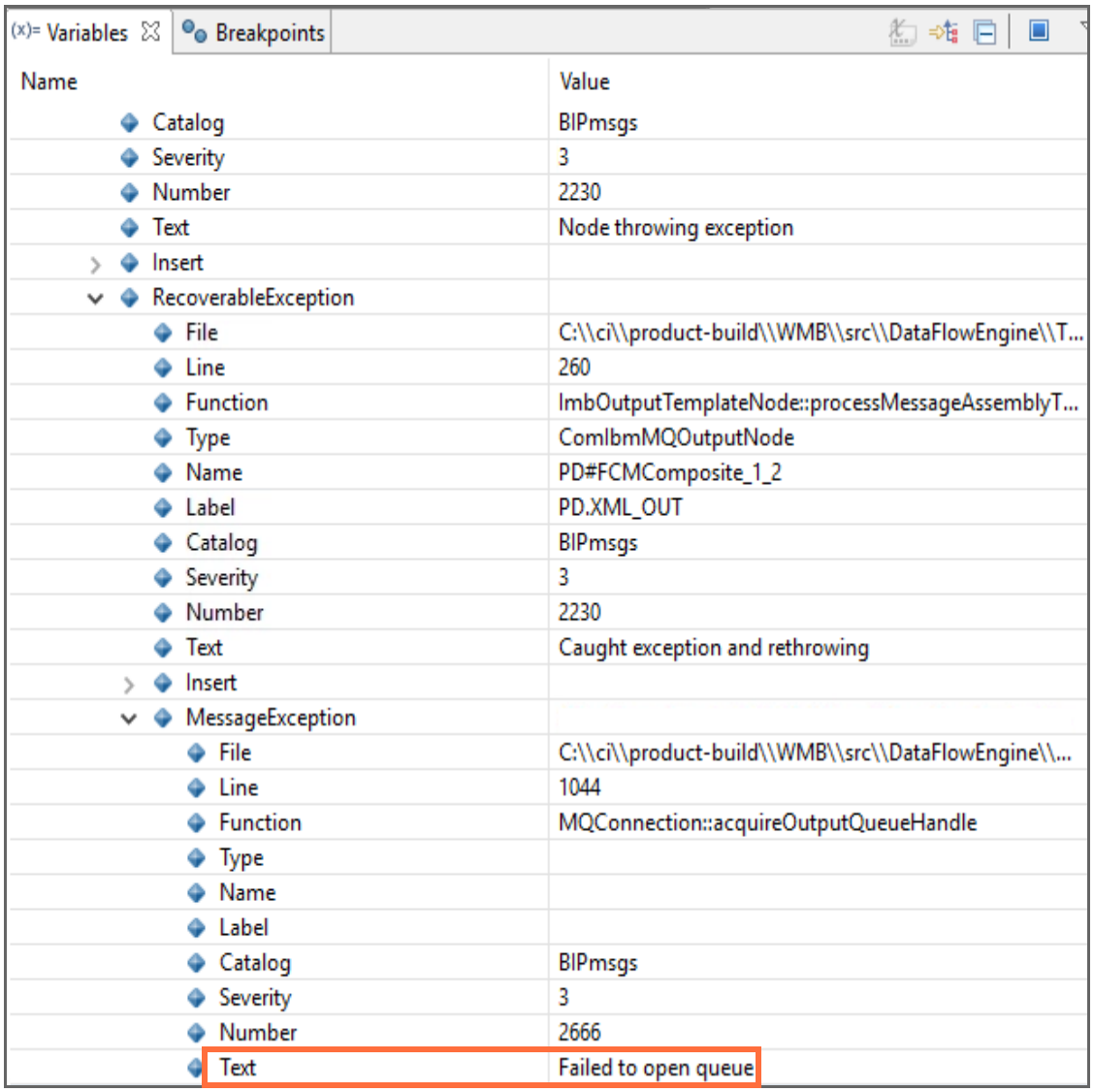
__ c. Click **Resume** to continue processing the message.

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.



__ d. 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.



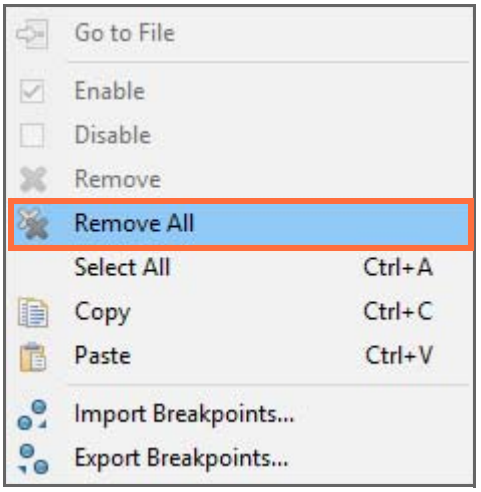
__ e. Disconnect from the debug session by highlighting the debug configuration and clicking Disconnect.

The Debug session terminates.

__ 9. Remove all breakpoints from the message flow.

__ 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.

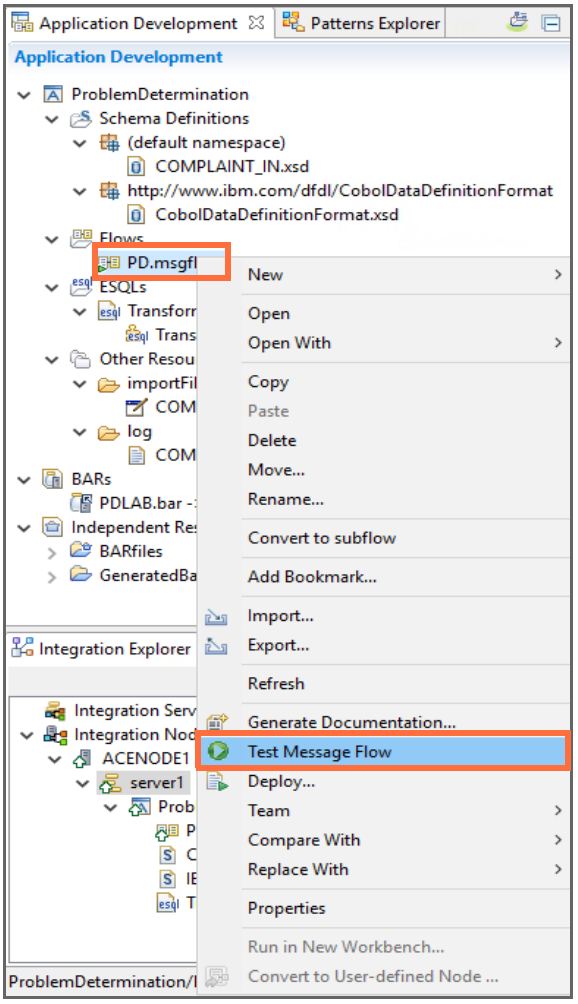
__ d. Switch back to the Integration Development perspective.

Part 5: Use Component Trace in the Unit Test Client

In this part of the exercise, you enable the Unit Test Client in the IBM App Connect Enterprise Toolkit and then use the Test Client to generate a Component Trace.

__ 1. Start the Test Client for the message flow.

__ a. Right-click the **PD.msgflow** message flow in the Application Development pane and select **Test Message Flow**

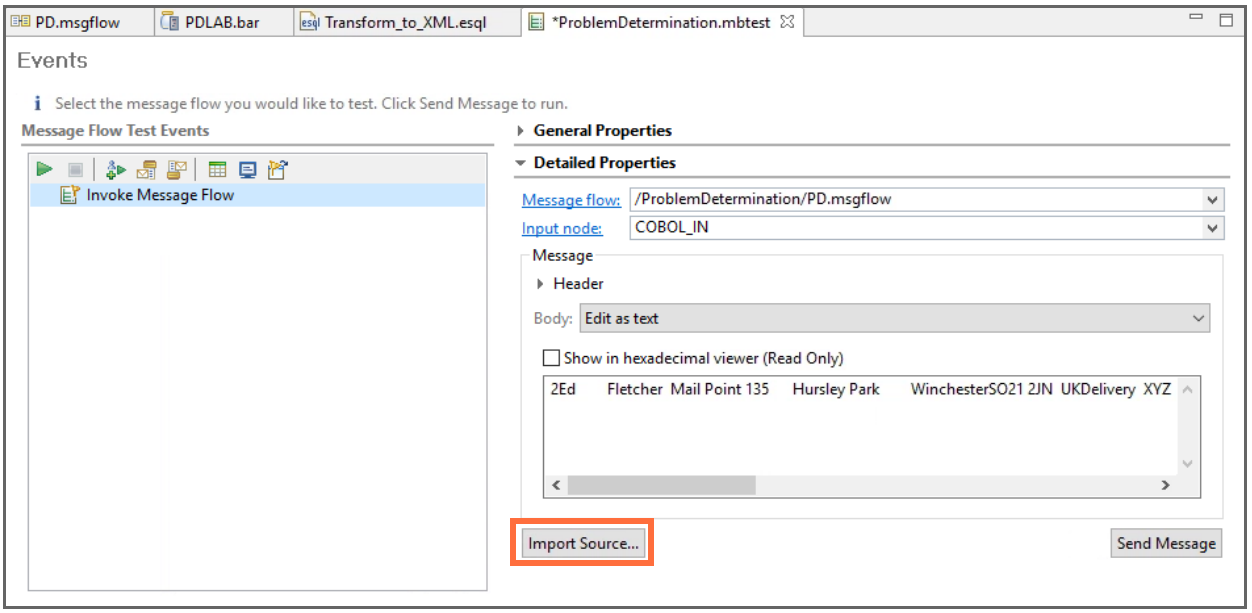


__ b. Click OK in the Confirmation window.

The message flow Test Client file is created and the Test Client pane opens. The Test Client file is named `ProblemDetermination.mbtest` and is saved in the **GeneratedBarFiles** folder under the **Independent Resources** project.

__ 2. Import the message into the Test Client.

__ a. Click **Import Source**.



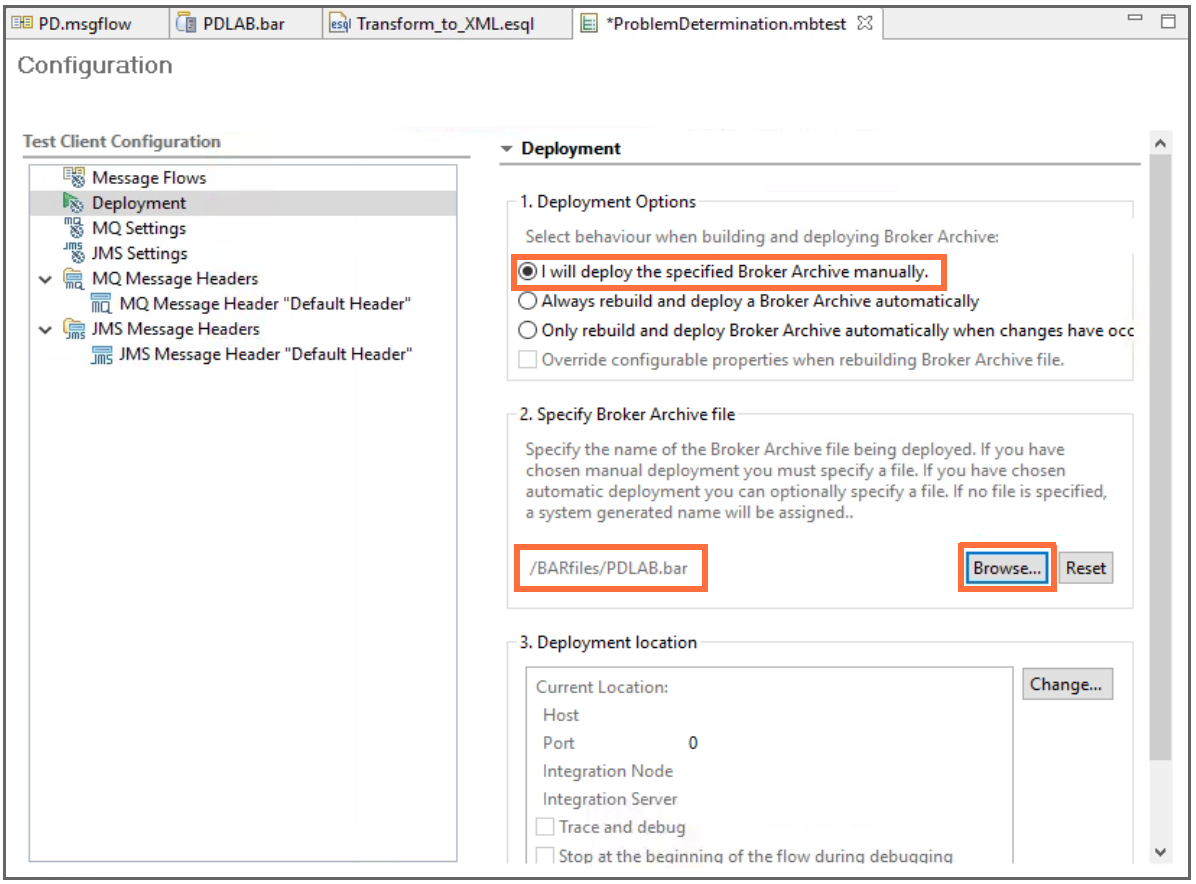
__ b. Browse to `C:\labfiles\Lab07-PDTools\data` directory, click `Complaint_cwf.txt`, and then click **Open**.

__ 3. Configure the Test Client for Component Trace to use the existing BAR file.

__ a. Click the **Configuration** tab (at the bottom of the ProblemDetermination.mbtest view).

__ b. In the **Deployment Options** section, click **I will deploy the specified Broker Archive manually**.

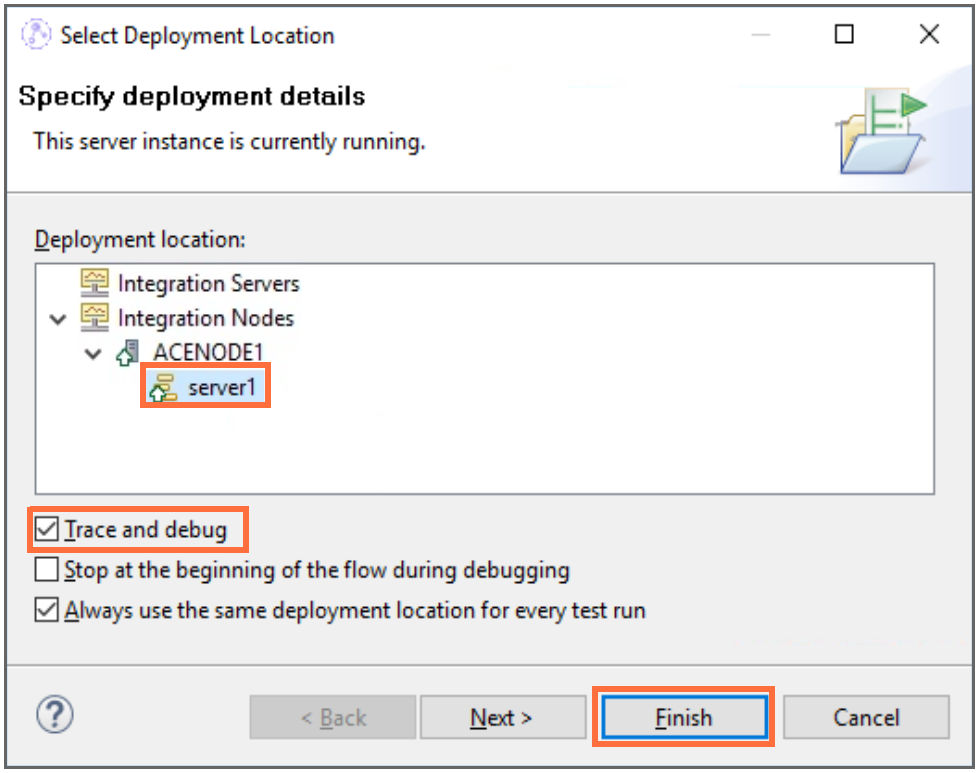
__ c. In the **Specify Broker Archive file** section, click **Browse**, click the **PDLAB.bar** file, and then click **OK**.



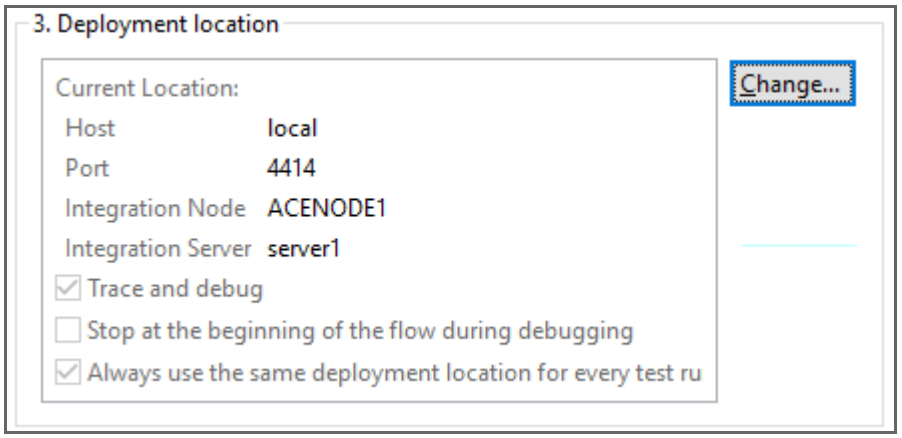
__ 4. Enable the component trace.

__ a. In the **Deployment Location** section on the **Configuration** tab, click **Change**.

__ b. Select **Trace and debug**, select **server1**, and then click **Finish**.

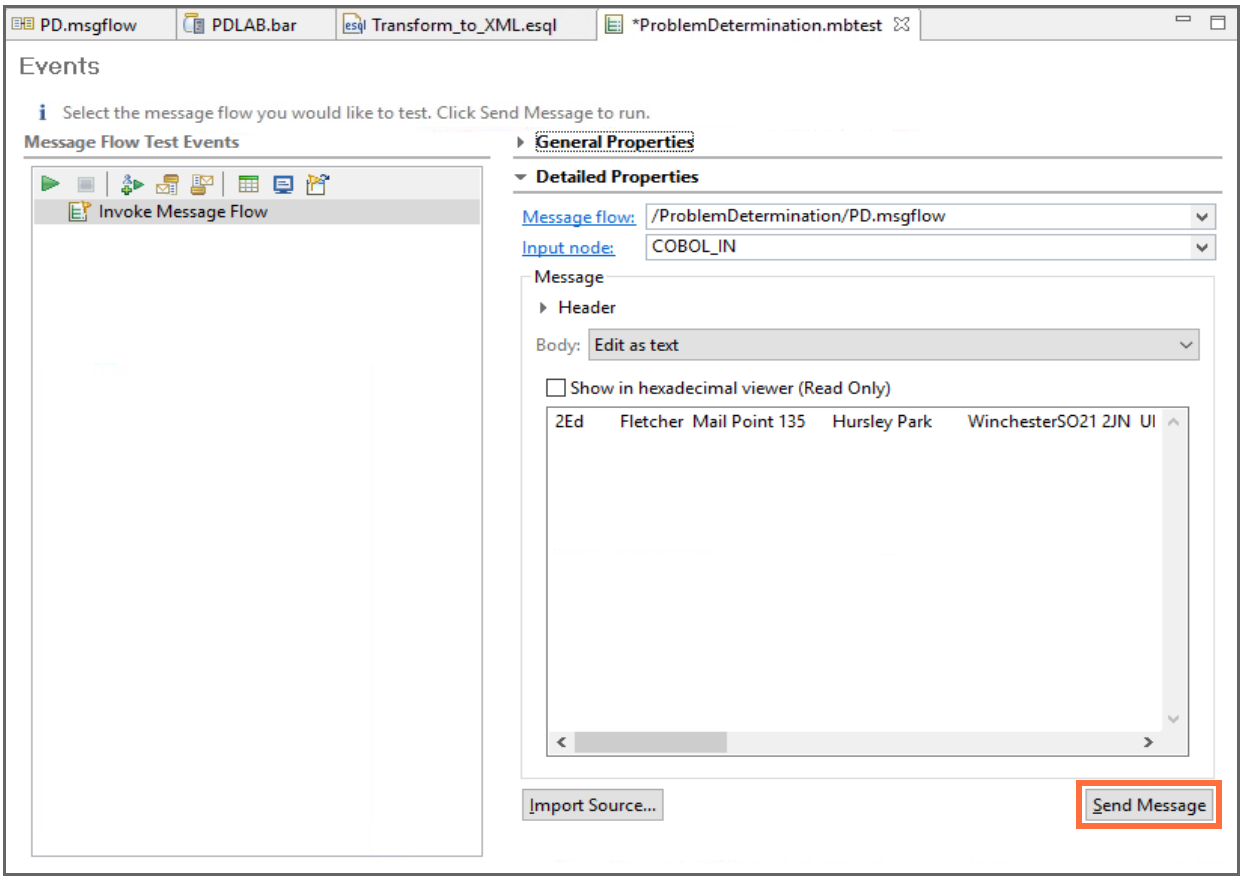


The Current Location is updated with the Host, Port, Integration Node, and Integration Server.



__ 5. 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.

__ 6. Review the Component Trace output.

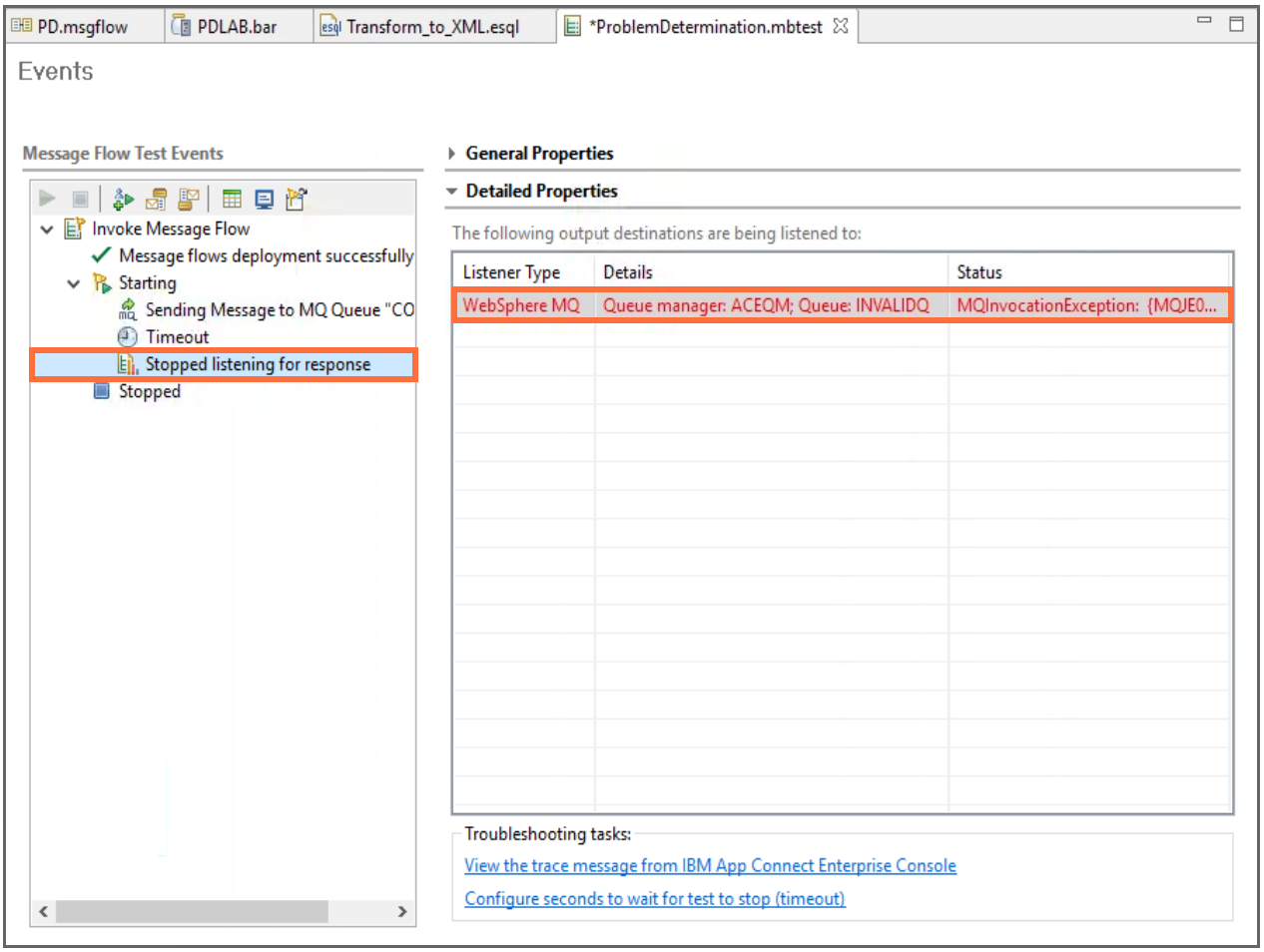
__ a. Wait for the Timeout event (two minutes) and the notification that the Flow exerciser stopped listening for a response.

__ b. Review the Message Flow Test Events in the Test Client **Events** tab.

__ c. Click **Timeout**. You see the message: **The test timed out after 120 seconds**.

__ d. Click **Stopped listening for response**.

__ e. Click the exception, and review the **Detailed Properties** section.

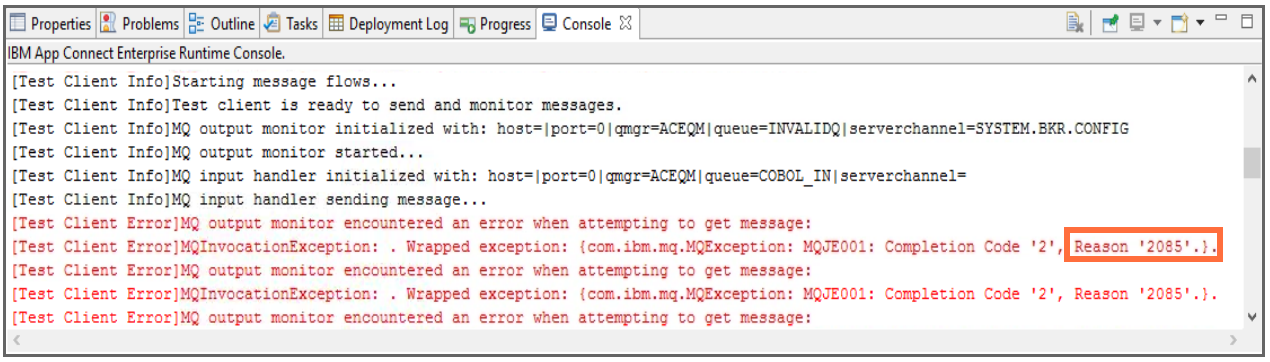


__ f. You see in the details the reference to INVALIDQ.

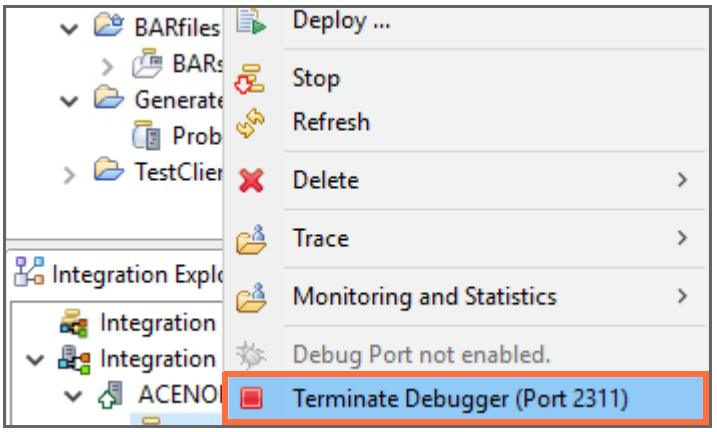
__ g. Click the **View the trace message from IBM App Connect Enterprise Console** link.

The Console is populated below.

__ h. Review the test client errors until you see the **2085** status code.



__ i. Stop the Message Flow debugger. In the **Integration Explorer view**, right-click the **server1** integration server and click **Terminate Debugger**.



Part 6: Exercise clean-up

__ 1. Close the two RfhUtil windows.

__ 2. Close the Bar file editor pane.

__ 3. Delete all resources from **server1** integration server on **ACENODE1**.

__ 4. In IBM MQ Explorer, delete the queues

__ a. Select all queues.

__ b. Right-click and select **Delete...**

__ c. When asked if you are sure that you want to delete the 3 selected objects, click **Delete**.

__ d. When asked to clear messages, click the checkbox to **Clear all messages from the queue**. Click **Delete**.

All queues are deleted.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you extended a 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 App Connect Enterprise 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 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 App Connect Enterprise 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 App Connect Enterprise 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 App Connect Enterprise Toolkit Test Client and message flow debugger view to step through a message flow application
- Examine the IBM App Connect Enterprise logs and system logs to diagnose problems

Exercise 8. Implementing explicit error handling

Estimated time

01:30

Overview

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.

Objectives

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

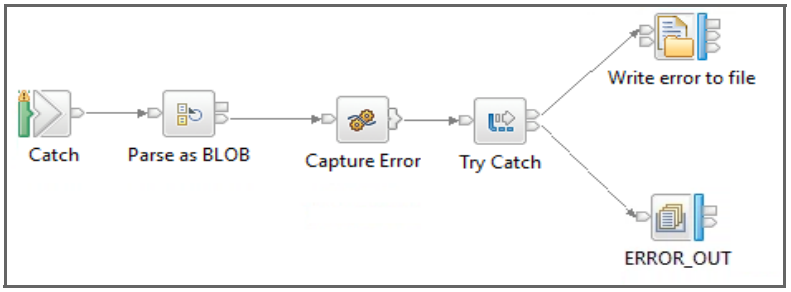
Introduction

In a typical IBM App Connect Enterprise 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.

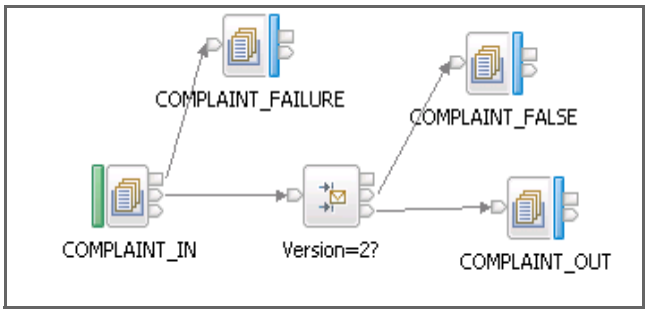
In this exercise, you add a general-purpose error handler to a message flow application. The error handler is written in the form of a subflow.

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 App Connect Enterprise Toolkit Flow exerciser to test the message flow and the error handler subflow.

Requirements

- A lab environment with IBM App Connect Enterprise V11 and IBM MQ V9
- Lab files in the `C:\labfiles\Lab08-ErrorHandler`
- The following local queues are on the ACEQM queue manager: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_FALSE, COMPLAINT_OUT, and ERROR_OUT
- The user “aceadmin” is a member of the “mqm” group

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab08`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, enter `C:\Workspace\Lab08`

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

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

__ 2. Verify that the node and server are running.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 3. This exercise requires an IBM MQ queue manager.

If you completed Exercise 3, you created a queue manager that is named ACEQM. You can use that queue manager in this exercise. Proceed to the next step.

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

__ a. Start IBM MQ Explorer.

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

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

__ 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.

__ 4. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

__ a. In the command window, type:

```
runmqsc ACEQM < C:\labfiles\Lab08-ErrorHandler\CreateQueues.mqsc
```

__ 5. Use IBM MQ Explorer to verify that the queues were created.

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

__ 6. 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 App Connect Enterprise 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.

__ f. 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.

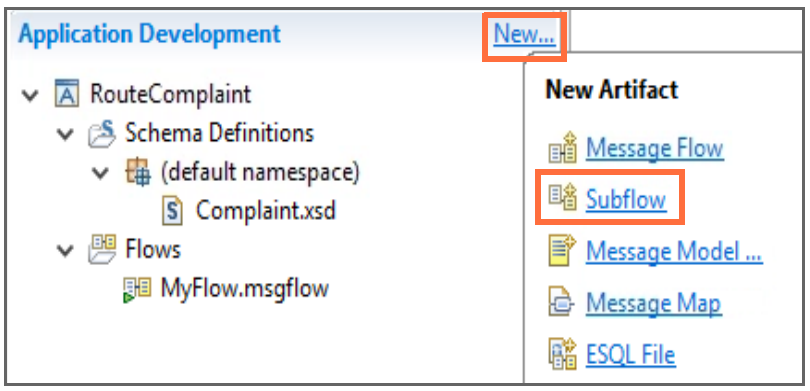
Part 2: Create the ErrorHandler subflow

In this part of the exercise, you use the IBM App Connect Enterprise Toolkit to construct an error handler subflow.

__ 1. Create the message flow container for the subflow.

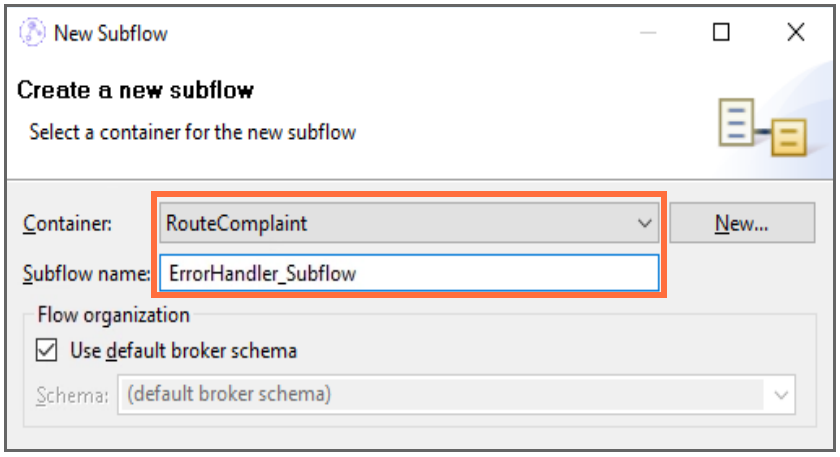
__ a. In the **Application Development** view, click **New**.

__ b. Click **Subflow** in the **New Artifact** menu.



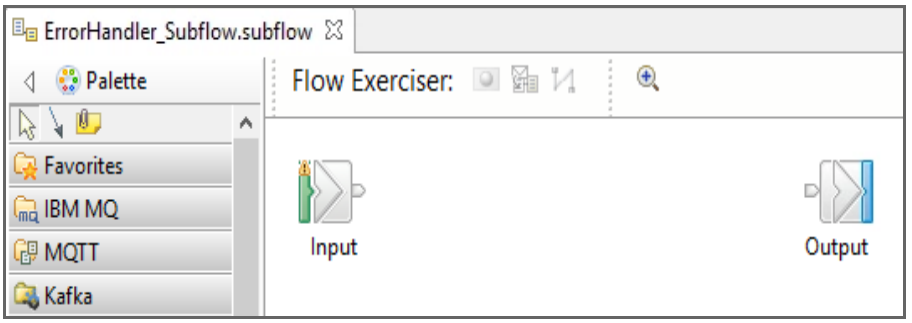
__ c. For **Container**, select **RouteComplaint**.

__ d. For **Subflow name**, type: **`ErrorHandler_Subflow`**



__ e. 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.

__ 2. Update the Input and Output nodes.

__ a. Rename the Input node to **Catch**. On the **Description** tab of the node **Properties**, change the **Node name** property to **`Catch`**.

__ b. Delete the **Output** node from the subflow Message Flow editor canvas.

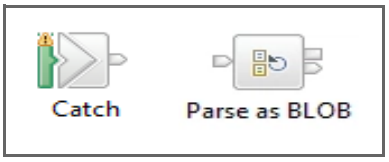
This subflow is not intended to send data back to the main flow.

__ 3. Add a **ResetContentDescriptor** node to the subflow after the subflow Input node.

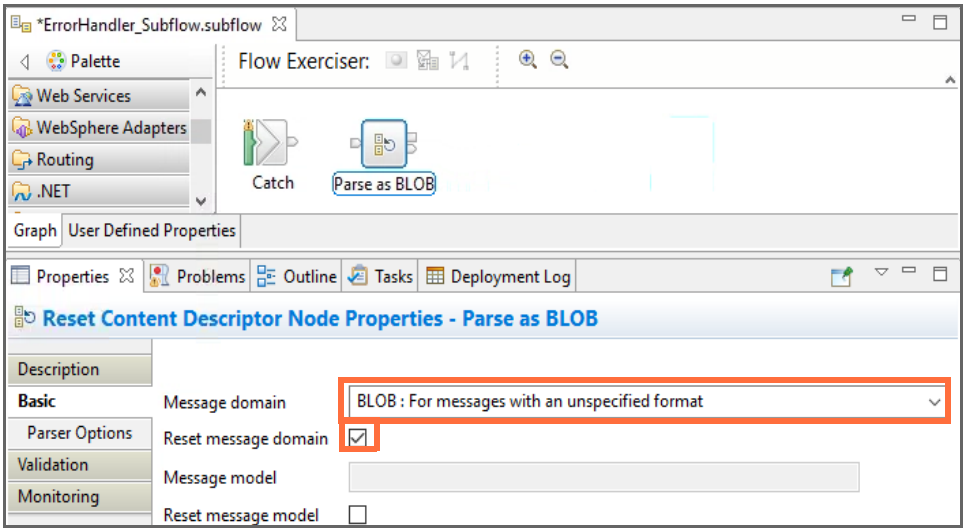
This node forces the message to be reparsed according to the parser domain you specify in the node properties. Regardless of the domain of the incoming message, it can be converted to a BLOB by using this technique.

__ 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** checkbox.



__ d. Save your work.

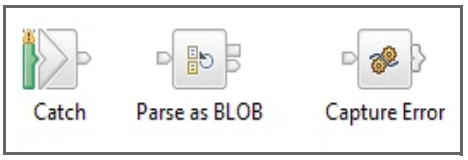
__ 4. Add a Compute node to the subflow.

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 App Connect Enterprise.

The ESQL code in the Compute node is only called if a runtime error occurs. The code retrieves the information about the error from the ExceptionList, formats it, and places it in the `OutputRoot.MQRFH2.usr` folder.

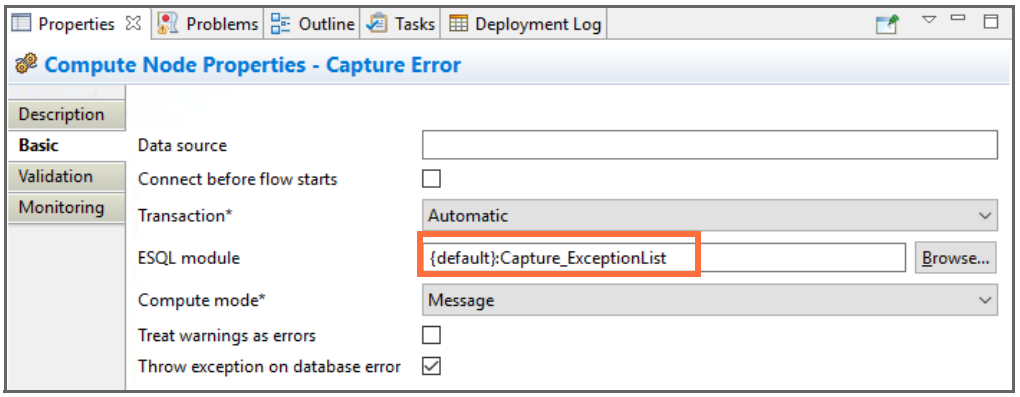
__ a. From the **Transformation** drawer, add a Compute node to the canvas after the ResetContentDescriptor node (**Parse as Blob)**.

__ 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

__ 5. Add a TryCatch node.

__ a. 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.

__ 6. Add an MQOutput node for the “Try” destination for the original message and the error information.

__ a. From the **IBM MQ** drawer, add an MQOutput node after the Try Catch node.

__ b. Rename the MQOutput node to `ERROR_OUT`.

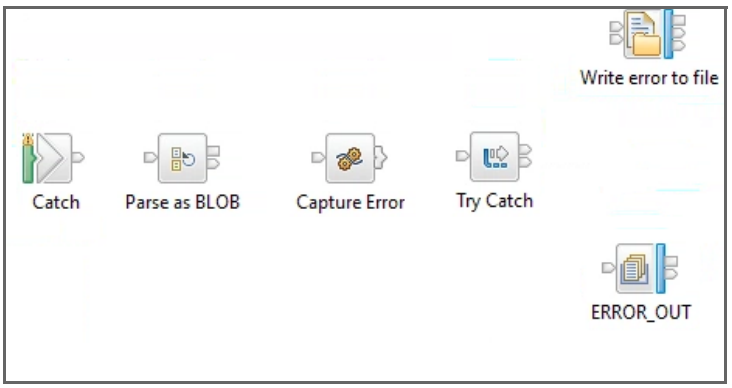
__ c. For the **Queue name** property on the **Basic** tab, type: `ERROR_OUT`

__ d. For the **Destination queue manager name** property on the **MQ Connections** tab, type: `ACEQM`

__ 7. Add a FileOutput node for 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.

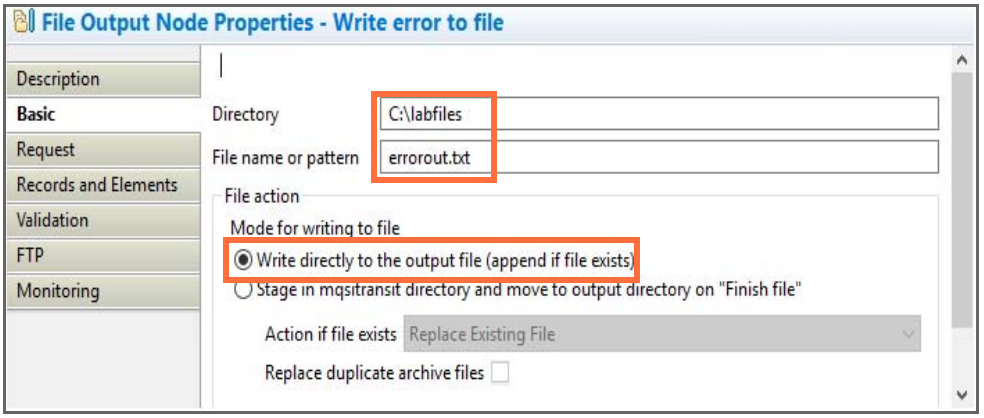
__ b. Rename the FileOutput node to `Write error to file`.



__ 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** tab, enter: `$Root`

__ 8. Connect the nodes.

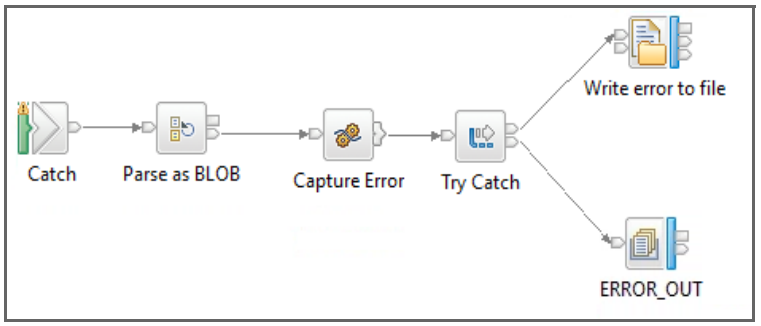
__ a. Wire the Input node (**Catch**) **Out** terminal to the ResetContentDescriptor node (**Parse as Blob**) **In** terminal.

__ 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.



__ f. Save the subflow

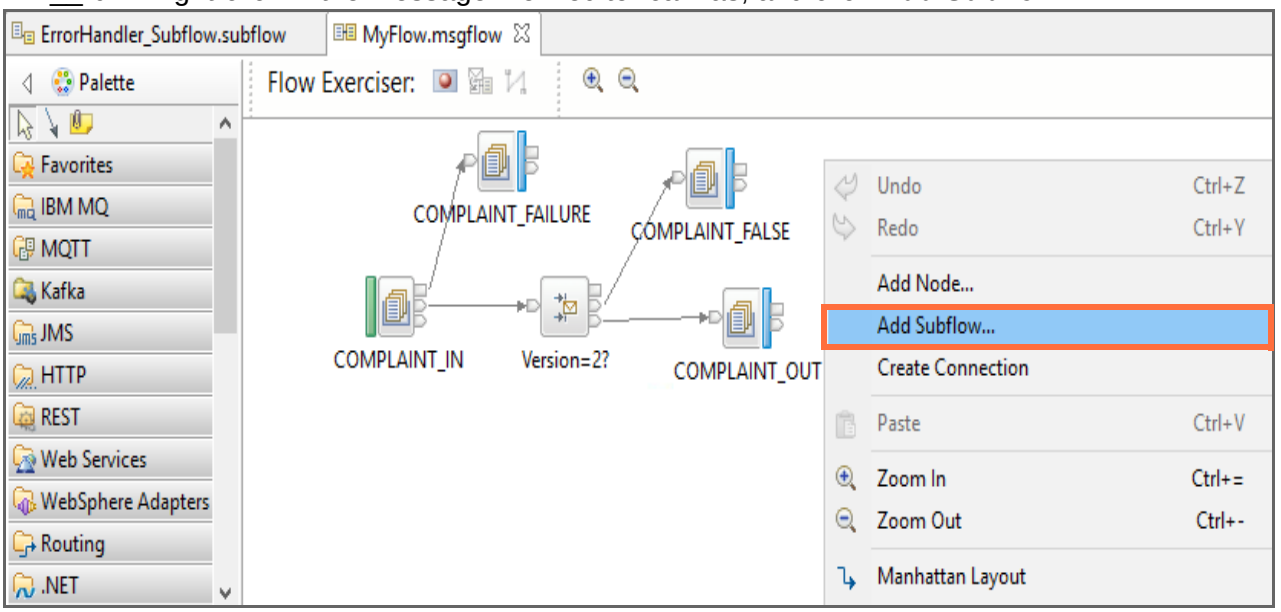
You now see the subflow file in the **Application Development** view, with the main flow that you started with at the beginning of the exercise.

Part 3: Add the subflow reference to the main flow

__ 1. Add the subflow reference to the main flow

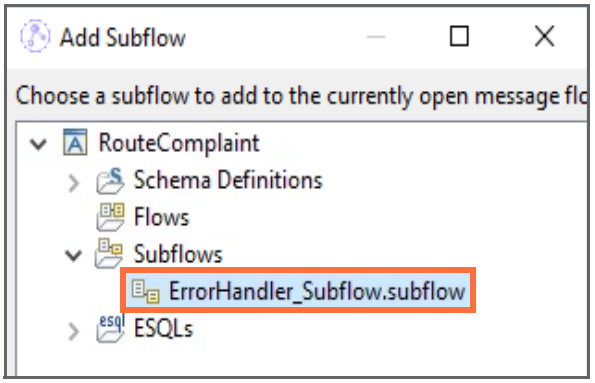
__ a. Open the main flow **Myflow.msgflow** (under **RouteComplaint > Flows**) in the Message Flow editor.

__ b. Right-click in the Message Flow editor canvas, and click **Add Subflow**.



The **Add Subflow** window is displayed.

__ c. In the Add Subflow window, expand **RouteComplaint > Subflows**, click **ErrorHandler_Subflow.subflow**

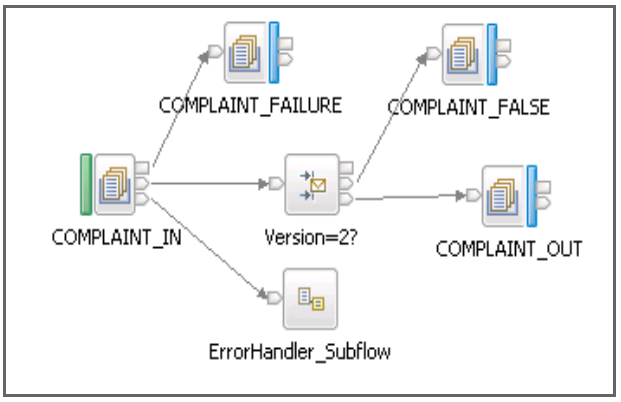


__ d. Click **OK**.

The Subflow node **ErrorHandler_Subflow** is added to the Message Flow editor canvas.

__ e. Position the Subflow node below and to the right of the MQInput node (COMPLAINT_IN).

__ f. Wire the MQInput node (COMPLAINT_IN) **Catch** terminal to the **Input** terminal of the ErrorHandler_Subflow node.



__ 2. Update the queue manager for the MQInput node.

This message flow was developed in a prior version of the product. You need to update the queue manager to use the ACEQM queue manager.

__ a. Under the MQ Connection tab for the COMPLAINT_IN node, update the **Destination queue manager name** to `ACEQM`

__ b. Repeat the above step for the COMPLAINT_FAILURE, COMPLAINT_FALSE, and COMPLAINT_OUT MQOutput nodes.

__ c. Save the message flow.

Part 4: Test the application

In this part of the exercise, you use the IBM App Connect Enterprise Toolkit Flow exerciser to test the message flow application. You also use the IBM MQ Explorer and RfhUtil to view the exception message.

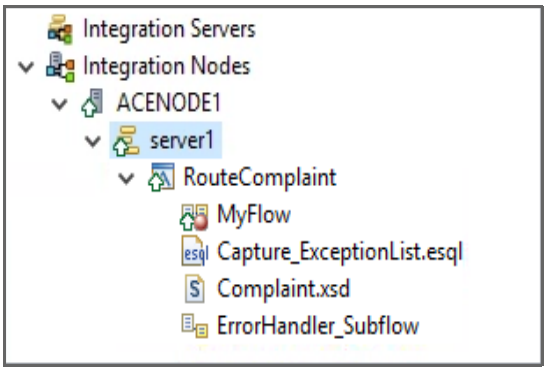
Test 1: Testing normal message flow processing

In this 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. Start the Flow exerciser

__ a. In the IBM App Connect Enterprise Toolkit Message Flow editor, start the Flow exerciser in the main message flow (**MyFlow.msgflow**).

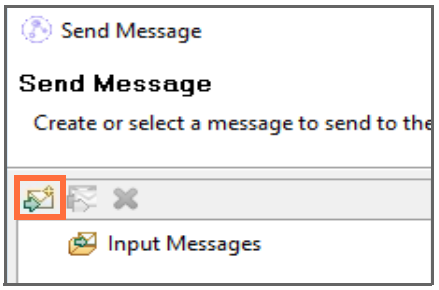
__ b. In the Integration Explorer view, verify that the message flow application deployed successfully and that the message flow is in record mode.



__ 2. 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.

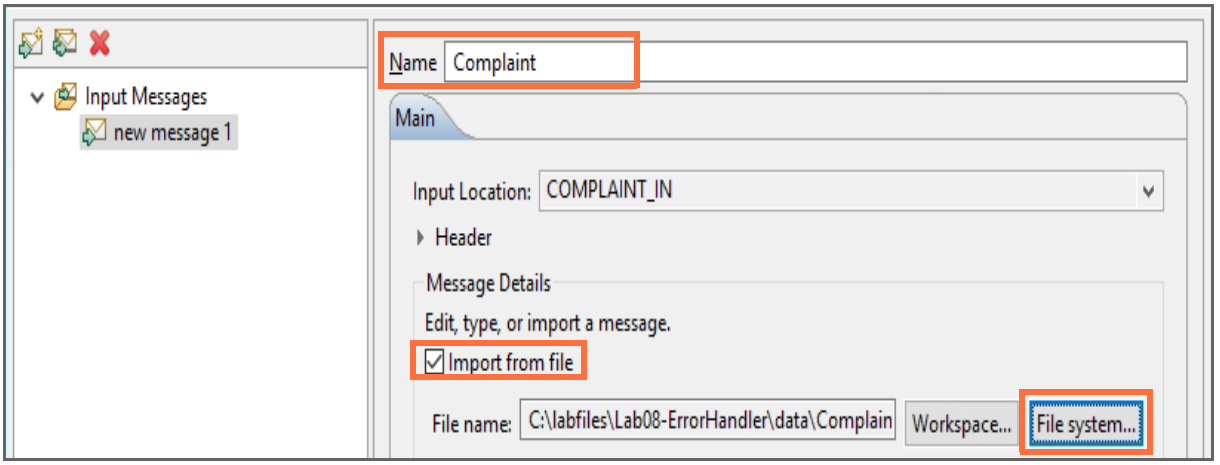
__ a. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

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



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

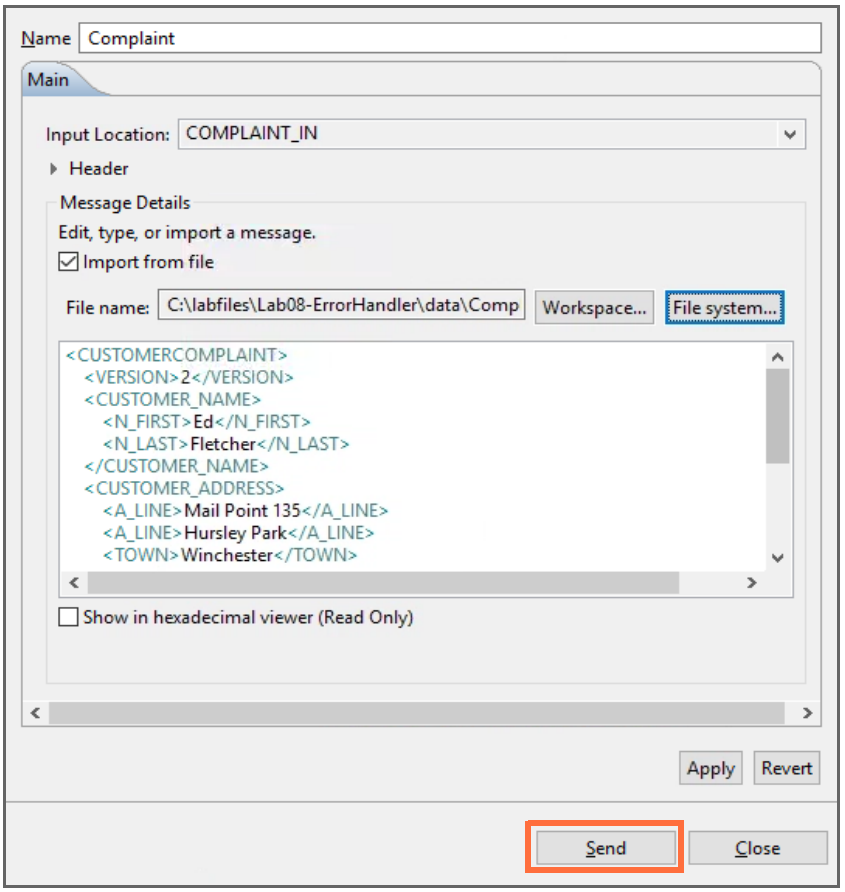
__ d. Select the **Import from file** checkbox and then click **File system**.



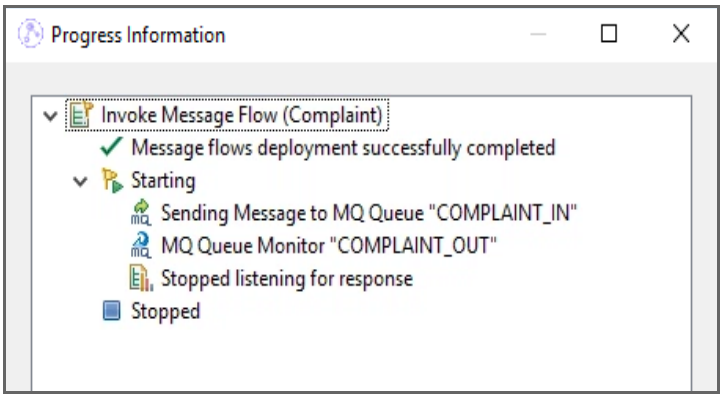
__ e. Go to the `C:\labfiles\Lab08-ErrorHandler\data` directory, click the `Complaint.xml` file, and then click **Open**.

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**.

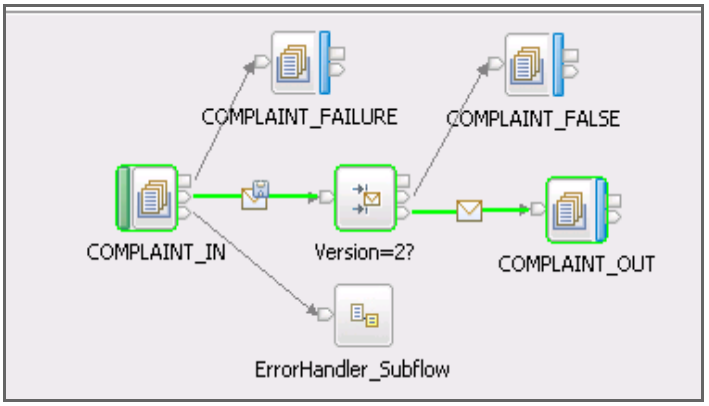


__ g. The **Progress Information** window shows that the message is sent to input queue COMPLAINT_IN. It also shows the destination, which is the COMPLAINT_OUT queue.



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

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



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 RfhUtil 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 ACEQM queue manager, type the following command in a command window:

```
amqsput COMPLAINT_IN ACEQM < 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.

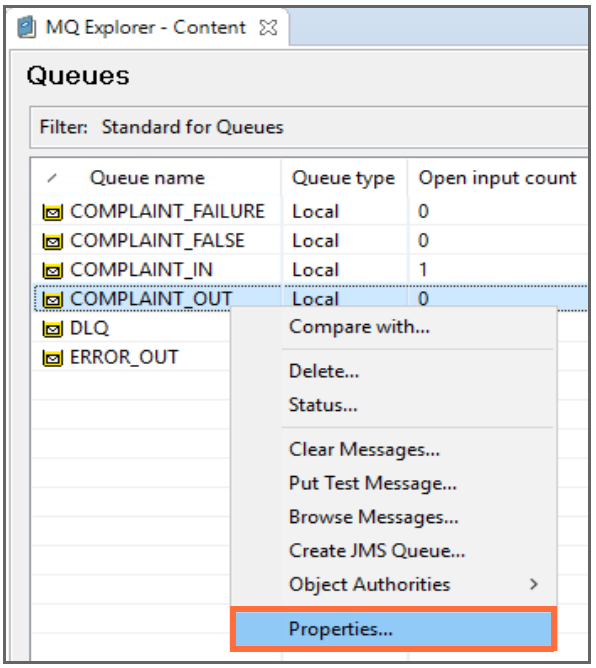
__ 1. To cause a runtime error, you set the COMPLAINT_OUT queue to the “Put inhibited” status so that the message cannot be written to the output queue.

__ a. Open IBM MQ Explorer if it is not already open.

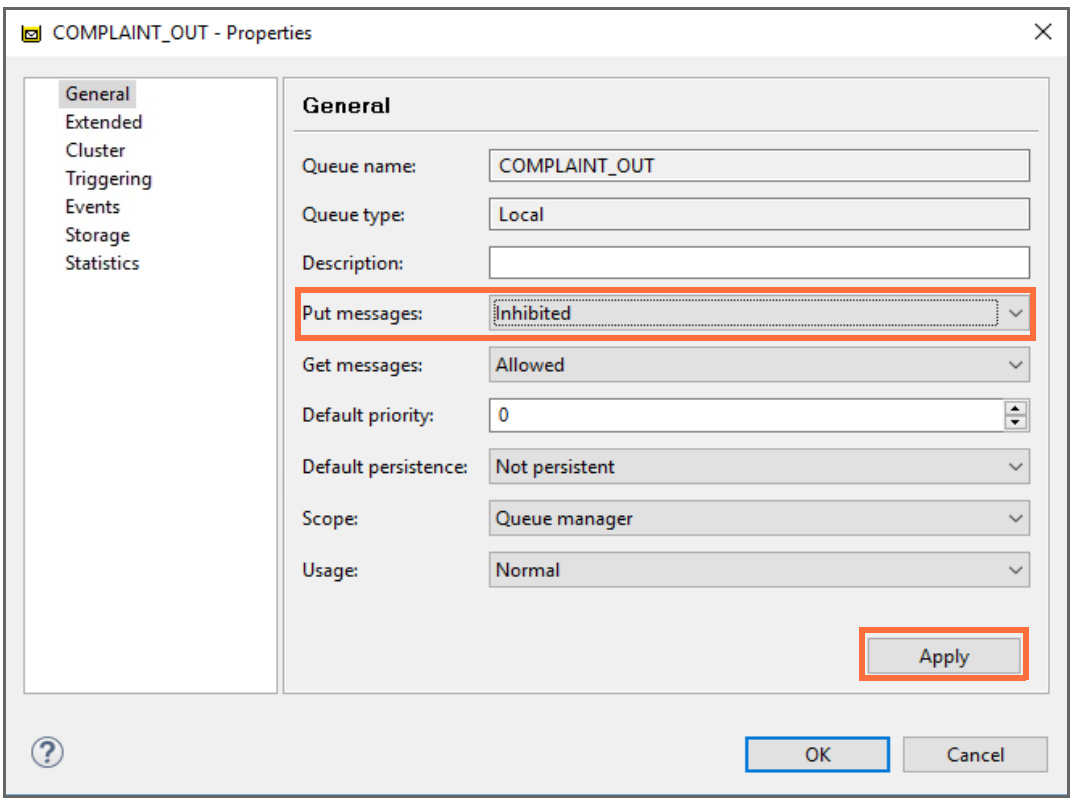
__ b. In the **MQ Explorer - Navigator** view, expand **Queue Managers > ACEQM**.

__ c. Click **Queues** to display the **Queues** content view.

__ d. Right-click COMPLAINT_OUT and then click **Properties**. The queue **Properties** window is displayed.

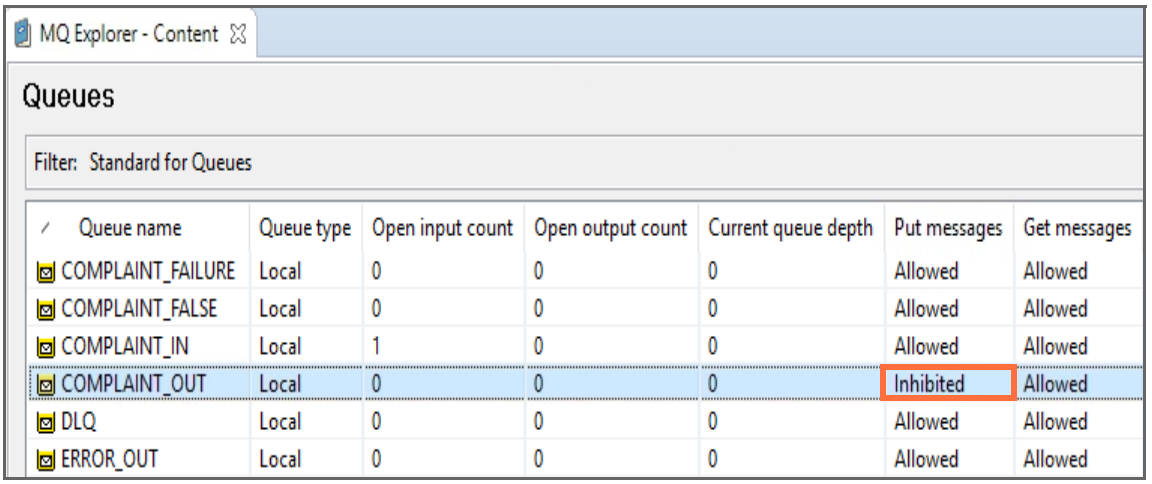


e. On the **General** tab change the setting for **Put messages** to **Inhibited** and then click **Apply**.



__ 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.



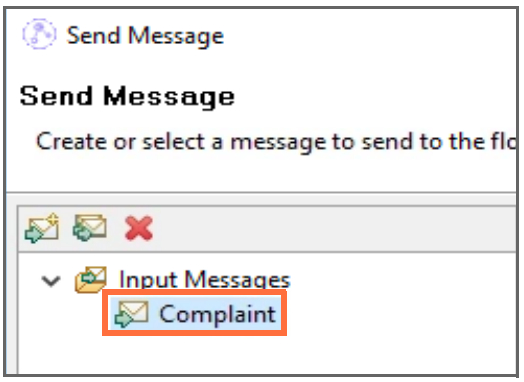
Queue name	Queue type	Open input count	Open output count	Current queue depth	Put messages	Get messages
COMPLAINT_FAILURE	Local	0	0	0	Allowed	Allowed
COMPLAINT_FALSE	Local	0	0	0	Allowed	Allowed
COMPLAINT_IN	Local	1	0	0	Allowed	Allowed
COMPLAINT_OUT	Local	0	0	0	Inhibited	Allowed
DLQ	Local	0	0	0	Allowed	Allowed
ERROR_OUT	Local	0	0	0	Allowed	Allowed

__ 2. Send the test message.

__ a. Return to the IBM App Connect Enterprise Toolkit.

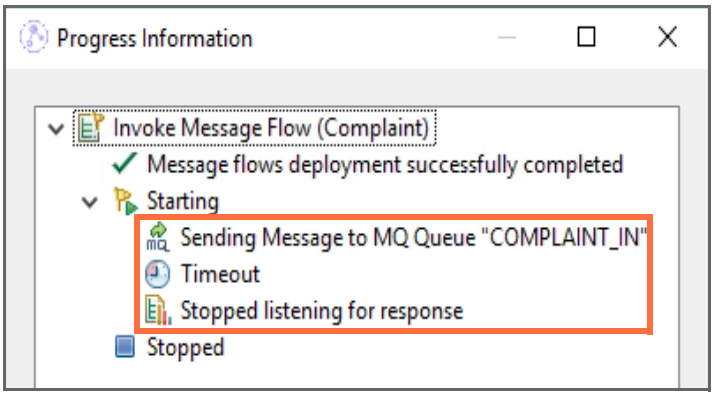
__ b. Click the **Send a message to flow** icon in the Flow Exerciser toolbar.

__ c. On the **Send Message** window, click **Complaint** under the **Input Messages** folder.



__ d. Click **Send**.

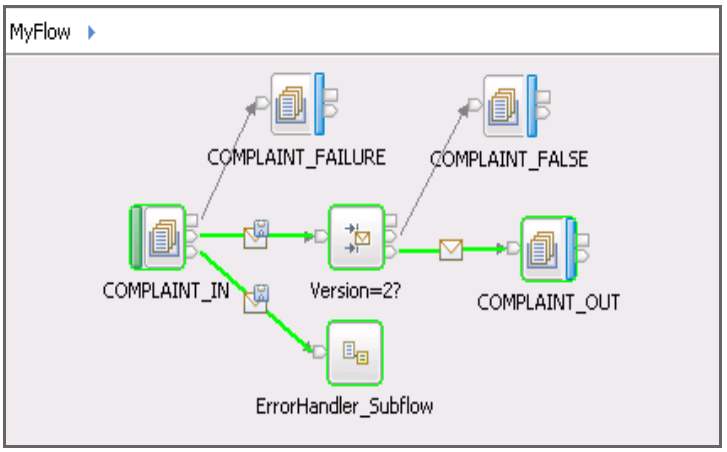
Wait for the Timeout event (two minutes) and the notification that the Flow exerciser stopped listening for a response.



Click **Close** to view the path.

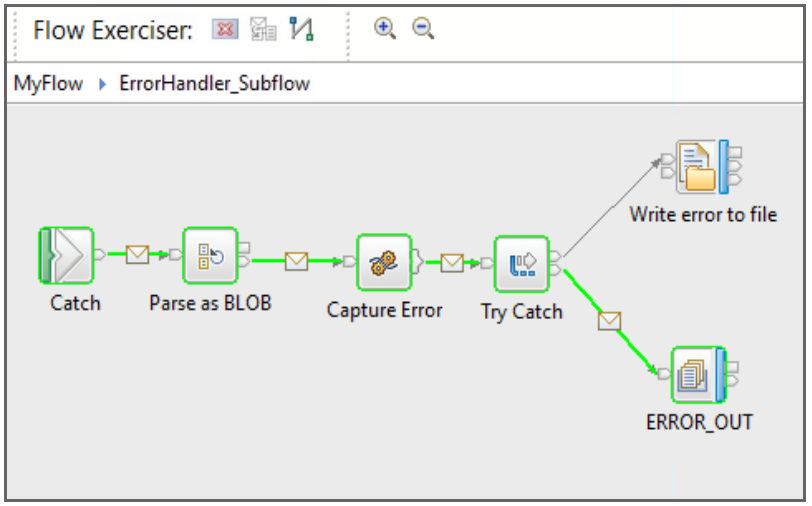
__ 3. View the message.

__ a. Based on the path in the 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.



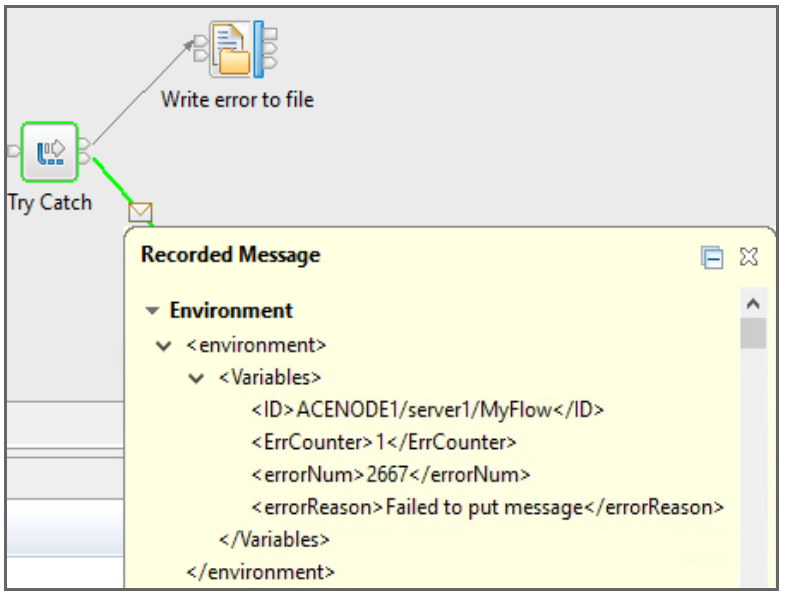
The message rolls back to the COMPLAINT_IN node because the **Failure** terminal on the COMPLAINT_OUT node is not wired. The message is then sent out the COMPLAINT_IN node **Catch** terminal because it is wired.

__ b. Double-click the **ErrorHandler_Subflow** node in the main flow to display the subflow path.



In the ErrorHandler_Subflow view, you see that the message is sent down the **Try** path of the TryCatch node to the ERROR_OUT queue.

__ c. 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 error:

```
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.

The 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;
```

__ d. Close the message.

__ 4. Use RfhUtil to verify contents of the **MQRFH2.usr** folder.

In the CaptureError Compute node, the ESQL code writes the exception information to the **MQRFH2.usr** folder.

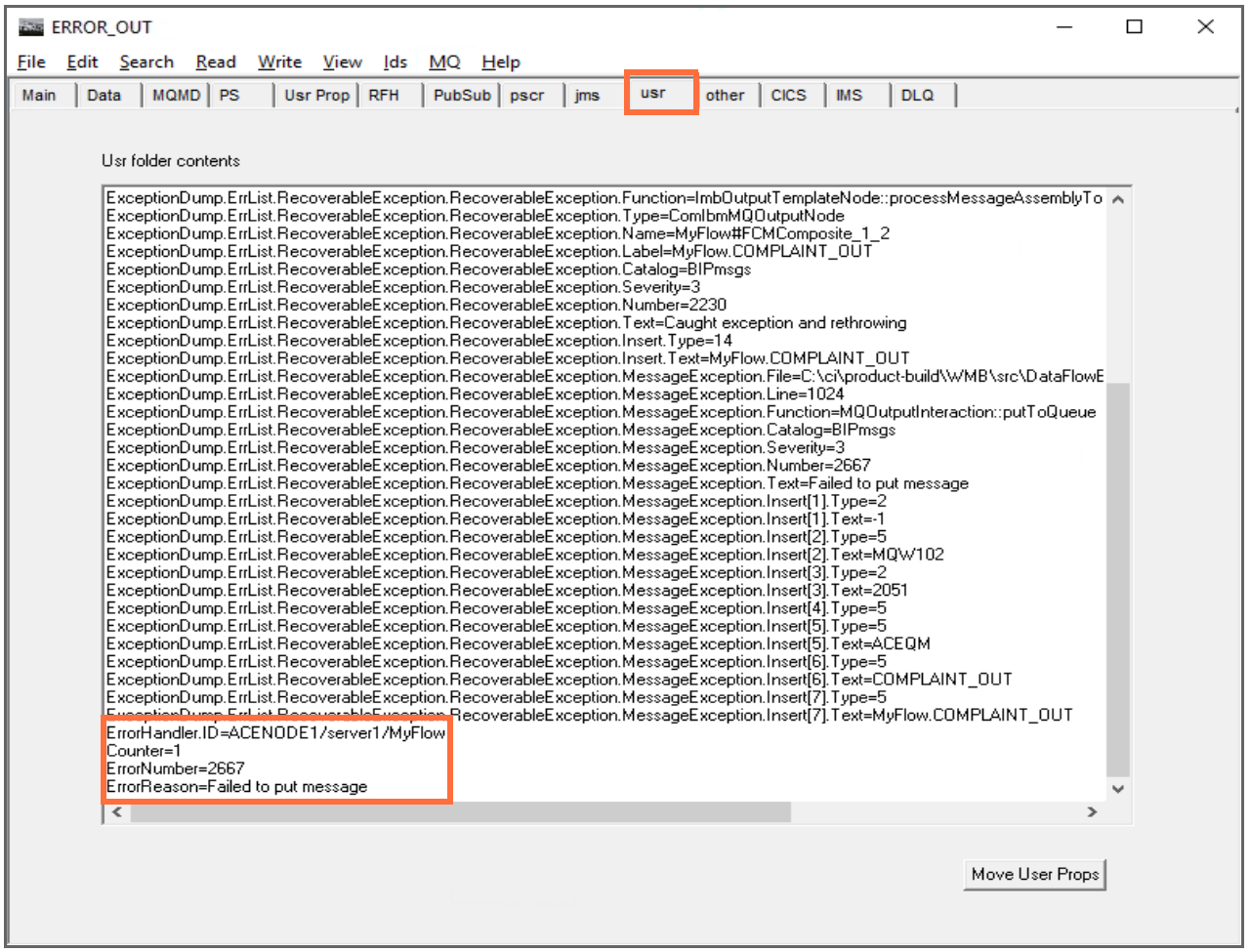
__ a. Start RFHutil by double-clicking the shortcut icon on the desktop.

__ b. On **Main** tab, set **Queue manager name** to **ACEQM** and **Queue name** to **ERROR_OUT**.

__ c. Click **Browse Q**.

__ d. Click the **usr** tab to display the contents of **MQFRH2.usr** folder.

__ e. Scroll to bottom of the **usr** folder contents pane to see the error summary that the **Capture Error** Compute node creates.



__ f. 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.

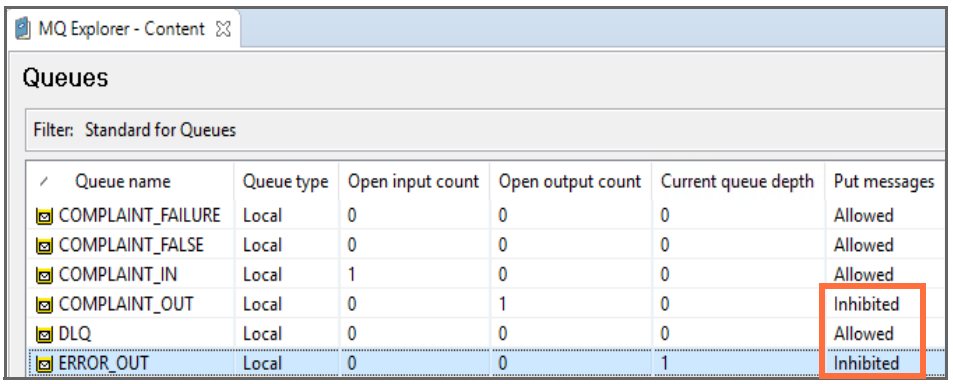
__ 1. Set the ERROR_OUT queue to inhibit PUT operations.

__ a. In IBM MQ Explorer, right-click the ERROR_OUT queue in the **Queues** view, and then click **Properties**.

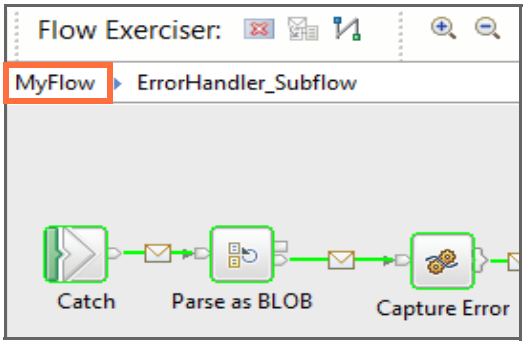
__ b. On the **General** tab, set the **Put messages** property to **Inhibited**.

__ c. Click **Apply** and then click **OK**.

__ d. In the list of queues, both ERROR_OUT and COMPLAINT_OUT should now show that the **Put messages** property is now **Inhibited**.



__ e. In the IBM App Connect Enterprise Toolkit Flow exerciser, return to main flow by clicking **MyFlow** in Flow exerciser breadcrumb trail.



__ 2. Send the test message again.

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

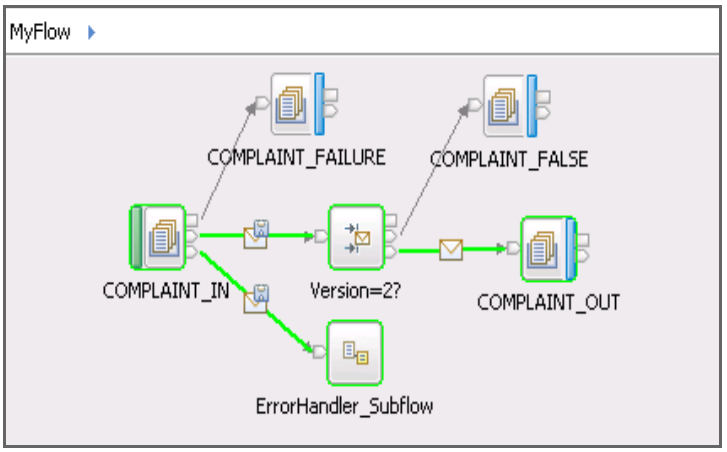
__ b. On the **Send Message** window, click **Complaint** under the **Input Messages** folder.

__ c. Click **Send**.

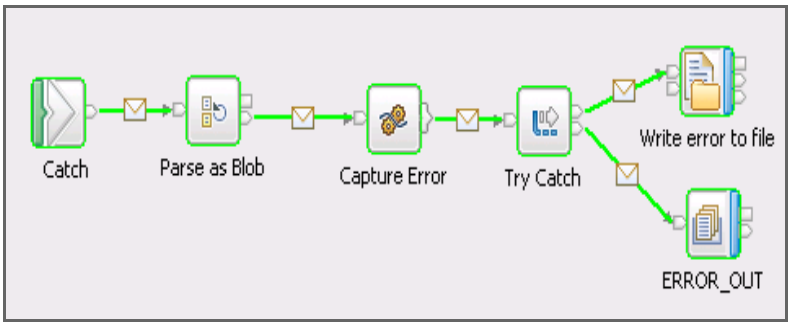
__ d. Wait for the timeout event and notification that the Flow exerciser stopped listening for a response. Click **Close** to view the path.

__ 3. View the message.

__ a. 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.



__ b. Double-click the **ErrorHandler_Subflow** node in the main flow to display the subflow path.

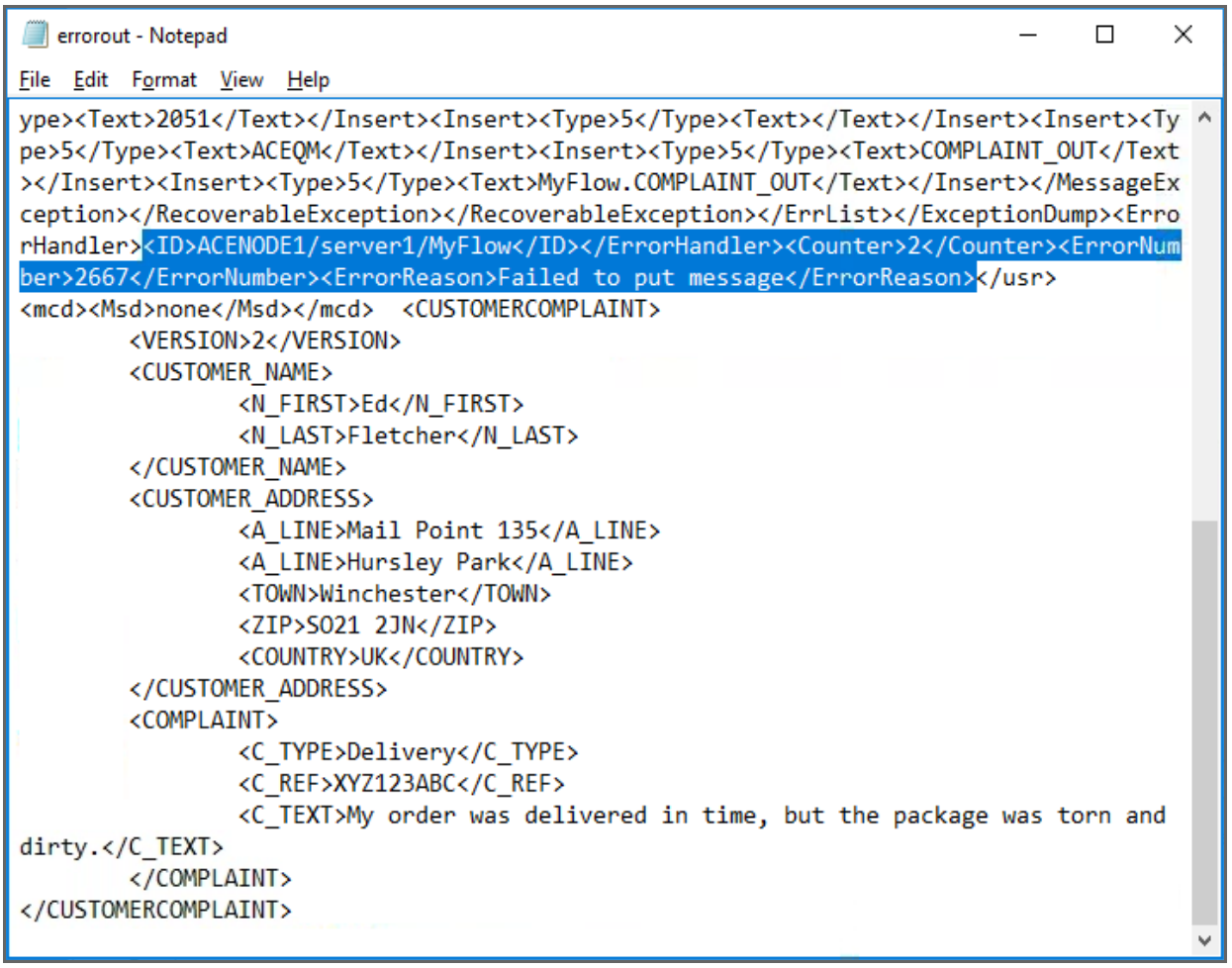


This time, when the TryCatch node in the ErrorHandler_Subflow attempts to write the error message to ERROR_OUT, it fails. So, as shown in the highlighted path, the message is sent down the **Catch** path of the Try Catch node to the **Write error to file** FileOutput node.

__ c. To verify that the exception information is written to the output file: `C:\labfiles\errorout.txt`

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.



__ d. Close Notepad.

Part 5: Exercise clean-up

__ 1. In the IBM App Connect Enterprise Toolkit **Integration Explorer view**, stop recording on the message flow.

__ 2. Delete all resources from **server1** integration server on **ACENODE1**.

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

__ 4. In IBM MQ Explorer, delete the queues

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 App Connect Enterprise 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

Estimated time

01:30

Overview

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.

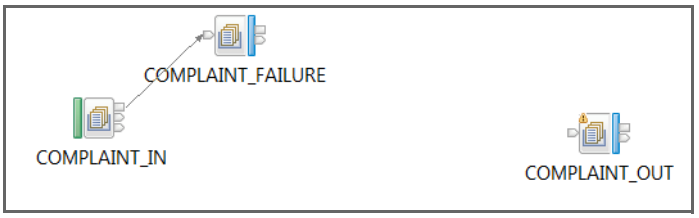
Objectives

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
- Define database connectivity
- 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.
```

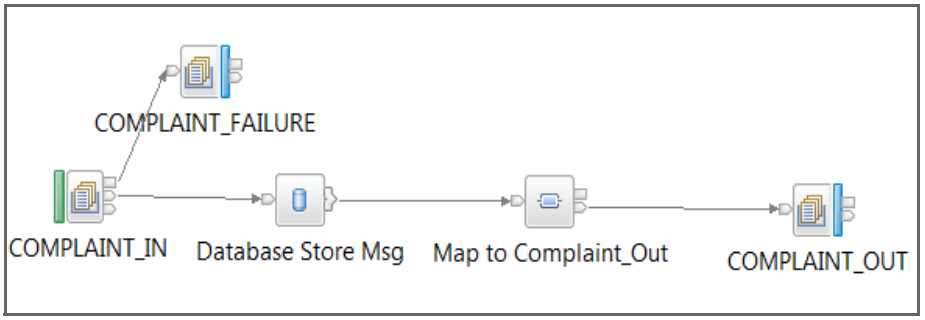
The output message is an XML message that the `Complaint_Out.xsd` schema defines. An example of the output message is shown here.

```
<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 subsequent 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 taken 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 App Connect Enterprise V11 Toolkit, IBM MQ V9, and IBM DB2
- A local IBM MQ queue manager that is named ACEQM with the following local queues: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_OUT, and DLQ
- Lab files in the `C:\labfiles\Lab09-DBMap` directory
- User `aceadmin` that is a member of the “mqm” group
- Users `aceadmin` 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 source that are required for this exercise are already created on the lab exercise image.

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab09`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, enter `C:\Workspace\Lab09`

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

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

__ 2. Verify that the node and server are running.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 3. This exercise requires an IBM MQ queue manager.

If you completed Exercise 3, you created a queue manager that is named ACEQM. You can use that queue manager in this exercise. Proceed to the next step.

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

__ a. Start IBM MQ Explorer.

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

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

__ 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.

__ 4. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

In a command window, type:

```
runmqsc ACEQM < C:\labfiles\Lab09-DBMap\CreateQueues.mqsc
```

__ 5. Use IBM MQ Explorer to verify that the queues were created.

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

__ 6. Import the project interchange file that is named `DB_StartingApp_PI.zip` file from the `C:\labfiles\Lab09-DBMap` directory into your workspace.

__ a. From the IBM App Connect Enterprise Toolkit, click **File > Import**.

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

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

__ d. Browse to the `C:\labfiles\Lab09-DBMap` directory.

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

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

The **RouteComplaint** application contains a partial message flow that is named `DB.msgflow`.

Part 2: Create a shared library that contains the data models

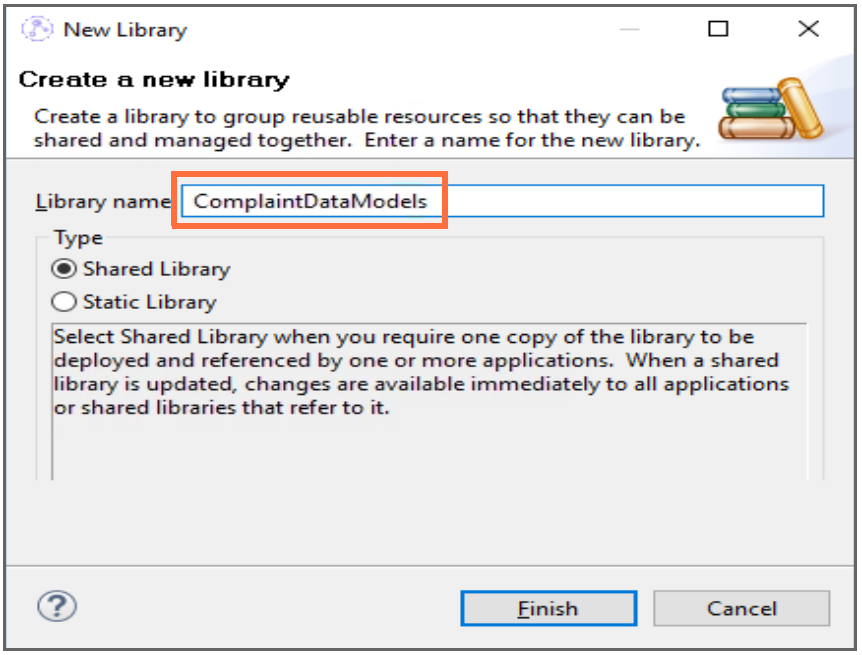
In this part of the exercise, you create a shared library and then create the data models for the COBOL Copybook that defines the input file and the XML schema that defines the output file. You then reference the shared library in the RouteComplaint application.

__ 1. Create a shared library that is named **ComplaintDataModels**.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > New > Library**.

__ b. Ensure that **Shared Library** is selected.

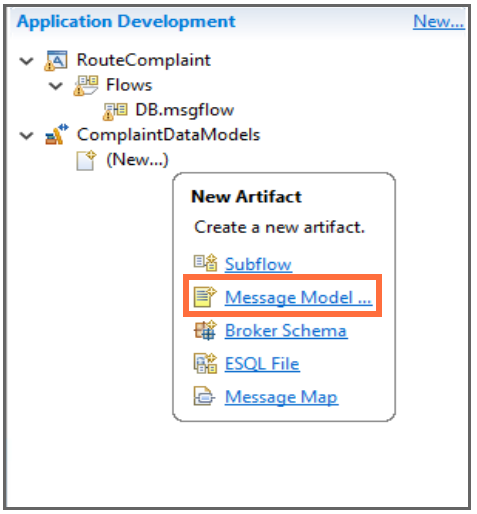
__ c. For the **Library Name**, type: **`ComplaintDataModels`**



__ 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.



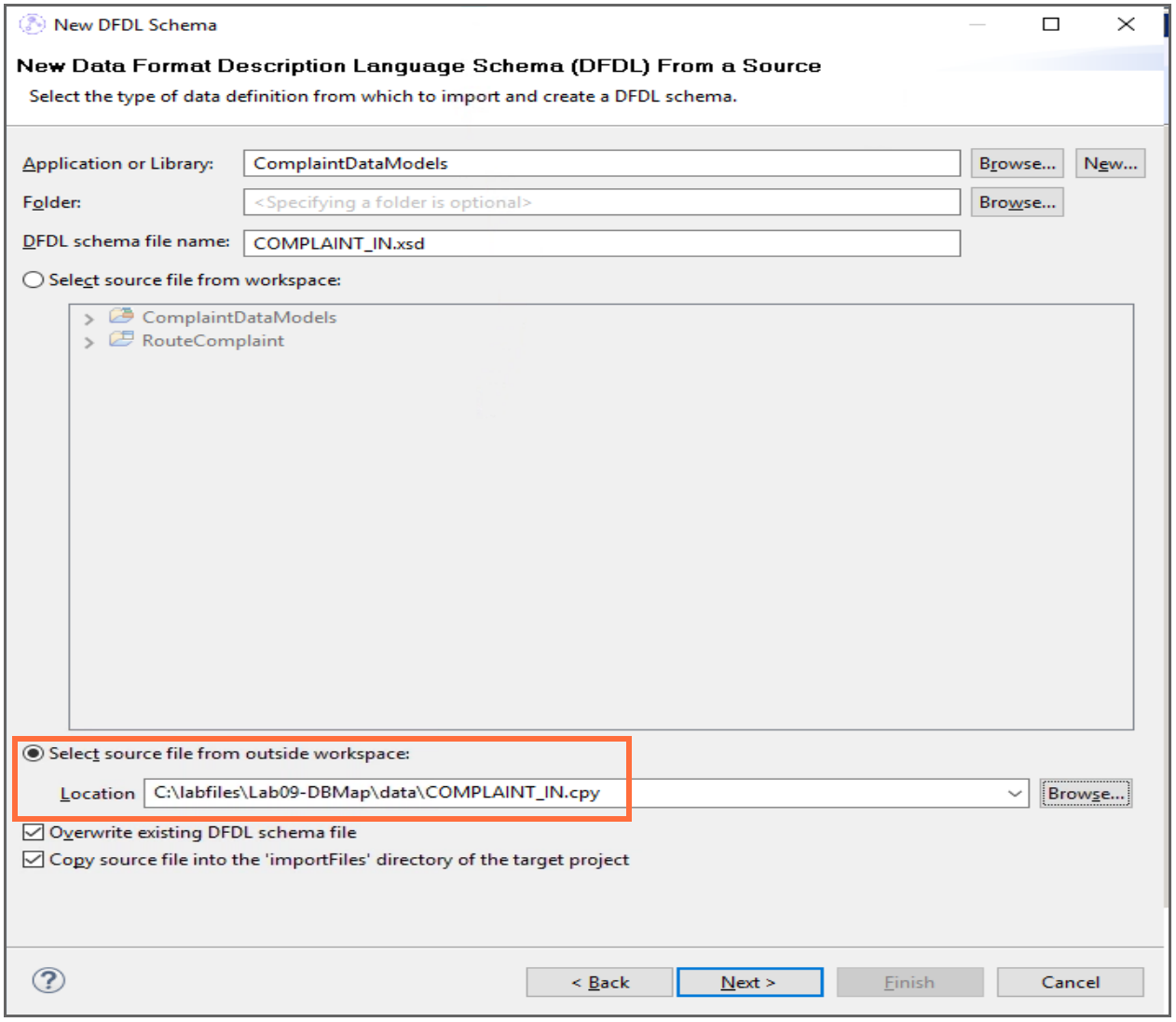
__ b. Click **COBOL** under the **Text and binary** section and then click **Next**.

__ 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**.

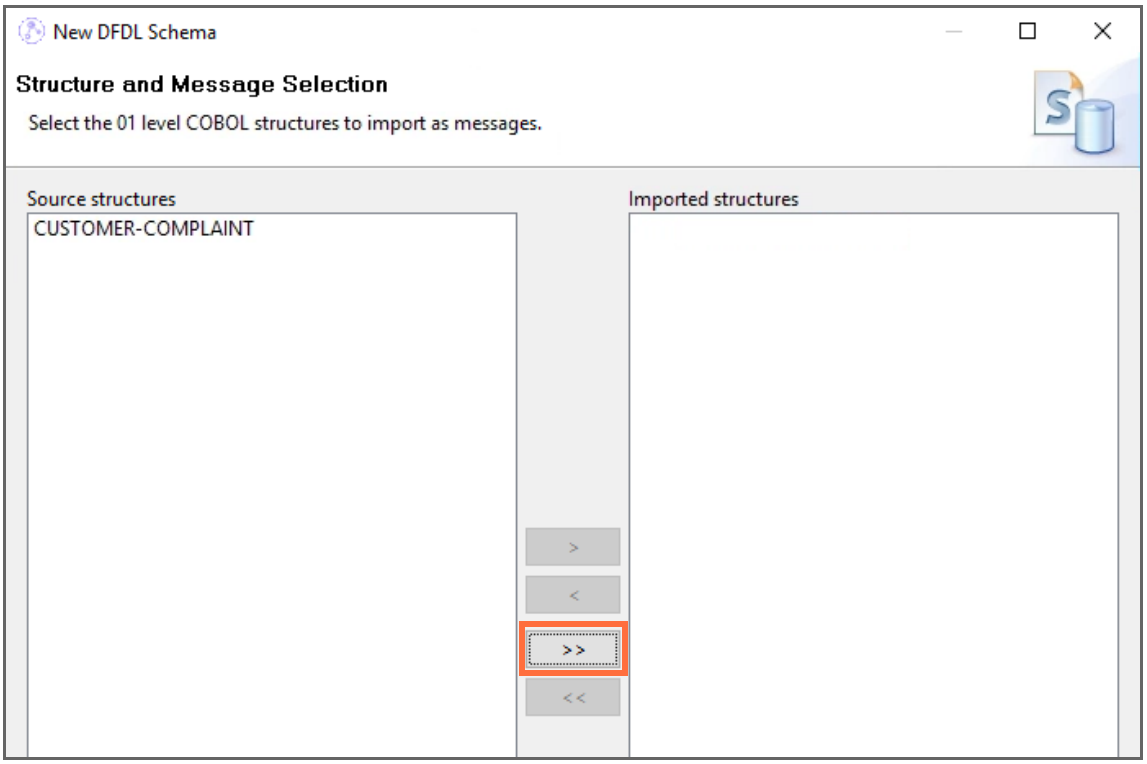


__ g. Click **Next**. The COBOL Copybook is read and analyzed.

__ 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.

__ k. Review the **COMPLAINT_IN.xsd** schema so that you understand its structure and elements.

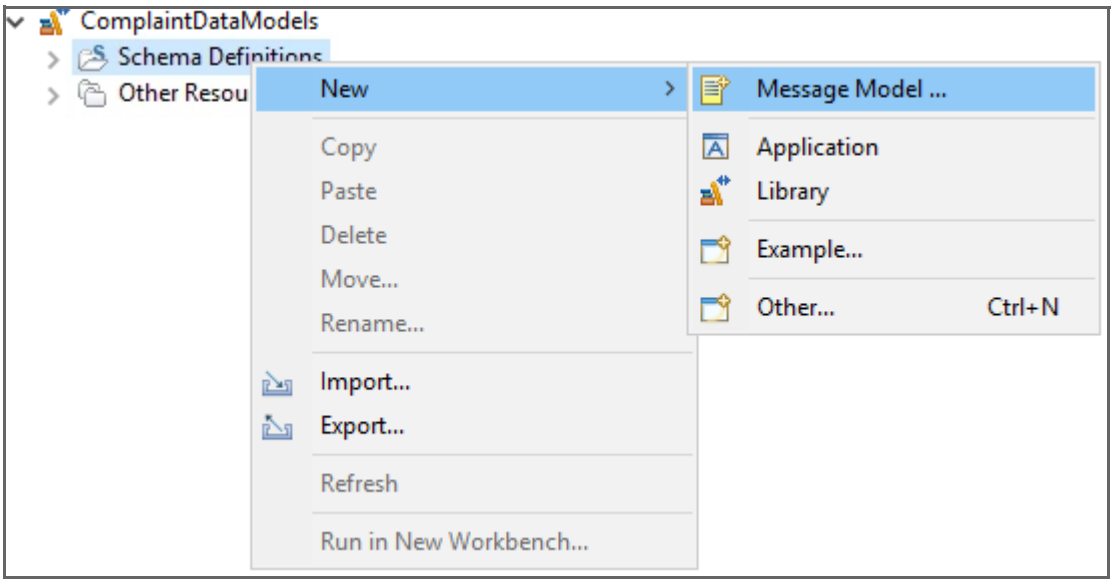
Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
CUSTOMERCOMPLAINT	CUSTOMERCOMPLAINT				
sequence		1	1		
VERSION	<PIC9-Display-Zoned_short>	1	1	0	0
CUSTOMER_NAME		1	1		
sequence		1	1		
N_FIRST	<PICX_string>	1	1		
N_LAST	<PICX_string>	1	1		
CUSTOMER_ADDRESS		1	1		
sequence		1	1		
A_LINE	<PICX_string>	2	2		
TOWN	<PICX_string>	1	1		
ZIP	<PICX_string>	1	1		
COUNTRY	<PICX_string>	1	1		
COMPLAINT		1	1		
sequence		1	1		
C_TYPE	<PICX_string>	1	1		
C_REF	<PICX_string>	1	1		
C_TEXT	<PICX_string>	1	1		

Add a Local Element

__ l. Close the schema editor when you are finished reviewing the schema.

__ 3. 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** pane, right-click **Schema Definitions** under the **ComplaintDataModels** library folder and then click **New > Message Model**.



The New Message Model window is displayed.

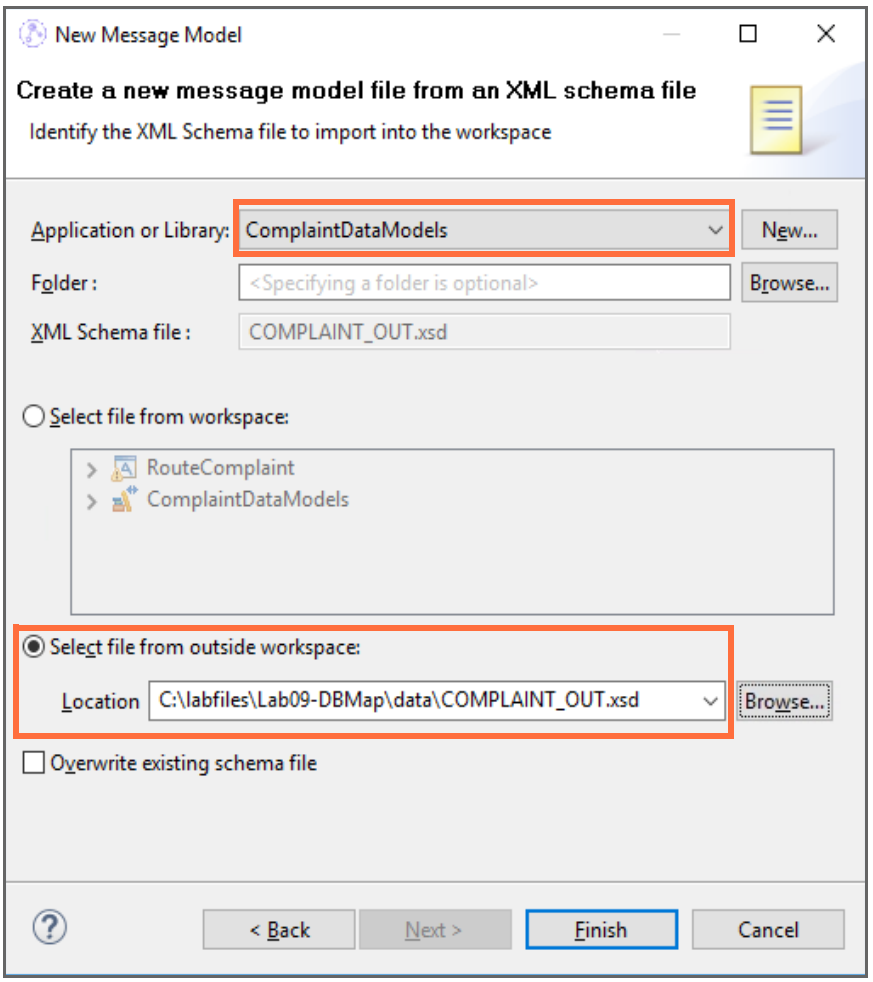
__ b. Under the **XML** section, click **Other XML** and then click **Next**.

__ c. On the **Other XML** page, click **I already have an XML schema file for my data** and then click **Next**.

__ d. Select **ComplaintDataModels** for the Application or Library.

__ e. Click **Select file from outside workspace** and then click **Browse**.

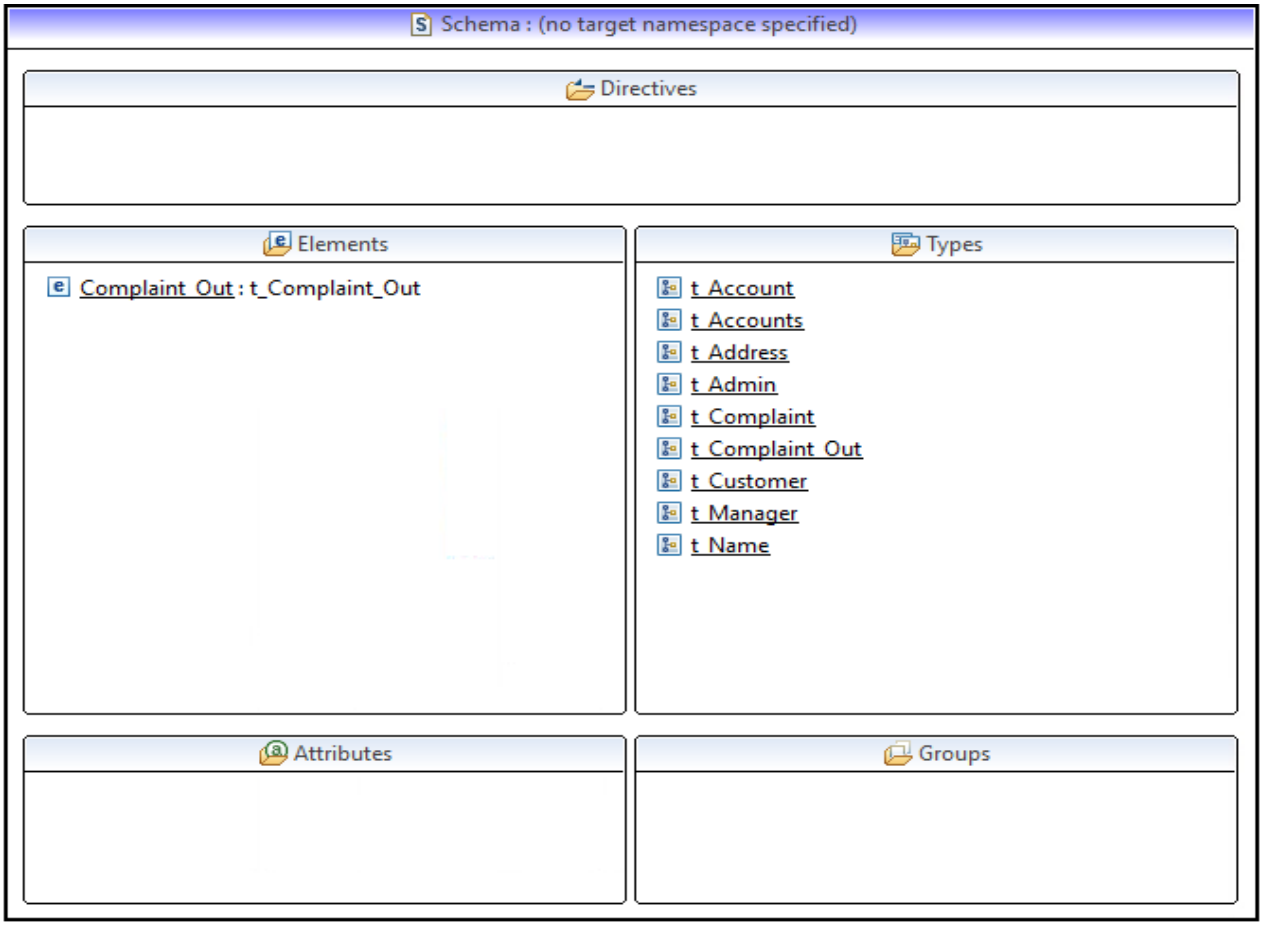
__ f. Browse to `C:\labfiles\Lab09-DBMap\data` directory, click **COMPLAINT_OUT.xsd**, and then click **Open**.



__ g. Click **Finish**. The XML schema is read and analyzed.

__ 4. Review the message definition.

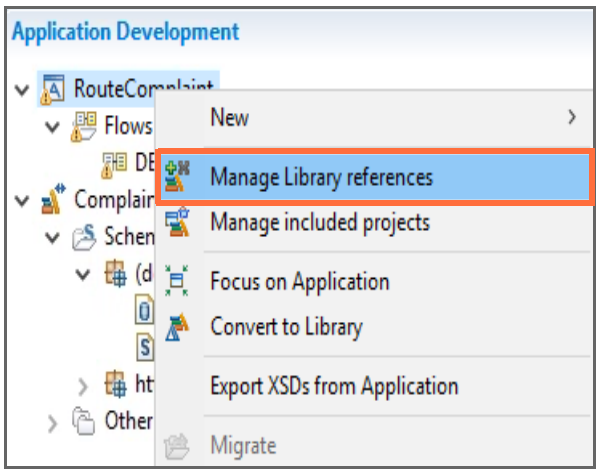
__ a. Review the newly created message definition **COMPLAINT_OUT.xsd** in the **Outline** view. The COMPLAINT_OUT.xsd file also opens in the schema editor.



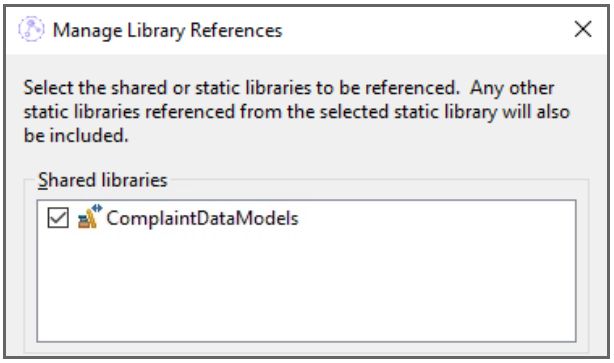
__ b. Close the schema editor when you are finished reviewing the schema.

__ 5. Reference the **ComplaintDataModels** shared library in the **RouteApplication** application.

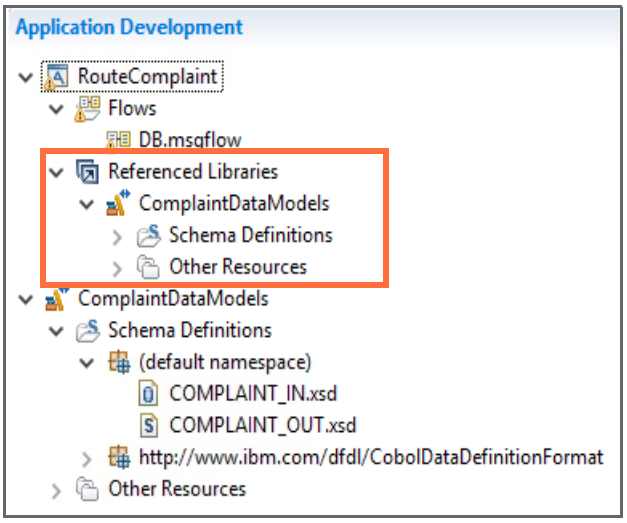
__ a. In the Application Development pane, right-click **RouteComplaint** and then click **Manage library references**.



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



__ c. A library reference is added to the **RouteComplaint** application.



Part 3: Discover the database definitions

In this part of the exercise, you define the connections to the SAMPLE and MSGSTORE databases in the IBM App Connect Enterprise Toolkit.

Note

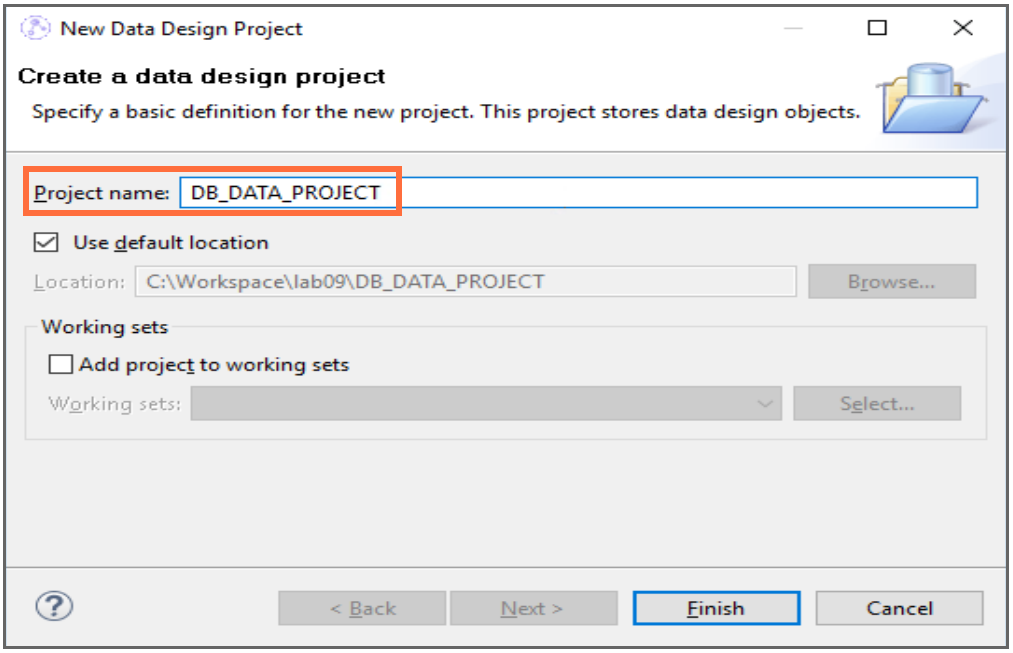
The DB2 database tables are already created on the lab exercise image.

__ 1. 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 database definition file** page, click **Next**.

__ f. On the **Select Connection** page, click **New**.

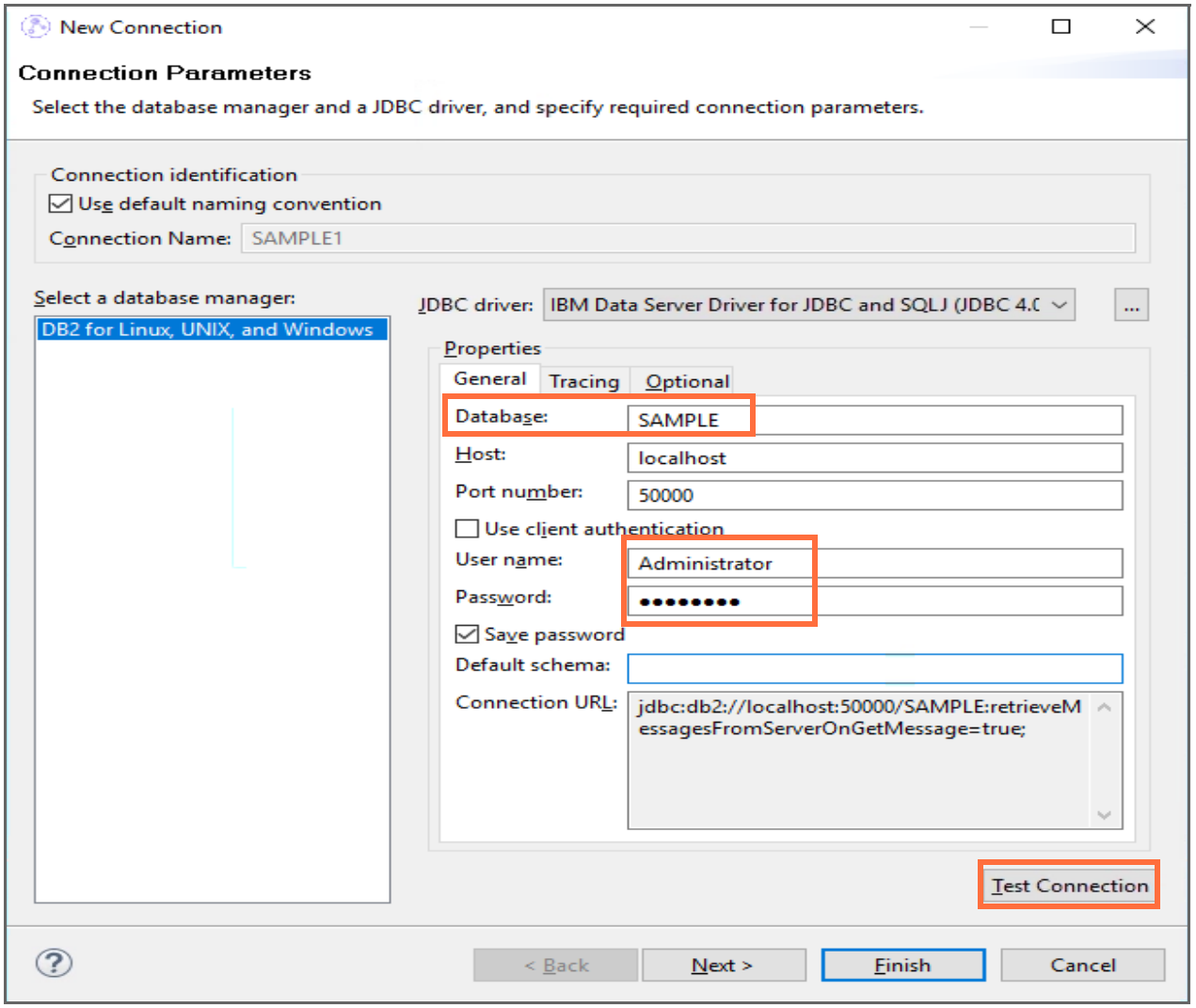
__ g. In the **Connection Parameters** window, for **Database**, type: `SAMPLE` (if it is not already entered)

__ h. For **User name**, type: `Administrator`

__ i. For **Password**, type: `passw0rd`

__ j. Click **Save password**.

__ k. Click **Test Connection** to test the JDBC connection.

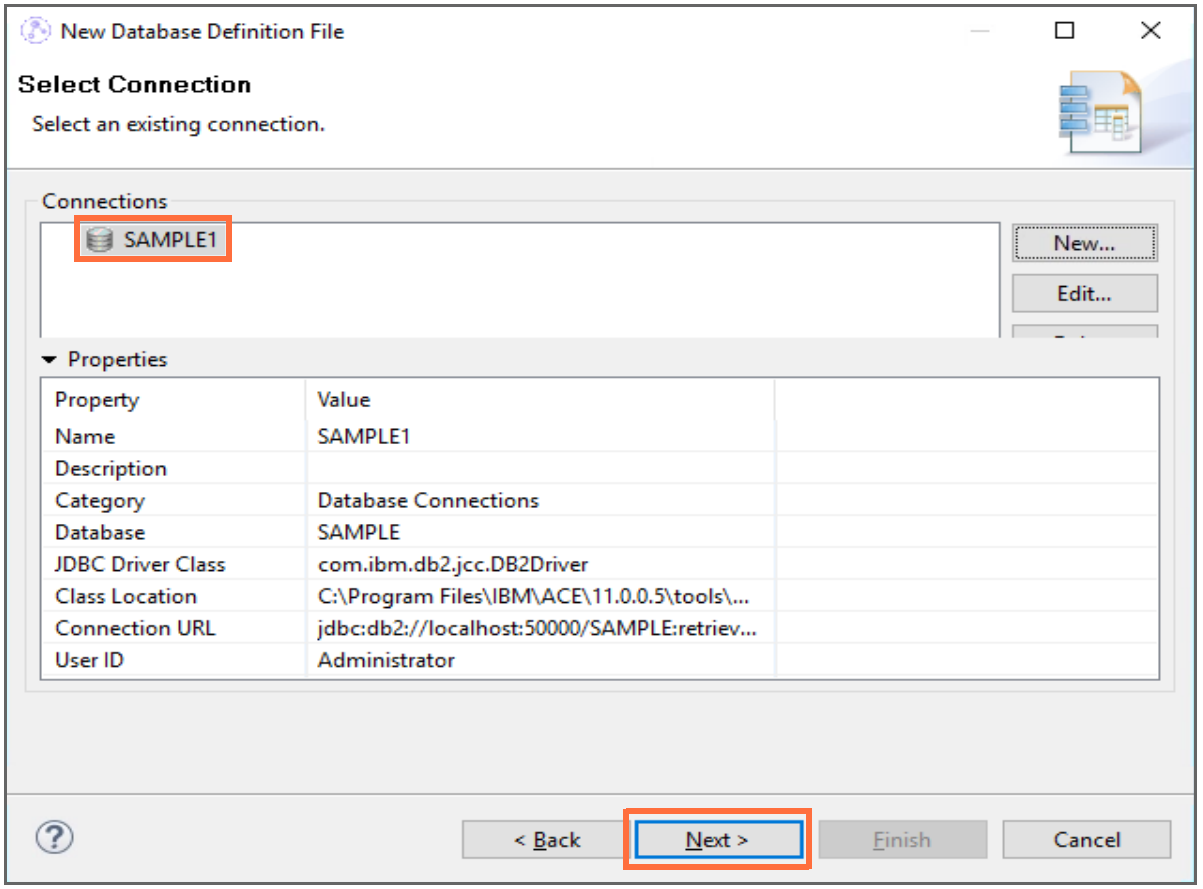


A “Connection succeeded” message is displayed.

__ l. Click **OK**.

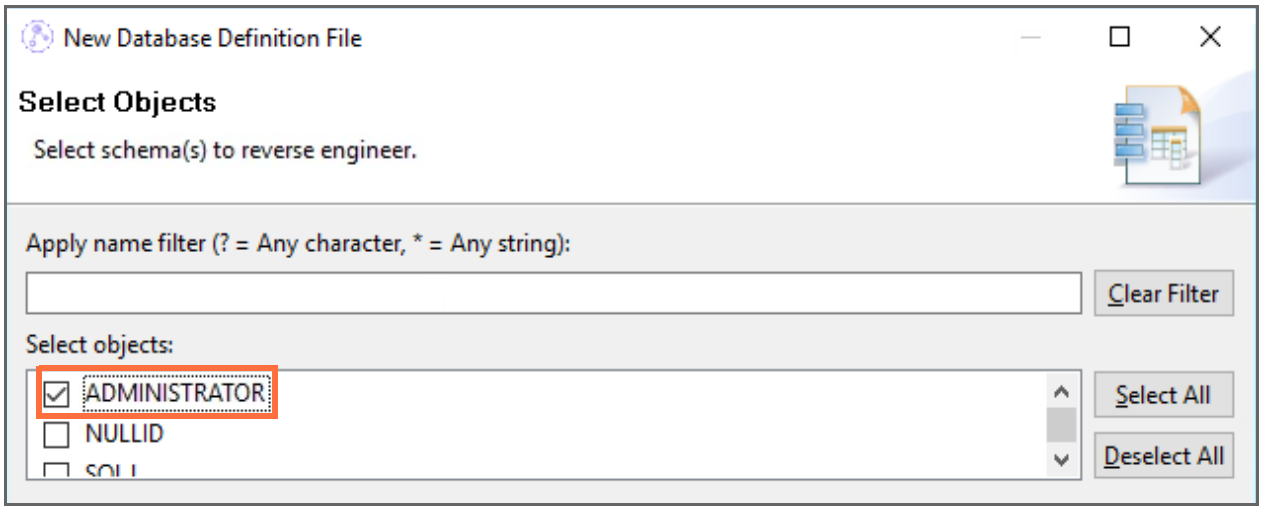
__ m. Click **Finish**. The **New Database Definition File** window displays the connection information for the SAMPLE database.

__ n. Click **SAMPLE1**, and then click **Next**.



Your screen might display SAMPLE or SAMPLE1.

__ o. Select **ADMINISTRATOR**

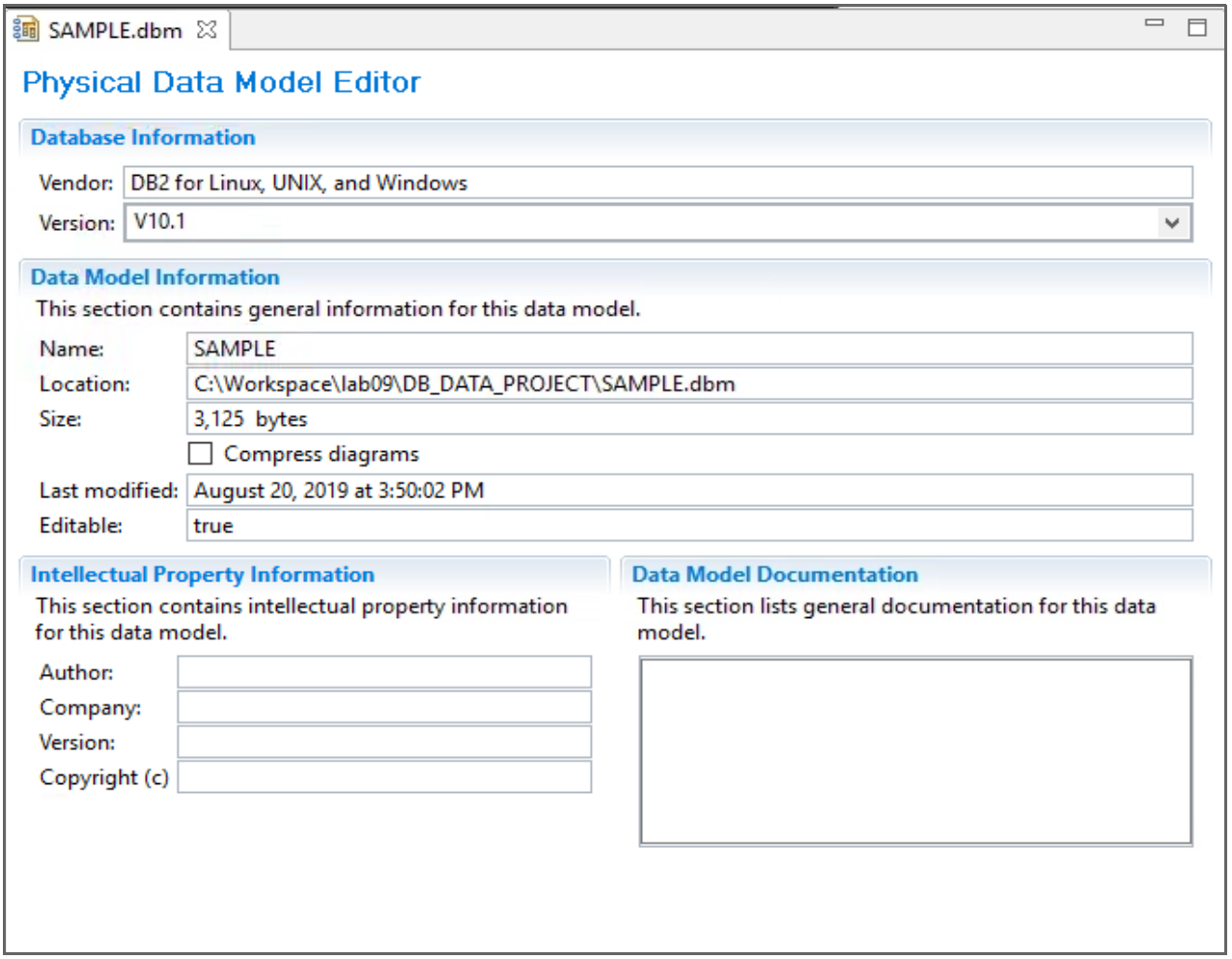


__ p. Click **Finish**.

The model is created. The database connection opens in the Physical Model data editor.

Review the model, but do not change anything.

If the model editor does not open automatically, you can double-click the **SAMPLE.dbm** in the Application Development pane.



__ q. Close the editor when you are finished reviewing the model.

__ 2. 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 **Data design 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: `passw0rd`

__ g. Click **Save password**.

__ h. Click **Test Connection.** The JDBC connection is tested.

A “Connection succeeded” message is shown.

__ i. Click **OK**.

The Select Connection menu reappears.

__ j. Click **Finish**.

__ k. Click **MSGSTORE1**, and then click **Next**.

The Select Objects window opens.

Your screen might display MSGSTORE or MSGSTORE1.

__ l. Click **ADMINISTRATOR**, and then click **Finish**.

The model is created. The database connection opens in the Physical Model data editor.

Review the model, but do not change anything.

__ m. 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.

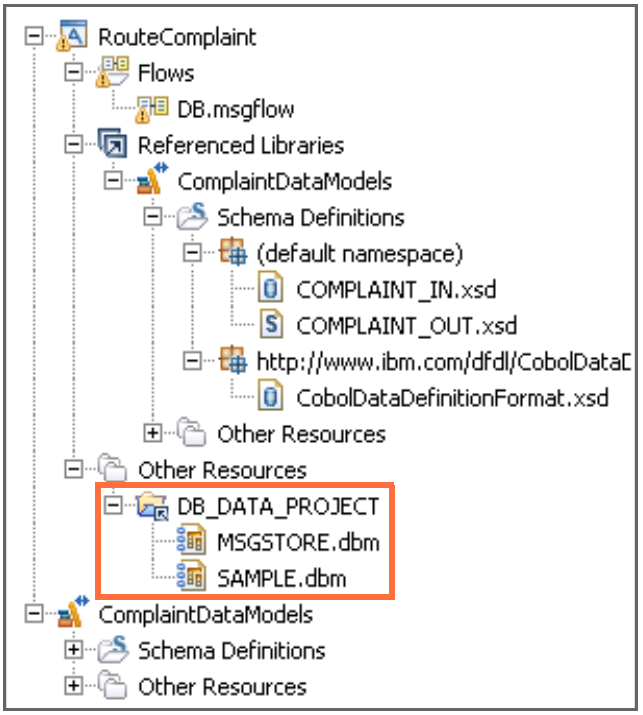
Database definition files are not automatically updated. If you change the database, you must re-create the database definition files.

__ 3. 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** pane 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.



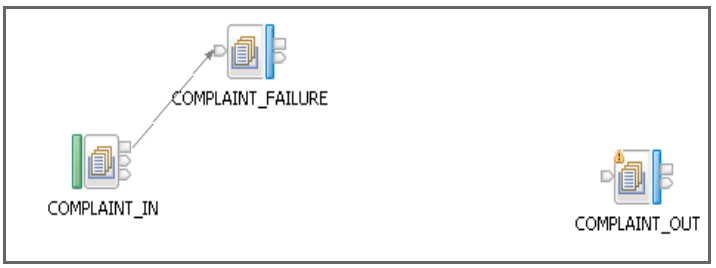
Part 4: Complete the message flow

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. Add nodes

__ a. 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.



__ b. Add a Database node from the **Database** drawer and rename it to: **Database Store Msg**.

__ c. Add a Mapping node from the **Transformation** drawer and rename it to: **Map to Complaint_Out**.

__ 2. Connect the nodes.

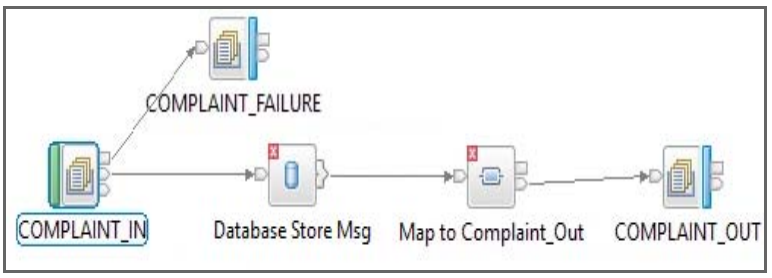
__ a. Wire the **COMPLAINT_IN** node **Out** terminal to the **Database Store Msg** node **In** terminal.

__ 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.

__ d. Save the message flow.

Your model will have errors until you configure the database and mapping nodes.



__ 3. Update the queue manager for the MQ Nodes

__ a. Click the COMPLAINT_FAILURE node.

__ b. Select the **MQ Connection** tab.

__ c. Change the **Destination queue manager name** from `IIBQM` to `ACEQM`.

__ d. Perform the same steps for the other two MQ Nodes: COMPLAINT_OUT and COMPLAINT_IN

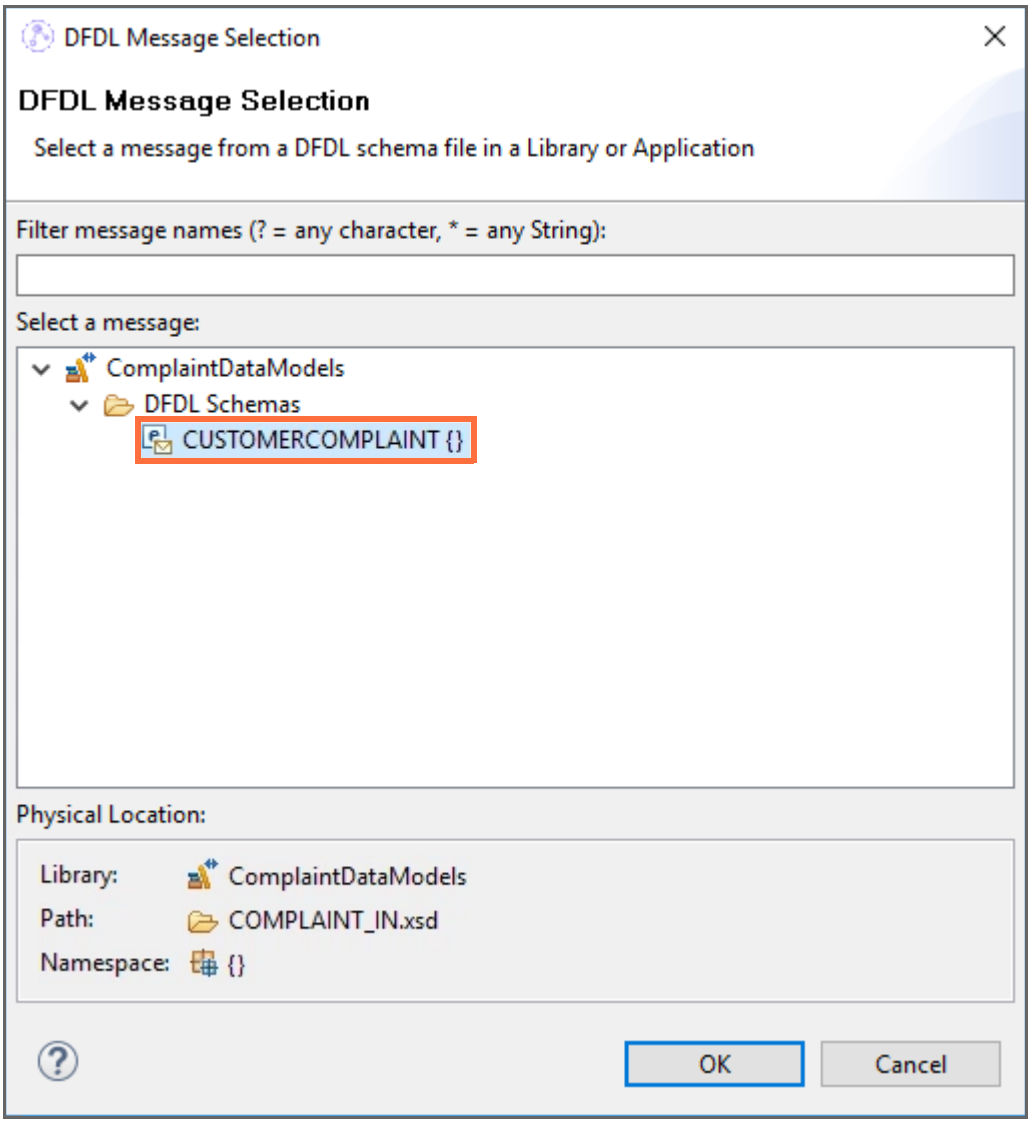
__ e. Save the model.

__ 4. Configure the COMPLAINT_IN node.

__ a. Select the **Input Message Parsing** tab for the COMPLAINT_IN node.

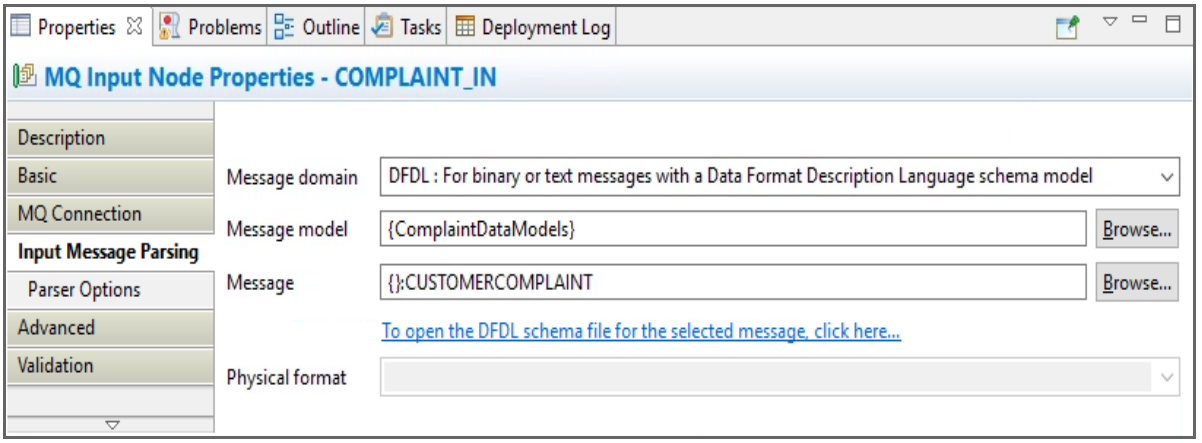
__ b. For the **Message domain,** select **DFDL**.

__ c. For the **Message**, click **Browse** and then click **CUSTOMERCOMPLAINT{}**.



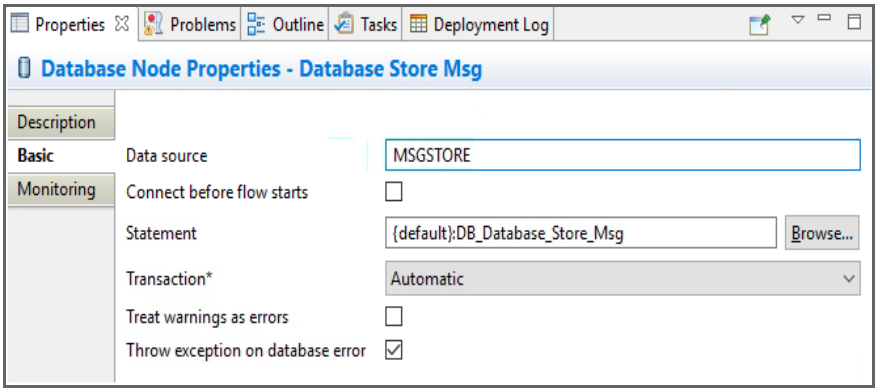
__ d. Click **OK**

__ e. Verify your configuration



__ 5. Configure the **Database Store Msg** node properties.

__ a. On the **Basic** properties tab, for **Data source** type: `MSGSTORE`



__ b. Double-click the **Database Store Msg** node.

The ESQL editor opens in a new module within the existing ESQL file.

__ c. 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;
```

__ d. Save the file and close the ESQL editor.

The error for the Database node is removed.

Note

Warning messages in the **Problems** view about ambiguous database table references are OK; the references are resolved at run time.

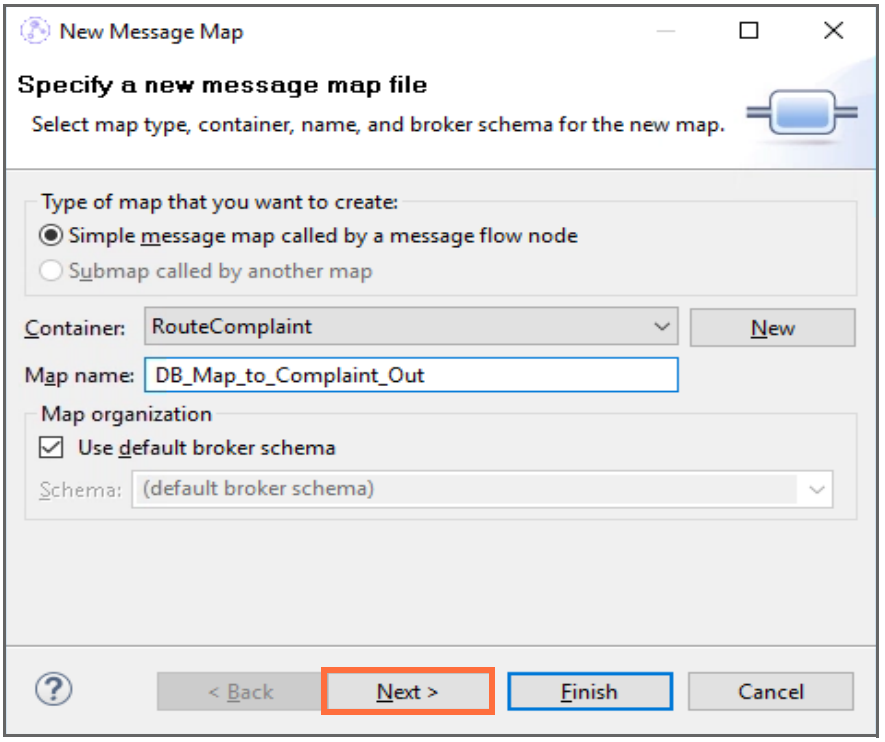
Part 5: Create the mappings

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.

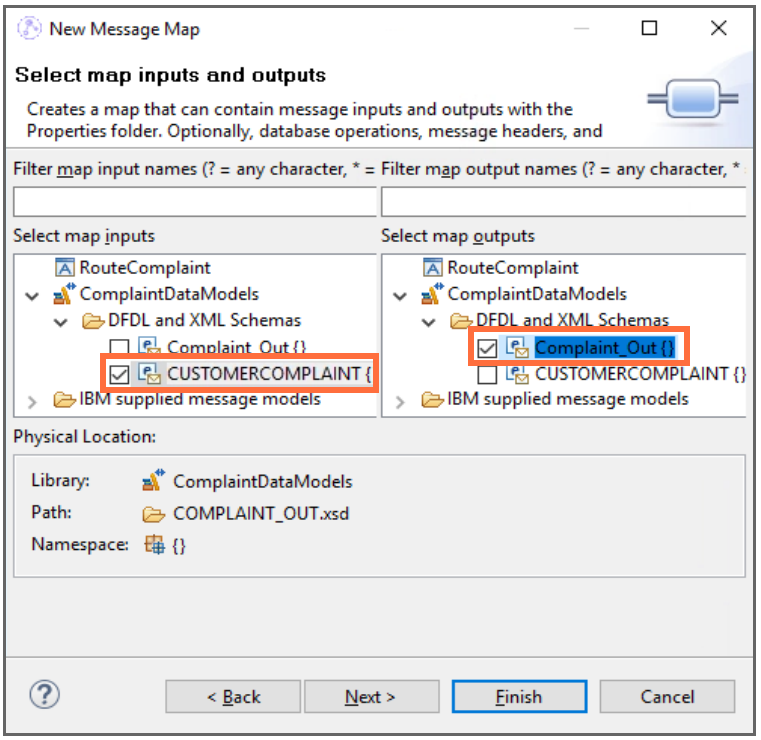
__ b. Accept the default values on this page for **Container** and **Map name** and click **Next**.



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.

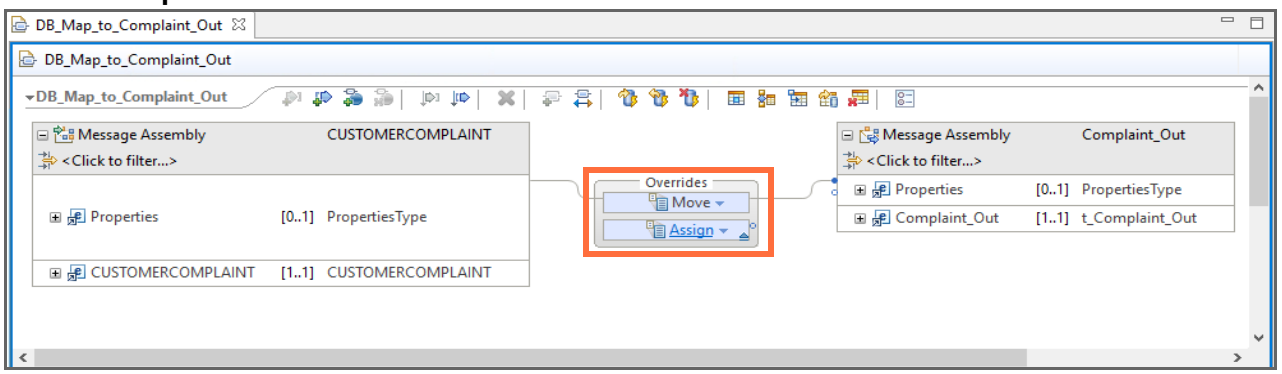
__ 2. The next step is to define how the source elements map to the target elements.

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.

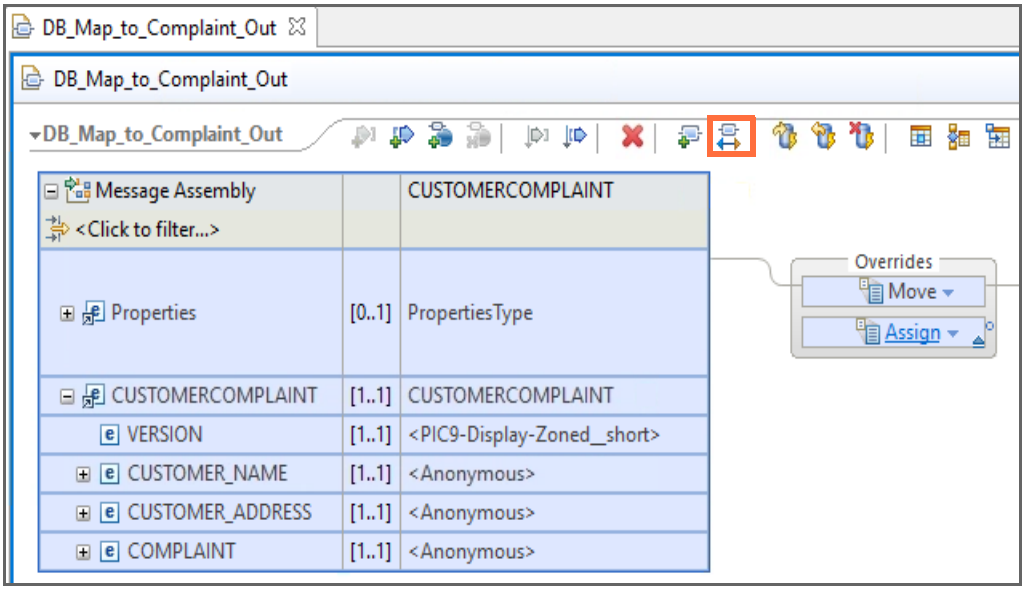
The **Assign** transform sets the **MessageSet** on the output to the library name **ComplaintDataModels**.



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.

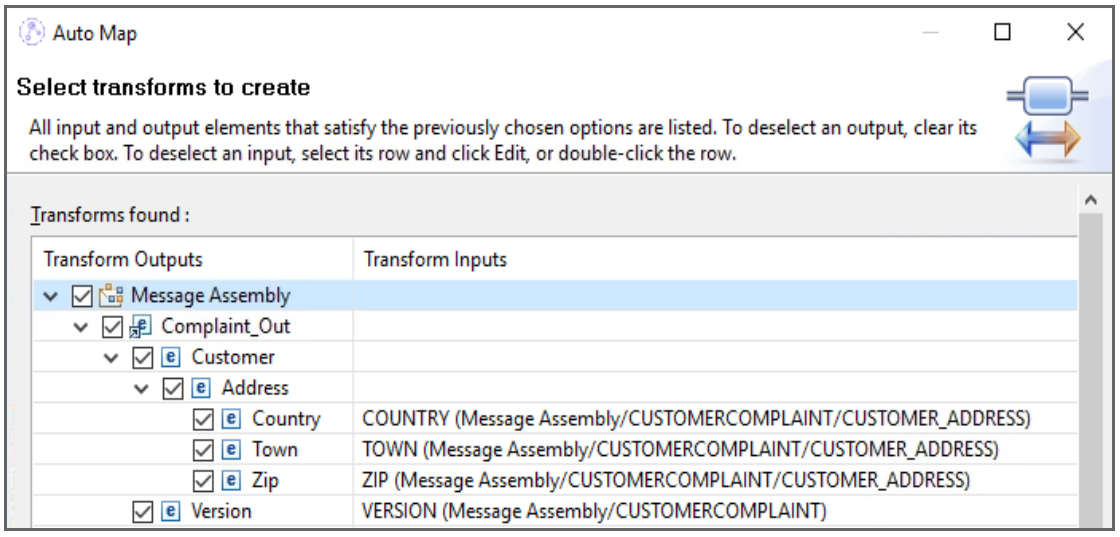
__ b. Click the **Auto map input to output** icon in the Mapping editor toolbar.



The AutoMap properties window opens.

__ c. Without changing any of the parameters, click **Next**. A preview of the mapped fields is displayed.

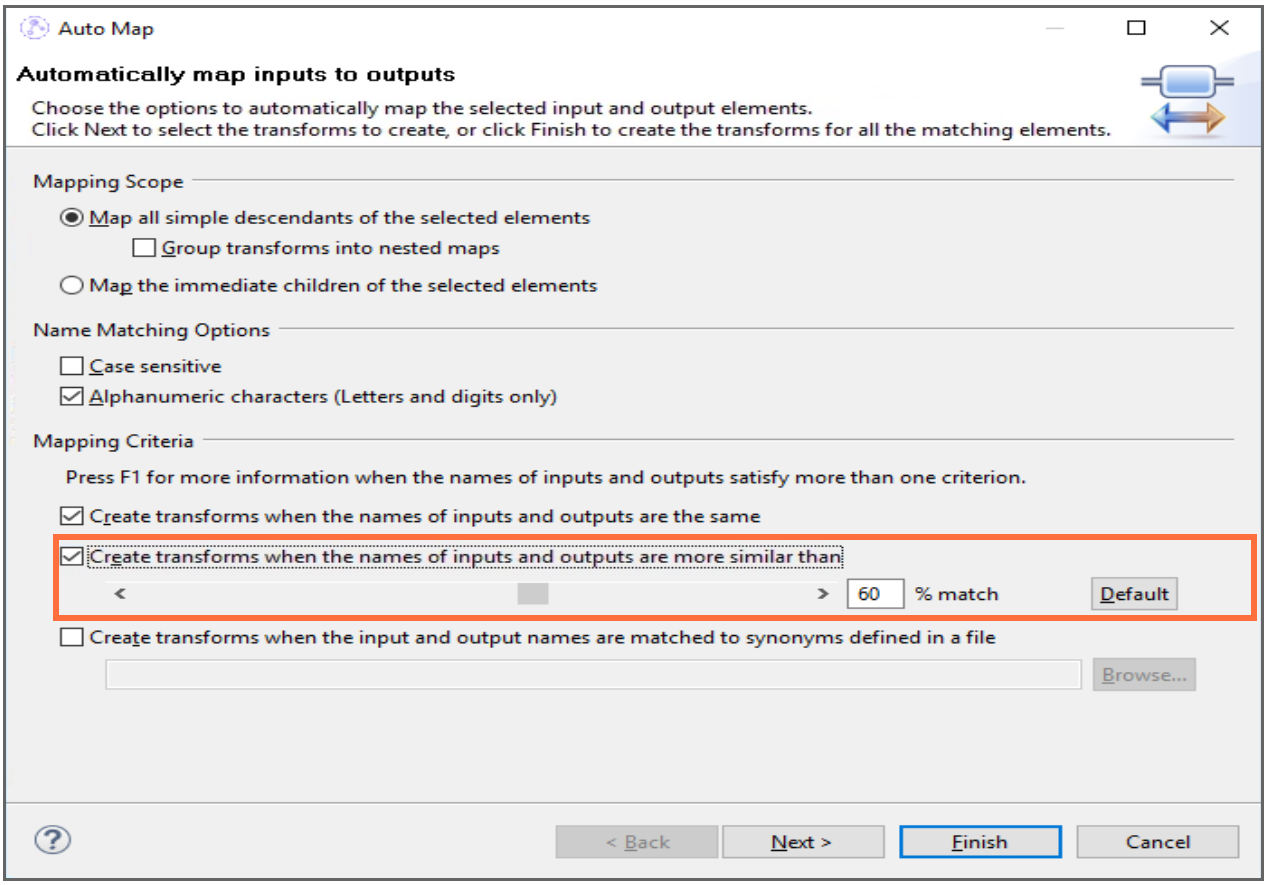
__ d. You see that only the fields that exactly matched are mapped. You can control this “strictness” in the matching so that names do not need to match exactly.



__ e. Click **Back**.

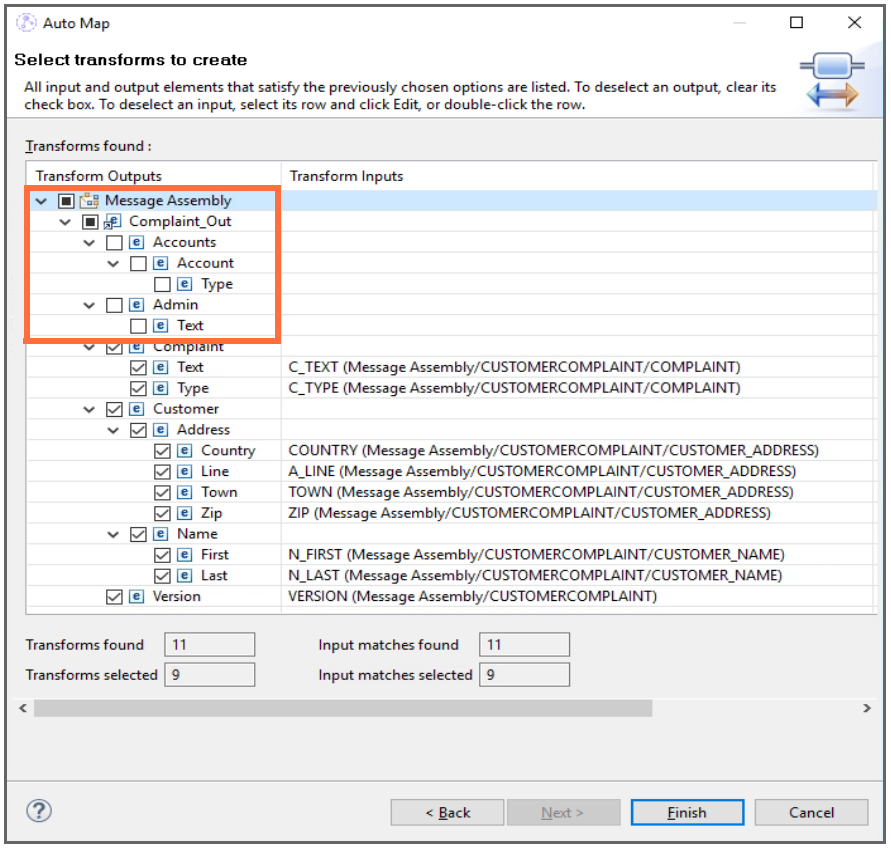
__ f. Under **Mapping Criteria**, click the **Create transforms when the names of input and outputs are more similar than** option.

__ g. Leave the default of **60%** selected.



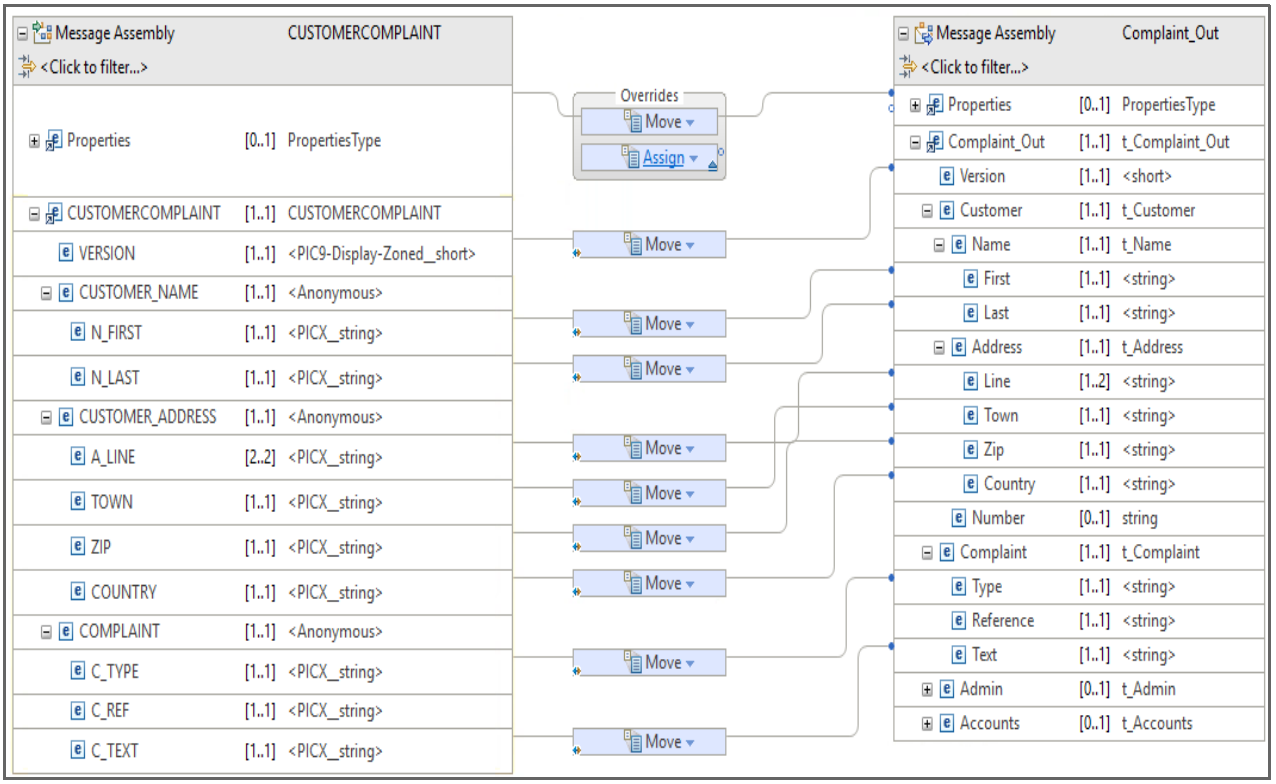
__ h. Click **Next**. Now the preview shows that more matches were found, based on “looser” matching of the source and target fields.

__ i. Clear the check boxes for the assignments of `Accounts/Account/Type` and `Admin/Text` Data.



__ j. Click **Finish**. The automatic mapping runs.

__ k. Review the results of the auto map.

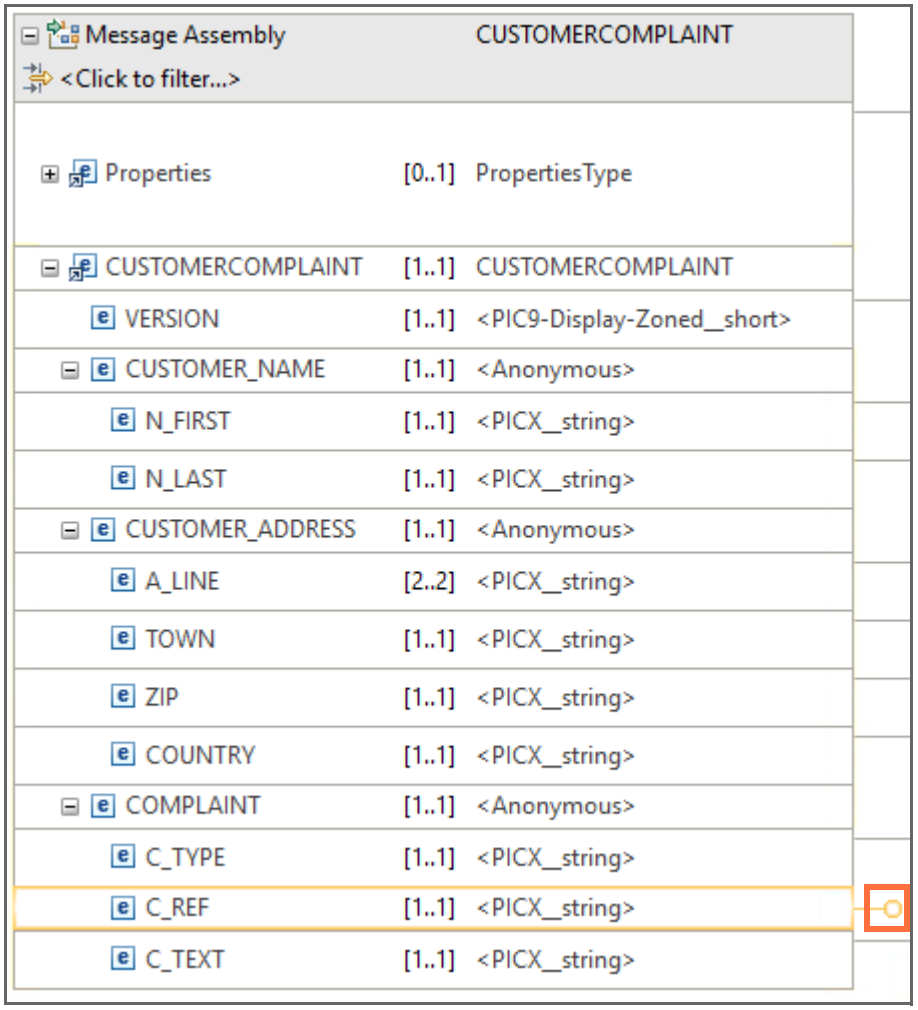


Hint

Double-click the **DB_Map_to_Complaint_out** tab to expand the Graphical Data Mapping editor view. Double-click the tab again to return the view to the default size.

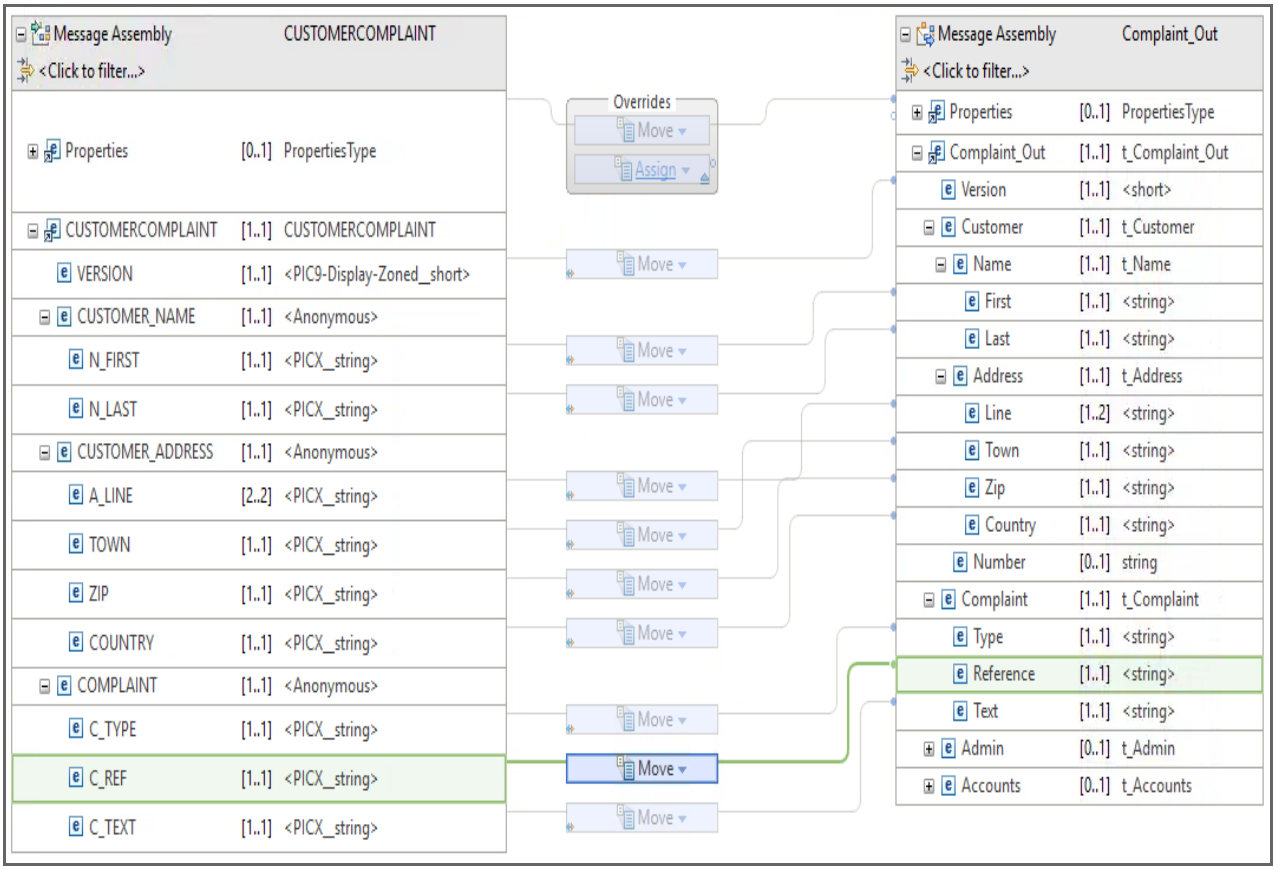
__ 3. The `COMPLAINT/C_REF` field in the input message did not get mapped to the `Complaint/Reference` field on the output message because the names were too dissimilar.

__ a. Manually map the fields by hovering the mouse cursor over `COMPLAINT/C_REF` in the source message. A yellow 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.



__ d. Save the message map (Ctrl+S).

Part 6: 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. Define which rows to select.

__ a. In the Mapping editor toolbar, click the **Select rows from a database** icon.

The **New Database Select** window opens.

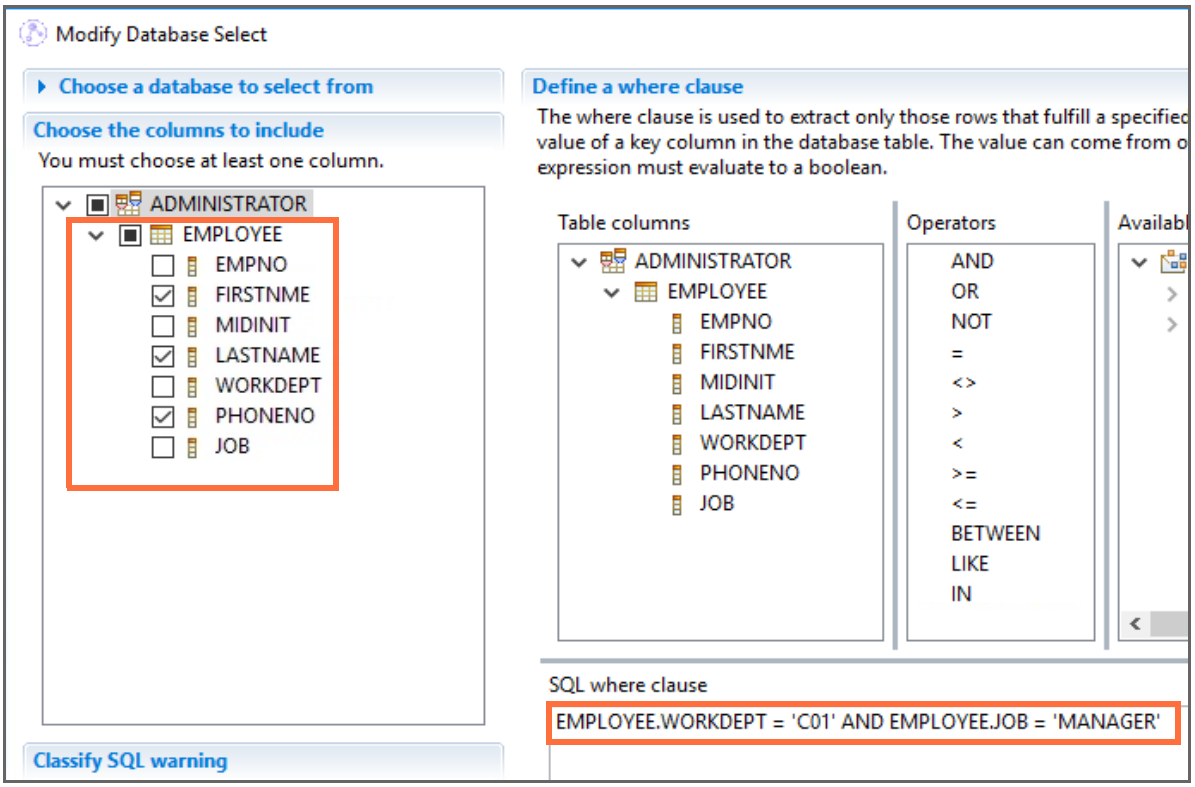
__ b. Under **Choose a database to select from**, select **SAMPLE**.

__ c. Under **Choose the columns to include**, expand **ADMINISTRATOR > EMPLOYEE**, and then click **FIRSTNME**, **LASTNAME**, and **PHONENO**.

__ d. In the **SQL where clause** field, under the **Define a where clause** section, enter:

```
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".



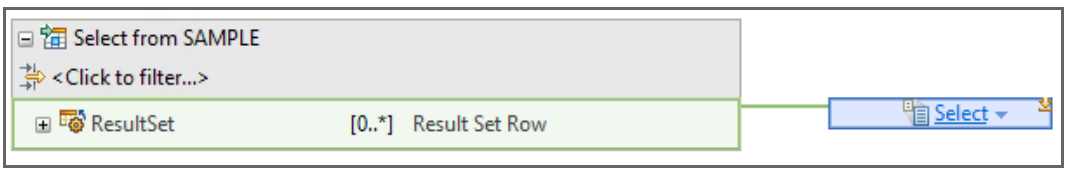
Important

Enter the text manually as displayed. Do not cut and paste.

__ e. Click **OK**.

The Mapping editor pane returns.

__ f. Scroll down under the source message. You should see that a **Select from SAMPLE** input is added to the map source.

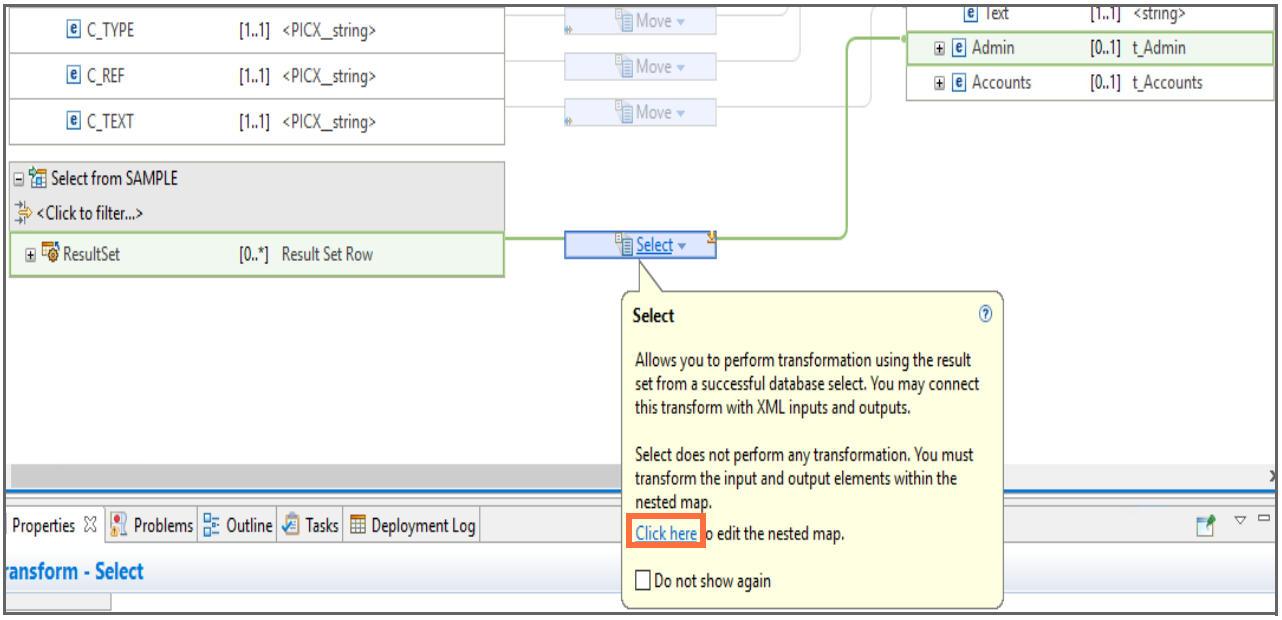


__ 2. Map the fields that are returned from the database SELECT statement.

__ a. 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.

__ b. Click the **Click here** link to open the editor for the nested map.



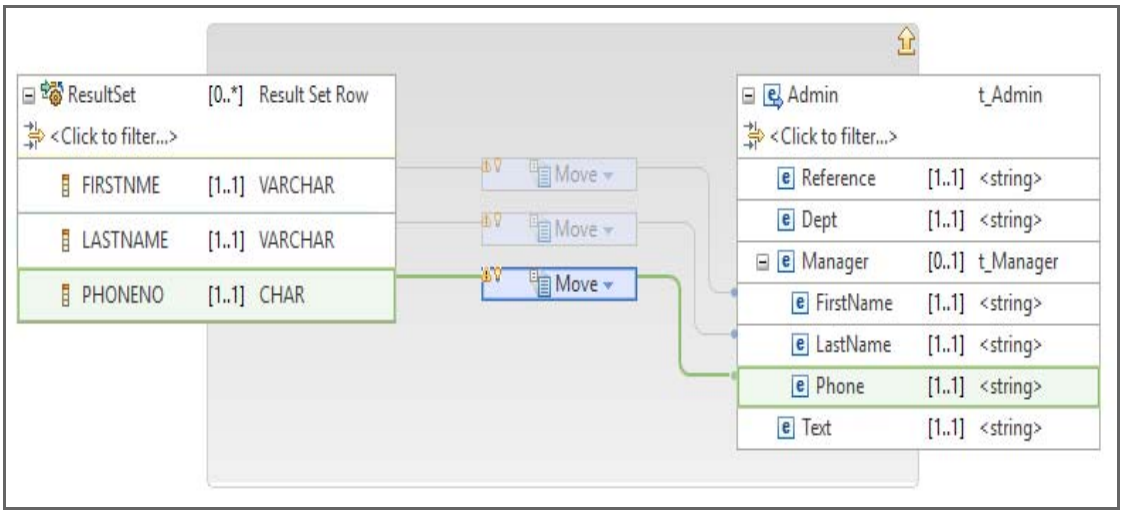
__ 3. Connect the **ResultSet** fields in the input to the **Manager** fields in the output

__ a. Expand **Manager** under **Admin**

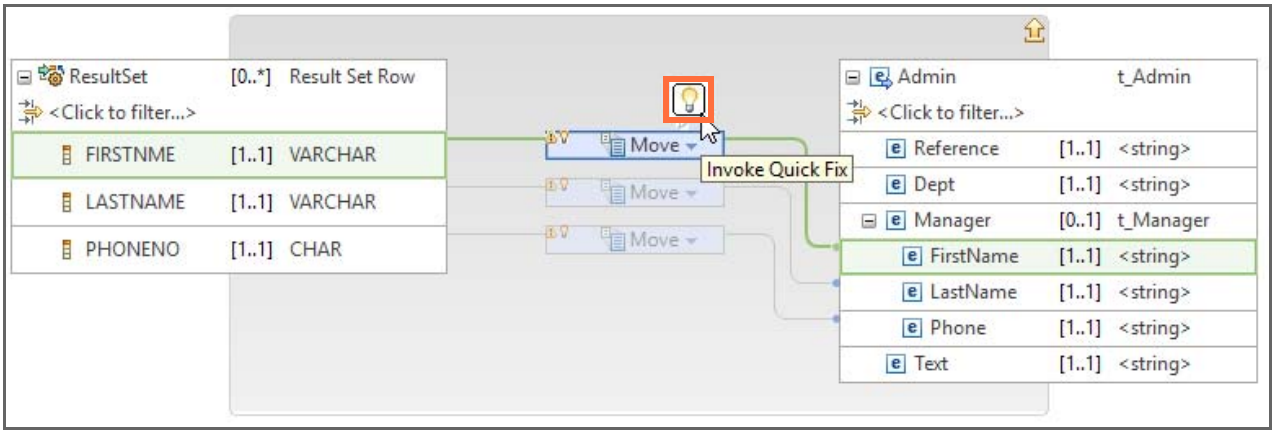
__ b. Connect **FIRSTNME** to **FirstName**.

__ c. Connect **LASTNAME** to **LastName**.

__ d. Connect **PHONENO** to **Phone**.



__ e. There are cardinality warnings displayed in the editor. To fix them, highlight one of the **Move** actions and select **Invoke Quick Fix**. Click the link **Set cardinality to first index** for the quick fix.



All three errors are corrected.

__ f. Save the map (Ctrl+S), and close the mapping editor.

The error for the **Map to Complaint_Out** node is now gone.

Information

The **Problems** view might contain ambiguous database table reference warnings and some *Unresolvable message field reference* warning messages. These references are resolved at run time and can be ignored.

Part 7: Configure database connectivity

In the first part of this section, you verify the ODBC DSN for MSGSTORE. To configure the JDBC connection to allow the ESQL code to connect to the database, you create and deploy a Policy project and use the `mqsisetdbparms` command.

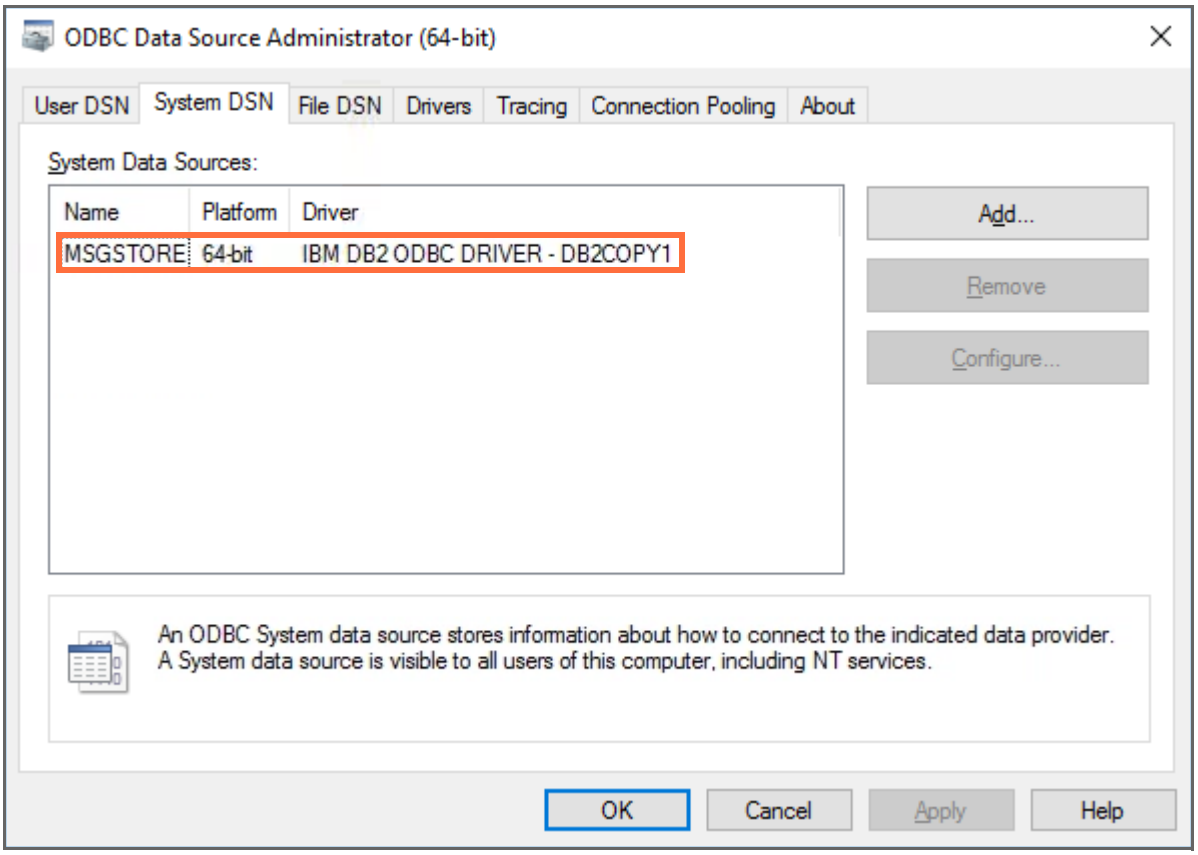
__ 1. Verify the ODBC DSN for MSGSTORE.

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

__ b. Double-click **ODBC Data Source Administrator (64-bit)**

__ c. Click the System DSN tab.

__ d. Verify that the ODBC Drivers are present.

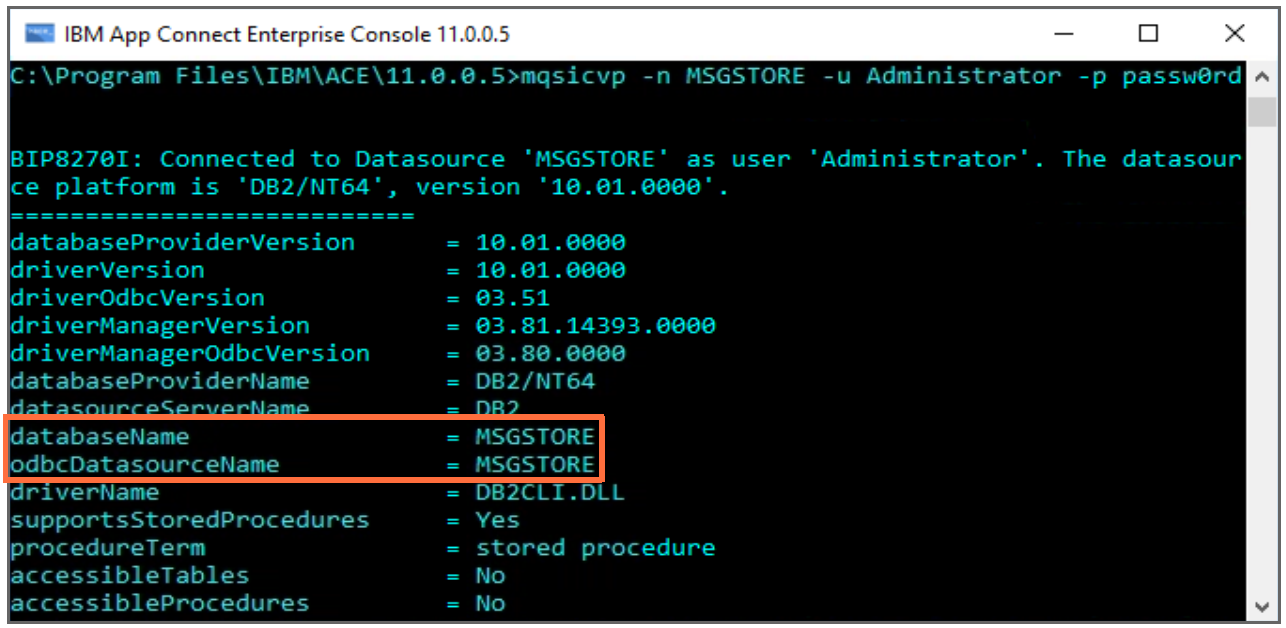


__ e. Close the window.

__ f. Open the App Connect Enterprise Command Console

You can also use the `mqsicvp` command to run verification tests on an integration node, or to verify ODBC connections.

__ g. Enter the following command: `mqsicvp -n MSGSTORE -u Administrator -p passw0rd`



__ 2. Deploy a policy project.

To access the SAMPLE database, you need to deploy a policy describing the JDBC connection.

__ a. Return to the App Connect Enterprise Toolkit.

__ b. Create a new policy project by navigating to: **File > New > Policy**.

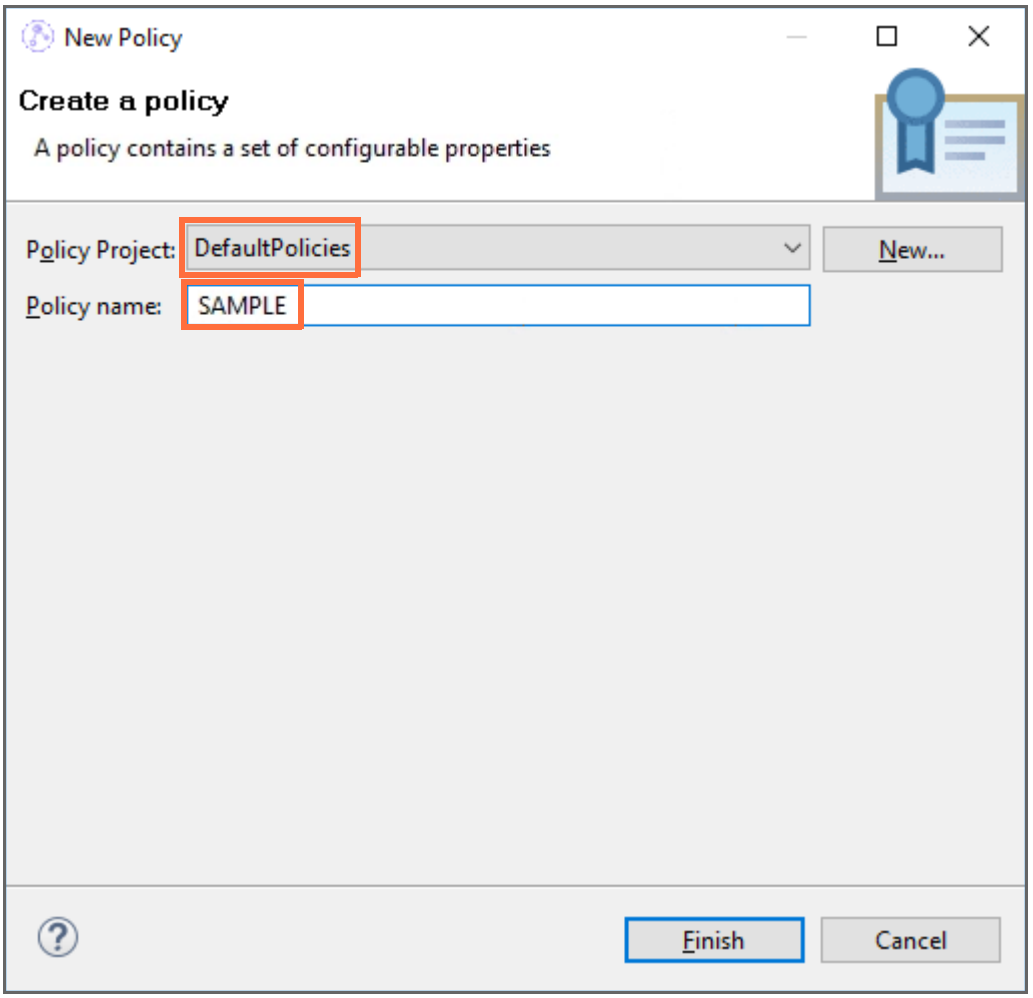
Important

Make sure you select **File > New > Policy** and not **File > New > Policy Project**.

__ c. Click **New** to create a new policy project.

__ d. Enter `DefaultPolicies` for the name and click **Finish**

__ e. Enter `SAMPLE` for the name of the Policy when returned to the New Policy window.



__ f. Click **Finish**.

The policy is opened in the policy editor.

__ g. For Type, select **JDBC Providers**.

The template is pre-filled with default values.

__ h. Enter SAMPLE for **Name of the database**.

__ i. Enter `localhost` for **Database server name**.

__ j. Enter `mydbuser` for **Security identity (DSN)**.

__ k. Delete the portion of the Connection URL dealing with authentication. Verify that the connect URL format is: `jdbc:db2//[serverName]:[portNumber];`

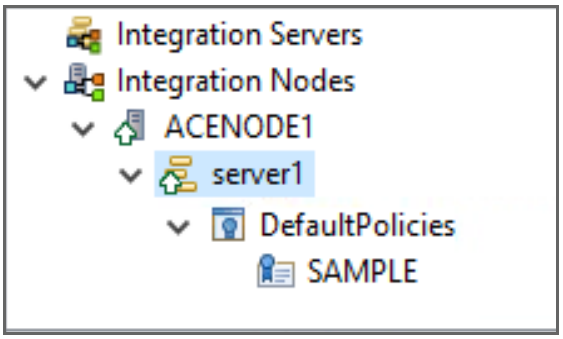
Property	Value
Name of the database*	SAMPLE
Type of the database	DB2 Universal Database
Version of the database	9.1
JDBC driver class name*	com.ibm.db2.jcc.DB2Driver
JDBC type 4 data source class name*	com.ibm.db2.jcc.DB2XADataSource
Connection URL format*	jdbc:db2://[serverName]:[portNumber];
Connection URL format attribute 1	
Connection URL format attribute 2	
Connection URL format attribute 3	
Connection URL format attribute 4	
Connection URL format attribute 5	
Database server name*	localhost
Database server port number*	50000
Type 4 driver class JARs URL	C:\\Program Files\\IBM\\SQLLIB\\java
Name of the database schema	useProvidedSchemaNames
Data source description	
Maximum size of connection pool	0
Security identity (DSN)	mydbuser
Environment parameters	
Supports XA coordinated transactions	true
Use JAR files that have been deployed in a .bar file	false

__ l. Save and close the policy.

__ m. Deploy the policy by dragging the **DefaultPolicies** policy project to **server1**.

Alternatively, you could add the policy project as a dependency in the BAR file editor. This method would ensure it exists prior to deployment.

__ n. Verify that the policy project is deployed to **server1**.



__ 3. Set the database parameters for the **jdbc** connection

__ a. Return to the App Connect Enterprise console.

__ b. Enter the following command:

```
mqsisetdbparms ACENODE1 -n jdbc::mydbuser -u Administrator -p passw0rd
```

__ c. Restart the server by entering the following commands:

```
mqsistop ACENODE1
```

```
mqsistart ACENODE1
```

A restart is necessary for the `mqsisetdbparms` command to take effect.

Part 8: Test 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 App Connect Enterprise 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.

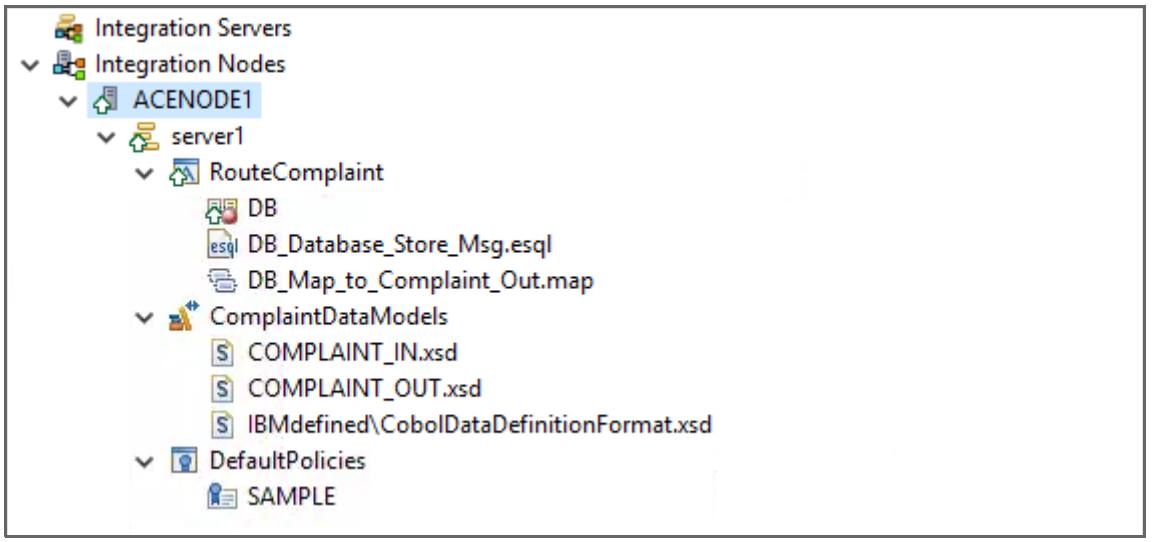
__ 1. Start the IBM App Connect Enterprise Toolkit Flow exerciser.

Because the message flow requires the shared library, you need to deploy that first before starting the Flow exerciser.

If ACENODE1 is displayed as stopped, refresh the view.

__ a. Deploy the **ComplaintDataModels** shared library by dragging it to the **server1** integration server.

__ b. In the Message Flow editor, click the **Start Flow exerciser** icon to build and deploy the application to the **server1** integration server on the ACENODE1 integration node.



__ 2. Send a message to the flow.

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

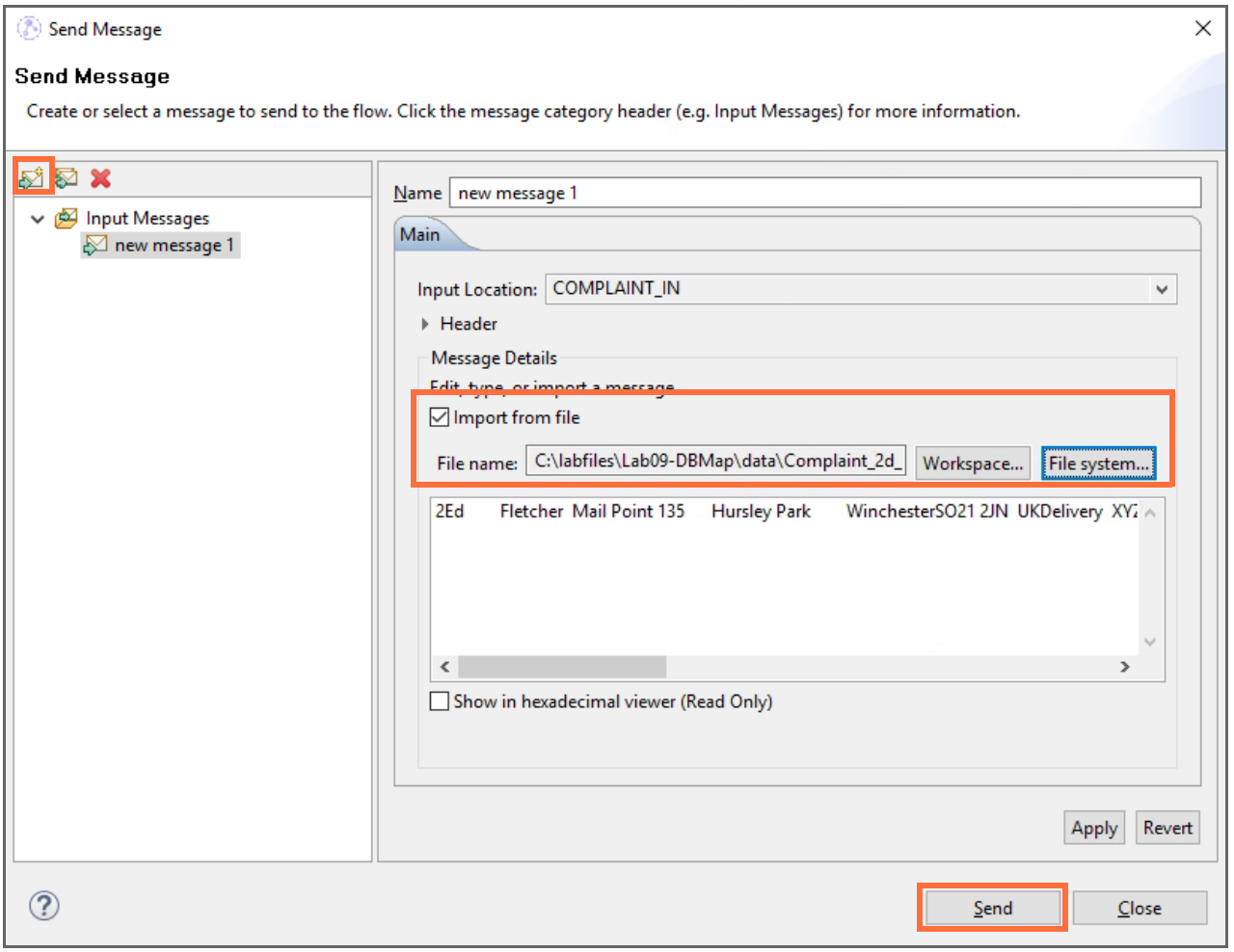
__ b. Click the **New Message** icon.

__ c. Click **Import from file**.

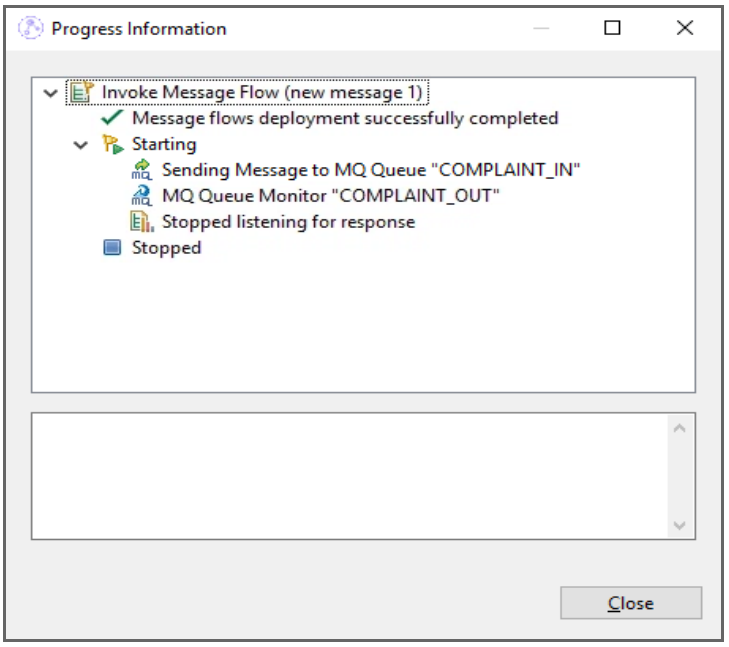
__ d. Click **File system**.

__ e. Go to the `C:\labfiles\Lab09-DBMap\data\` directory, click `Complaint_2d_cwf.txt`, and then click **Open**.

__ f. Click **Send**.



__ g. The **Progress Information** windows shows the status of the message flow. Verify that the message flow ran successfully.



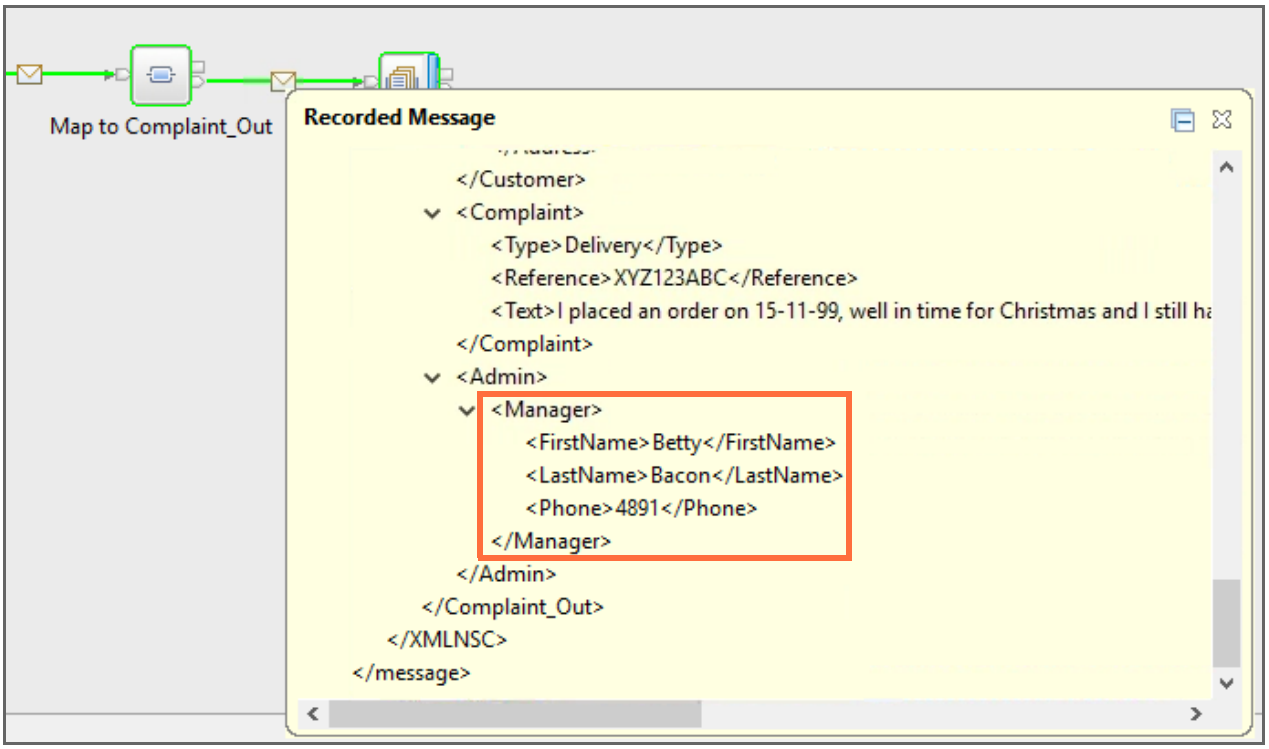
__ h. Click **Close** on the **Progress Information** window when you see the **Stopped** event.

__ 3. Review the results

The Flow exerciser path should show that the message was sent to the COMPLAINT_OUT node.

__ a. Click the message icon between the **Map to Complaint_Out** node and the **COMPLAINT_OUT** node.

__ b. Verify that the output message is correct and that it includes the **Manager** information from the EMPLOYEE table in the SAMPLE database.



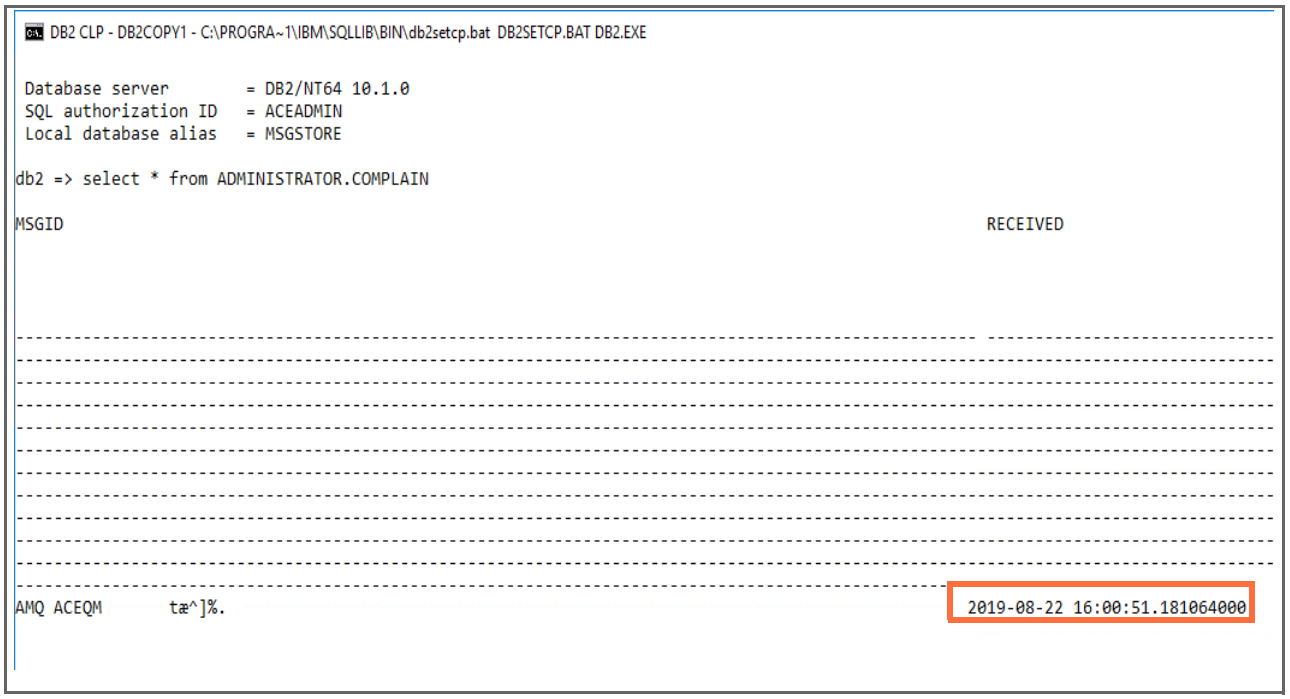
__ 4. The ESQL code in the Database node stored the message in a database table. Use the DB2 command line processor to verify that an entry is created in the COMPLAIN table in the MSGSTORE database.

__ a. From the Windows **Start** menu, click **All Programs > IBM DB2 > 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 ACEQM followed by a bit stream.
- 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 bit stream.
- There might be extra rows in the database from prior testing. Ensure that you view the last entry in the list.

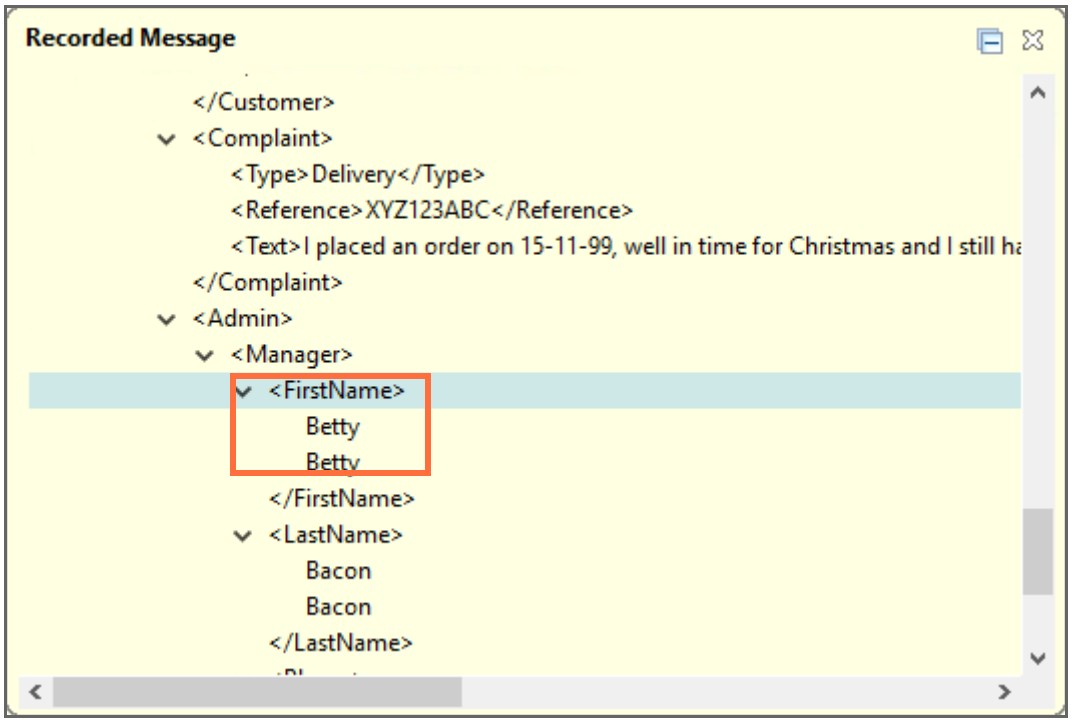
__ d. End the connection to the MSGSTORE database. Type: `terminate`

__ e. Close the DB2 command line processor window.

__ f. If necessary, correct the message flow, and retest. Use the problem determination tools that you learned in this course to determine the problem.

Troubleshooting

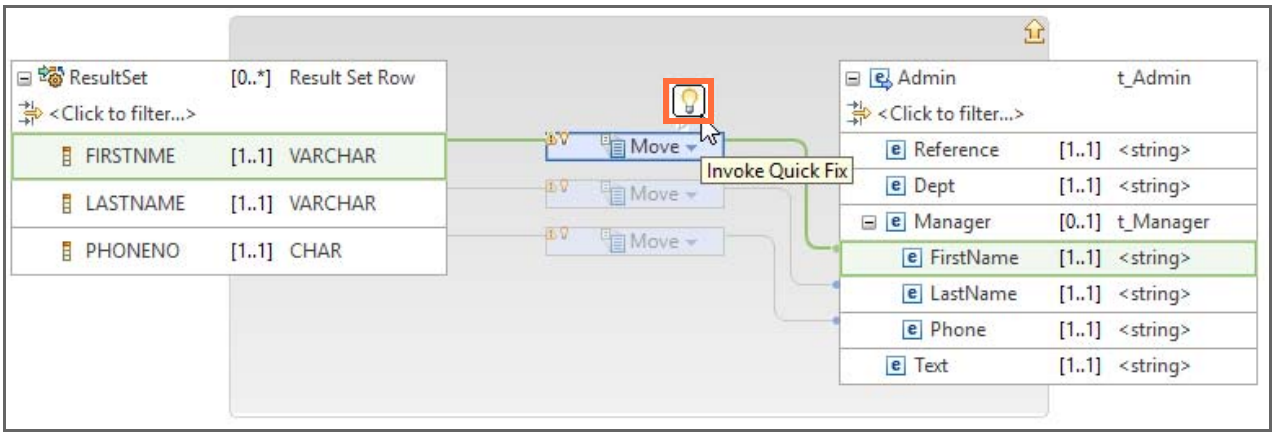
If your message contains repeating elements.



__ a. Go to the Problems view, double-click one of the messages pertaining to cardinality.

You are taken to the mapping editor.

__ b. Highlight one of the Move actions and select **Invoke Quick Fix**. Click the link for the quick fix.



__ c. Save your work.

__ d. All cardinality-related warnings are gone.

__ e. Retest the flow using the Flow exerciser.

Part 9: 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 Explorer view**, right-click the **server1** integration server and click **Delete > Delete all resources**.

__ 4. In IBM MQ Explorer, delete the queues.

End of exercise

Exercise review and wrap-up

In the first 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 defined the connections to the SAMPLE and MSGSTORE databases in the IBM App Connect Enterprise 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 configured the database connectivity for the MSGSTORE and SAMPLE databases. You then created and deployed a policy project defining the JDBC connection information for the mapping node.

In the final part of the exercise, you tested the message flow and verified the contents of the output message by using the IBM App Connect Enterprise 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
- Define database connectivity
- 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

Estimated time

01:30

Overview

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. As part of the exercise, you choose to create one or the other. If time permits, you can choose to implement both.

Objectives

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

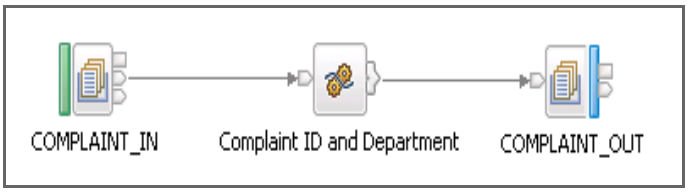
Introduction

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>
```

As part of the 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 App Connect Enterprise V11 Toolkit and IBM MQ V9
- The files that are required for this exercise are in the `C:\labfiles\Lab10-Compute` directory

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab10`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, enter `C:\Workspace\Lab10`

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

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

__ 2. Verify that the node and server are running.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 3. This exercise requires an IBM MQ queue manager.

If you completed Exercise 3, you created a queue manager that is named ACEQM. You can use that queue manager in this exercise. Proceed to the next step.

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

__ a. Start IBM MQ Explorer.

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

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

__ 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.

__ 4. Create the IBM MQ queues required for this exercise by running an IBM MQ script file.

In a command window, type:

```
runmqsc ACEQM < C:\labfiles\Lab10-Compute\CreateQueues.mqsc
```

__ 5. Use IBM MQ Explorer to verify that the queues were created.

__ a. In the IBM MQ Explorer Navigator view, expand the **ACEQM** 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_OUT, and DLQ.

Note

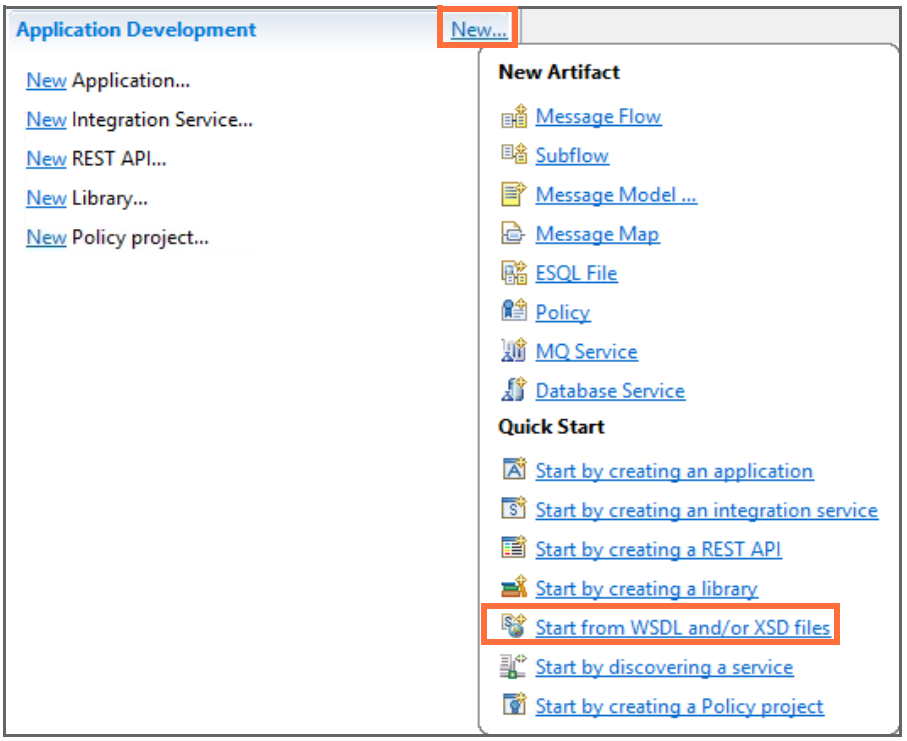
You might have other queues for this queue manager from a previous exercise. You do not need to delete the unused queues.

Part 2: 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.

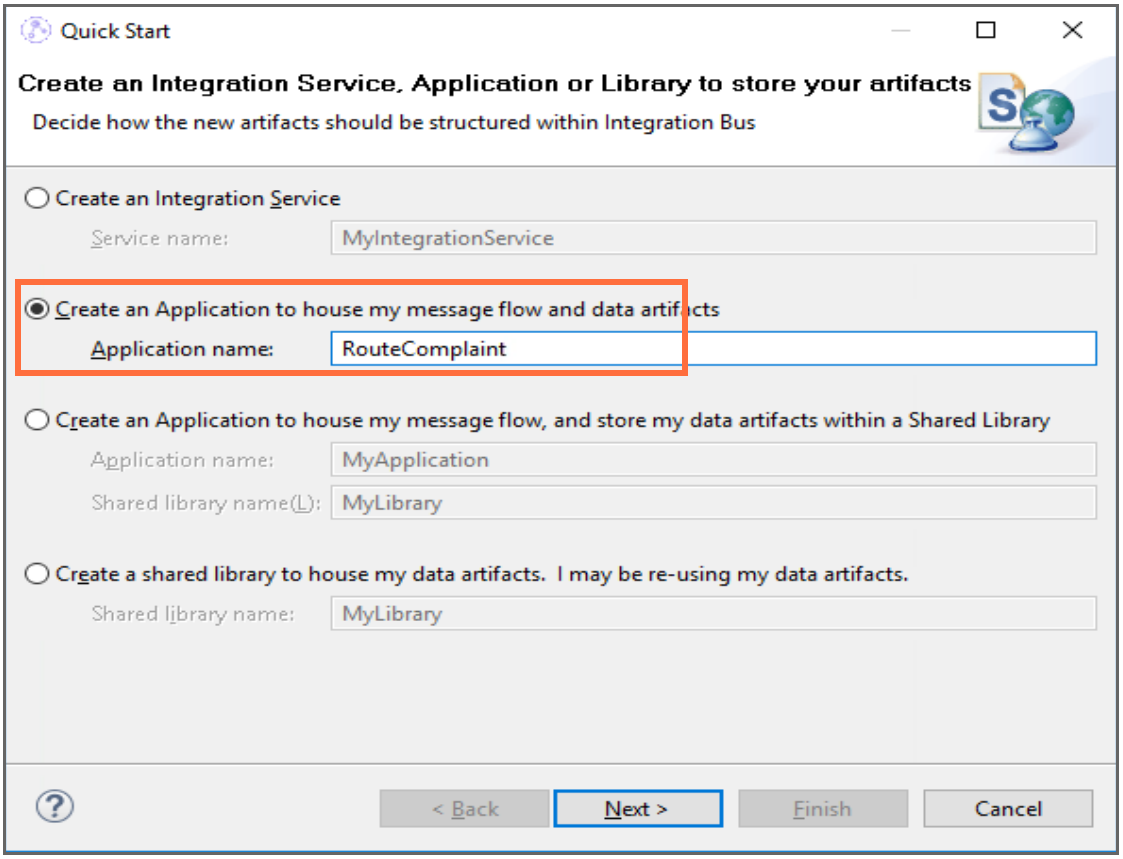
__ a. In the IBM App Connect Enterprise Toolkit Application Development view, click **New > Start from WSDL and/or XSD files**.



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`**



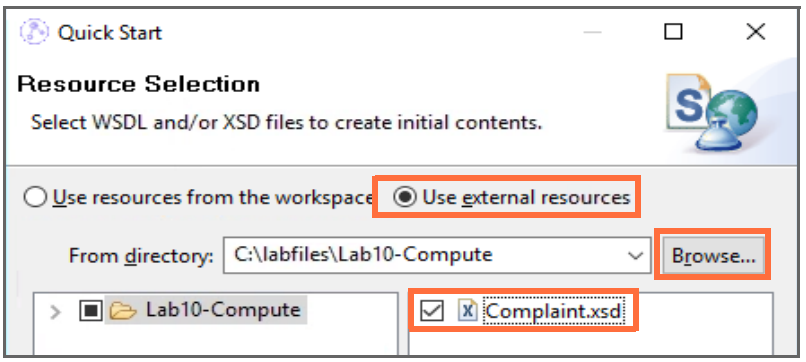
__ d. Click **Next**. The **Resource Selection** window opens.

__ e. Click **Use external resources**.

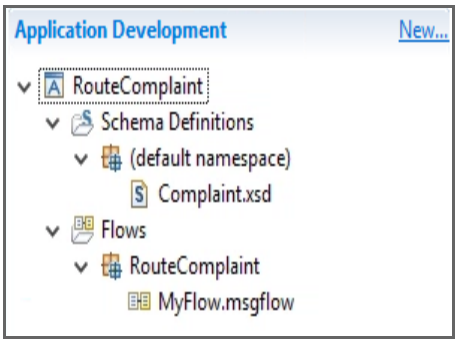
__ f. Click **Browse** to the right of **From directory**.

__ g. Browse to the **`C:\labfiles\Lab10-Compute`** directory and then click **OK**.

__ h. Select the **Complaint.xsd** check box.



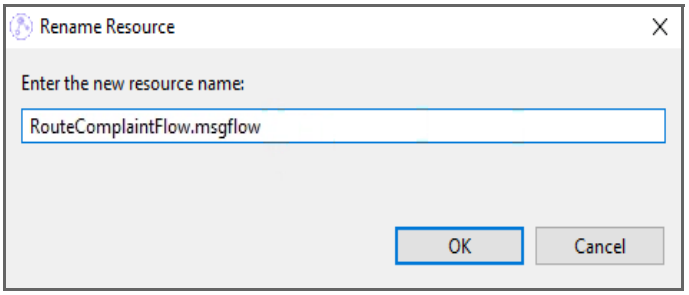
__ i. Click **Finish**. The schema definition is imported and the application is created in the **Application Development** view.



__ 2. The **RouteComplaint** application contains an empty 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. Add nodes to the Message Flow editor canvas.

__ a. Double-click the **RouteComplaintFlow.msgflow** to open it in the Message Flow editor

__ b. From the **IBM MQ** drawer, add one MQInput node

__ c. 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

__ d. From the **IBM 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.

__ 4. Rename the nodes to more descriptive names.

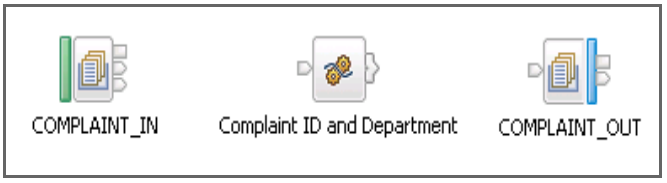
To rename a node, click it, and then switch to the **Properties** tab. Change the **Node name** field on the **Description** tab.

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`

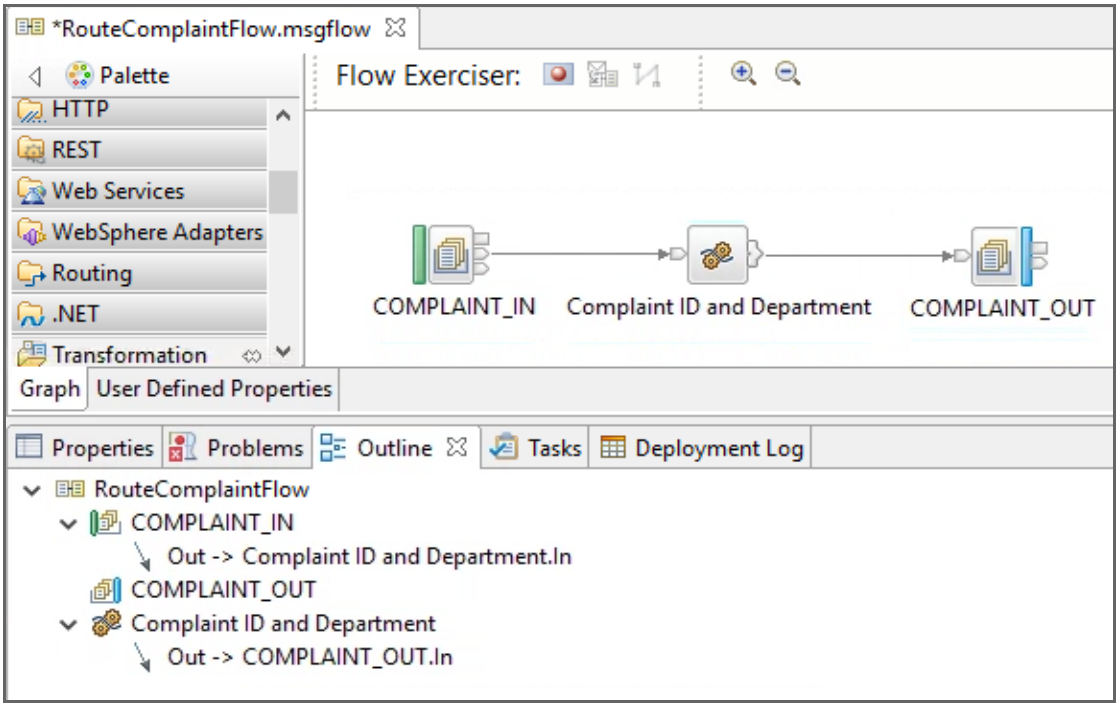


__ 5. Connect the nodes.

__ 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.

__ c. Verify the terminal connections in the **Outline** view.



__ 6. 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: `ACEQM`

__ c. On the **Input Message Parsing** tab, set the **Message Domain** property to: `XMLNSC`

__ 7. 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: `ACEQM`

__ c. 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 3: 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.

You only need to implement one of the options. 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.

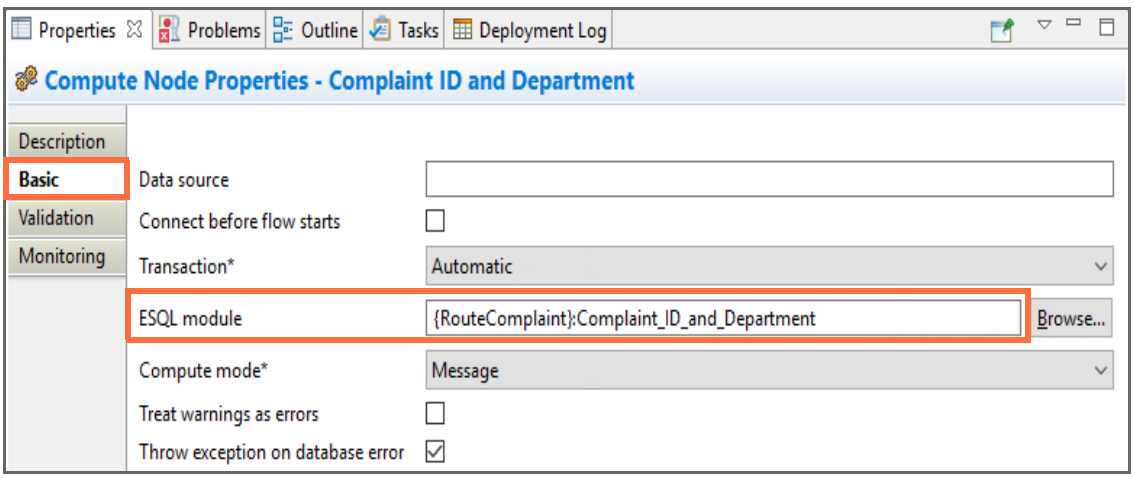
Option 1: Write a transformation by using a Compute node and ESQL

If you are using a Compute node in your message flow, the next step is to create the ESQL to transform the message.

__ 1. Write a transformation by using a Compute node and ESQL

__ a. 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`



__ b. 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**.

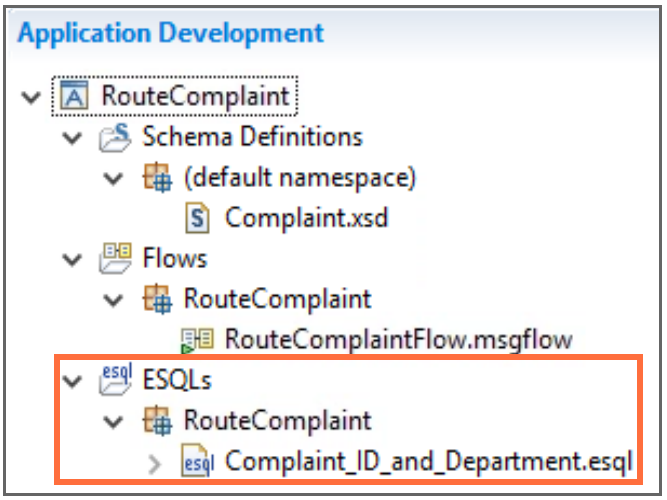
The second statement, `CREATE COMPUTE MODULE Complaint_ID_and_Department` is the Compute module name and must match the name that you entered in the ESQL module property in Step 1.

__ c. 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;
```

__ d. Save the ESQL file. The source for the ESQL module is saved with the application under the **ESQLs** folder.



__ e. Close the ESQL editor.

__ f. Save the message flow.

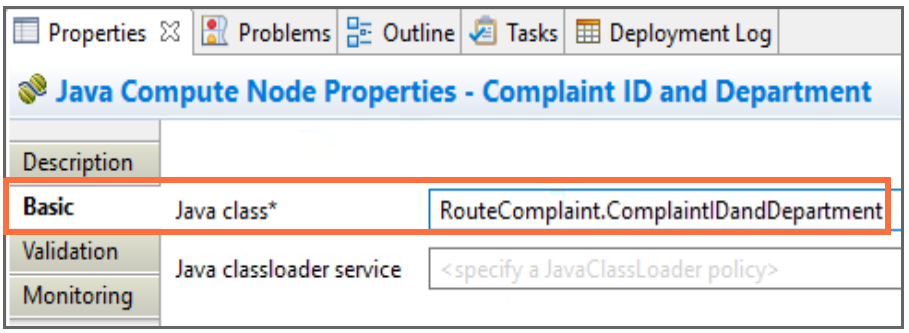
__ g. Verify that the **Problems** view is empty.

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.

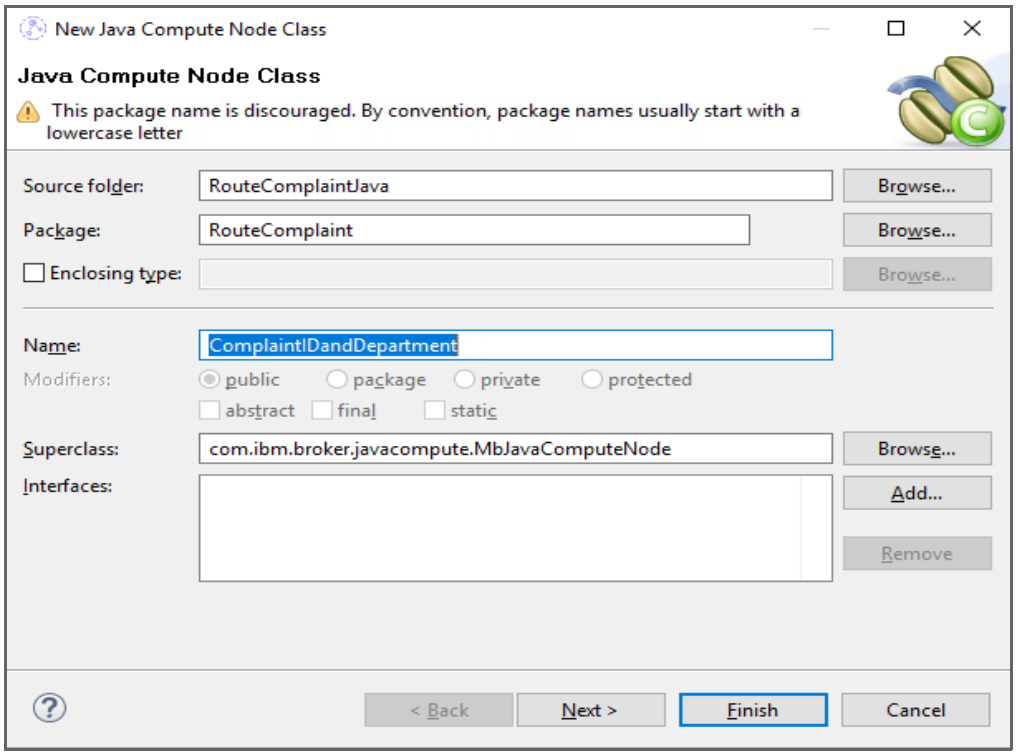
__ 1. Configure the JavaCompute node to use the `RouteComplaint.ComplaintIDandDepartment` Java class.

__ a. In the JavaCompute node that is named Complaint ID and Department, set the **Java class** property to `RouteComplaint.ComplaintIDandDepartment` on the **Basic** tab.



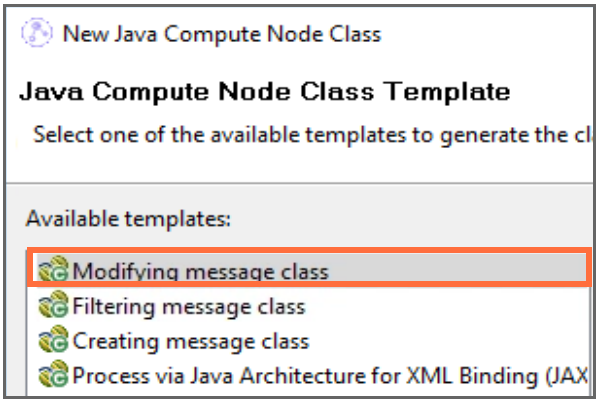
__ b. In the Message Flow editor, double-click the Complaint ID and Department node. The **New Java Compute Node Class** wizard is displayed.

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).



__ c. Click **Next**. The Java Compute Node Class Template window opens.

__ d. In this exercise, you modify the message in this JavaCompute node, select **Modifying message class**, and then click **Finish**.

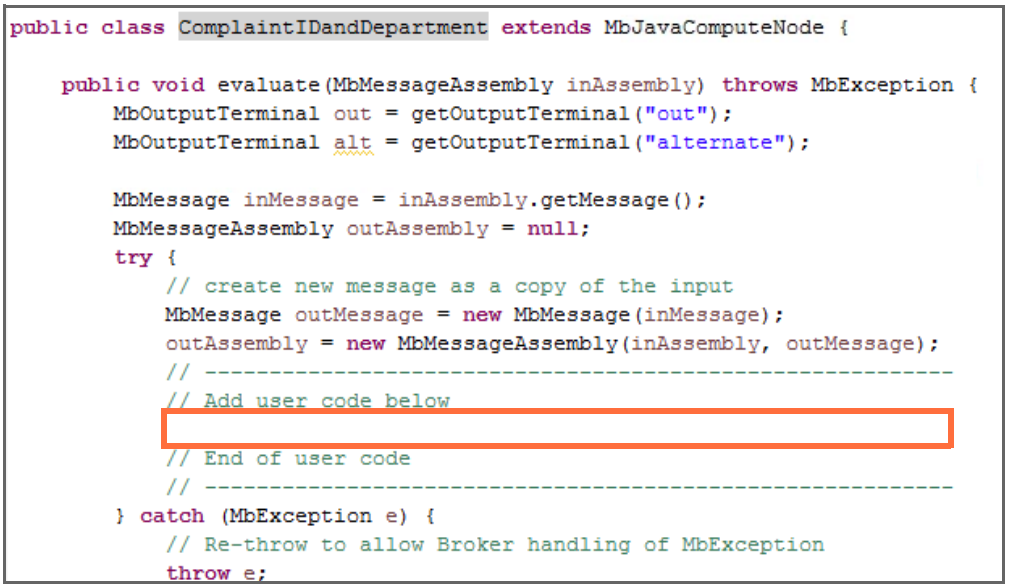


The Java editor opens with the code template for modifying a message.

__ 2. Enter the Java code for the transformation.

__ a. Copy the Java code from the `Java_Cut&Paste.txt` file in the `C:\labfiles\Lab10-Compute` directory.

__ b. Paste the Java code to the Java between the `// Add user code below` and `// End of user code` lines.

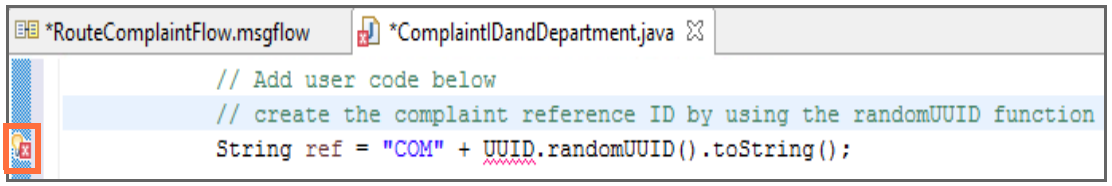


The code that you insert into the file is shown here:

```
// create the complaint reference ID by using the randomUUID function
   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
```

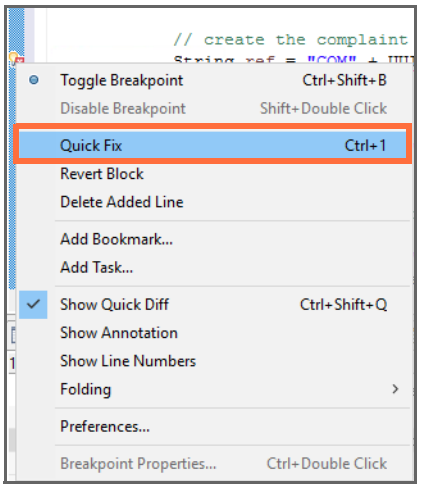
__ 3. Correct the error in the Java code.

__ a. 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*.

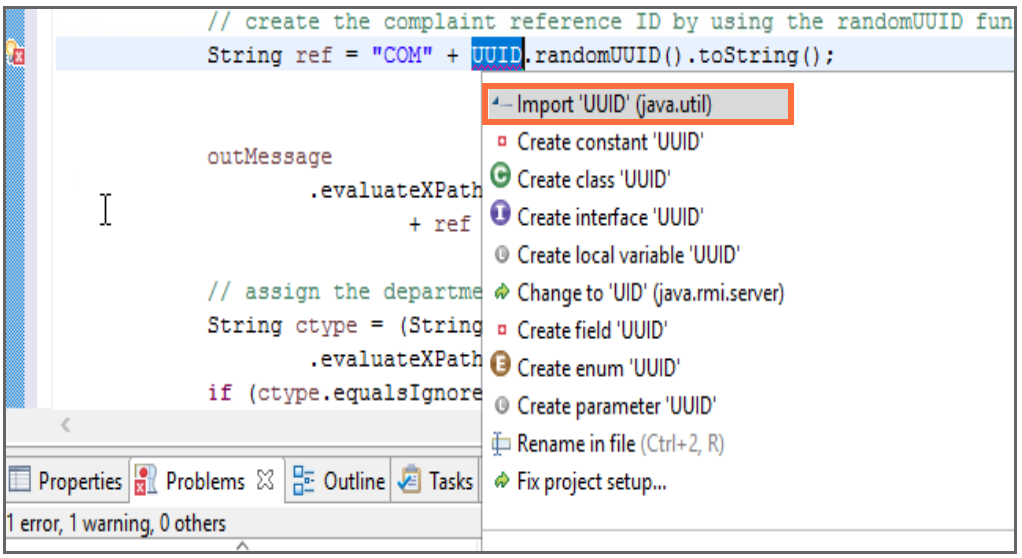


__ b. To fix this error, click the red error decorator in the margin. A quick fix menu opens.

__ c. Click **Quick Fix**.



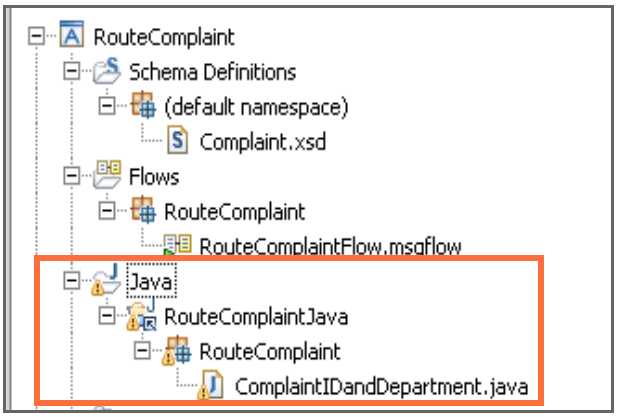
__ d. Double-click **Import 'UUID' (java.util)**.



The error decorator disappears.

__ 4. Save changes to the JavaCompute node.

__ a. Save the changes in the Java editor. The source for the Java file is saved with the application under the **Java** folder.

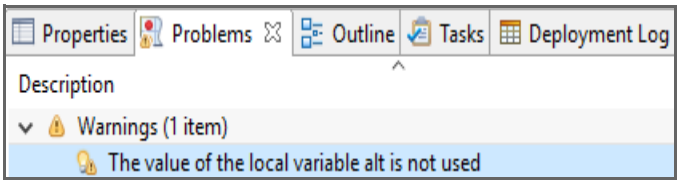


__ b. Close the Java editor.

__ c. Save the message flow.

__ d. Verify that there are no errors in the **Problems** view.

__ e. You can ignore the warning that is related to the `alt` variable.



Part 4: Test the message flow

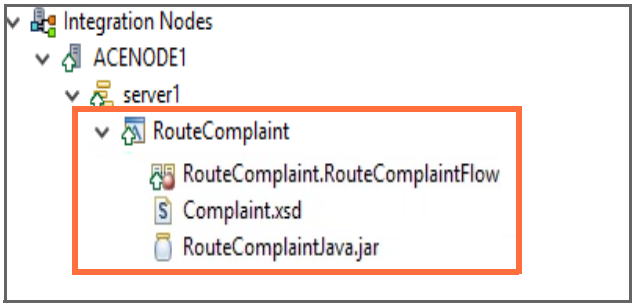
In this part of the exercise, you use the IBM App Connect Enterprise Toolkit Flow exerciser to test the message flow.

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 App Connect Enterprise Toolkit Message Flow editor, start the Flow exerciser to create the BAR file, deploy the application to **server1** integration server on the ACENODE1 integration node, and start recording.

__ 2. In the **Integration Explorer** view, verify that the message flow application deployed successfully.

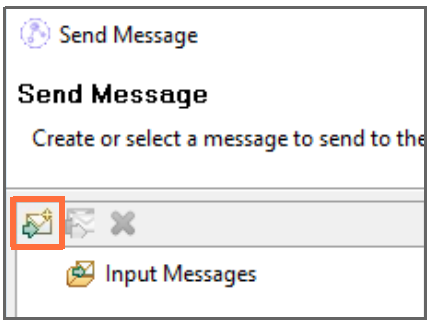


The screen capture displays the solution for the JavaCompute node.

__ 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.

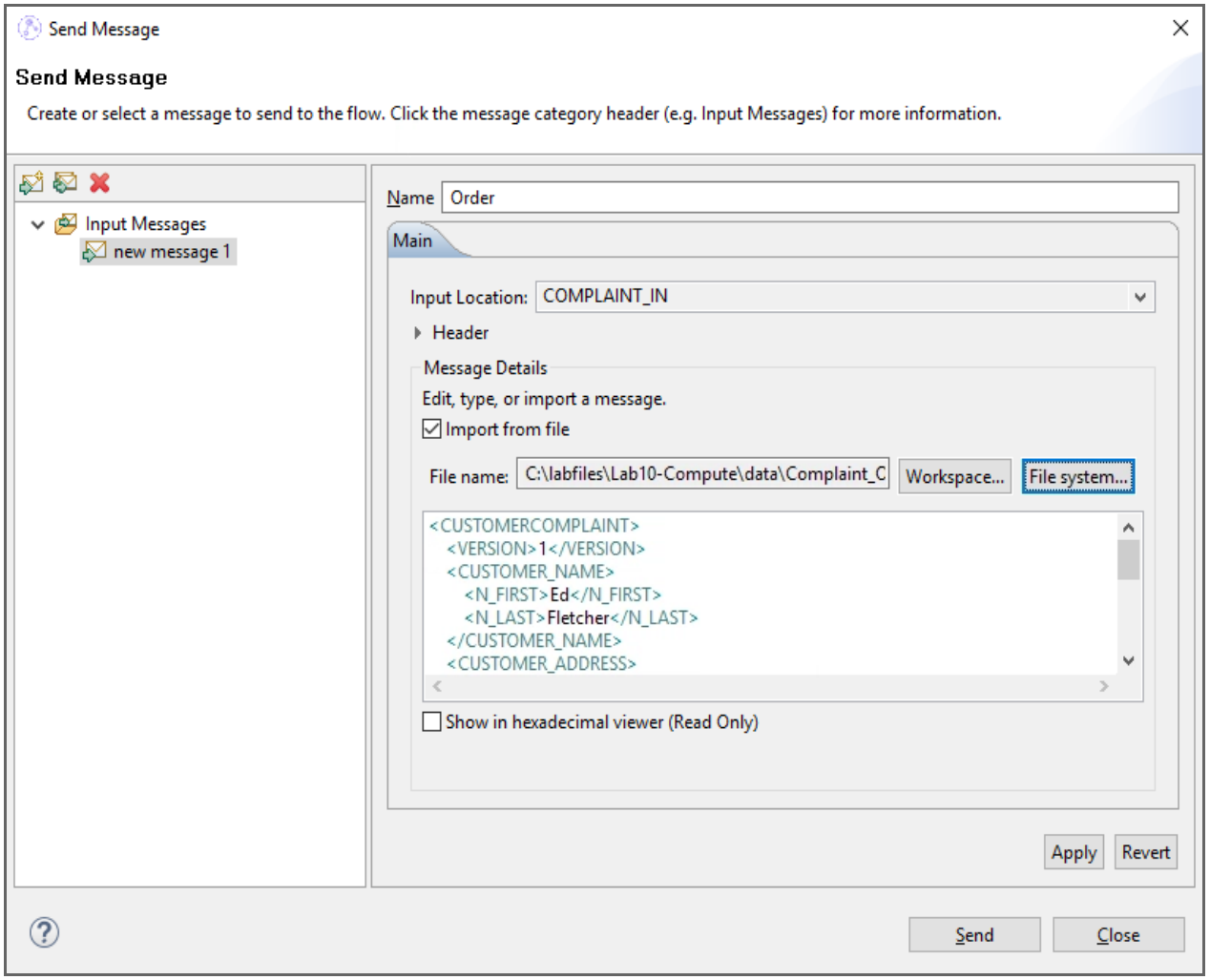


__ c. For the **Name**, type: `Order`

__ 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**.

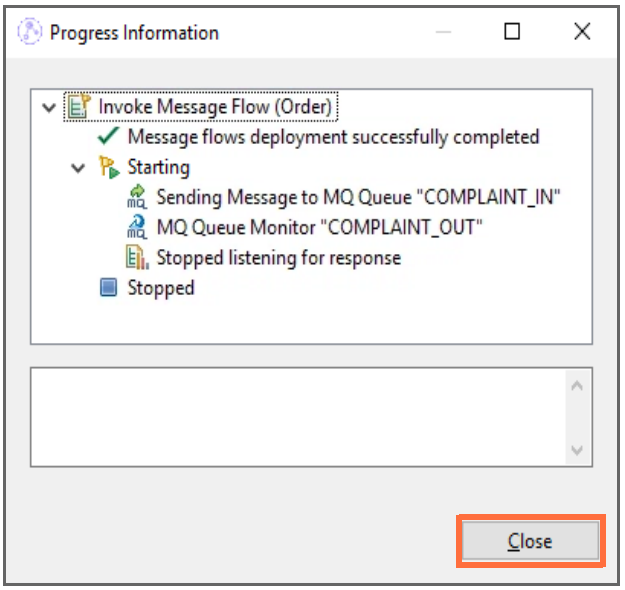
The file is imported into the Flow exerciser.



__ 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.

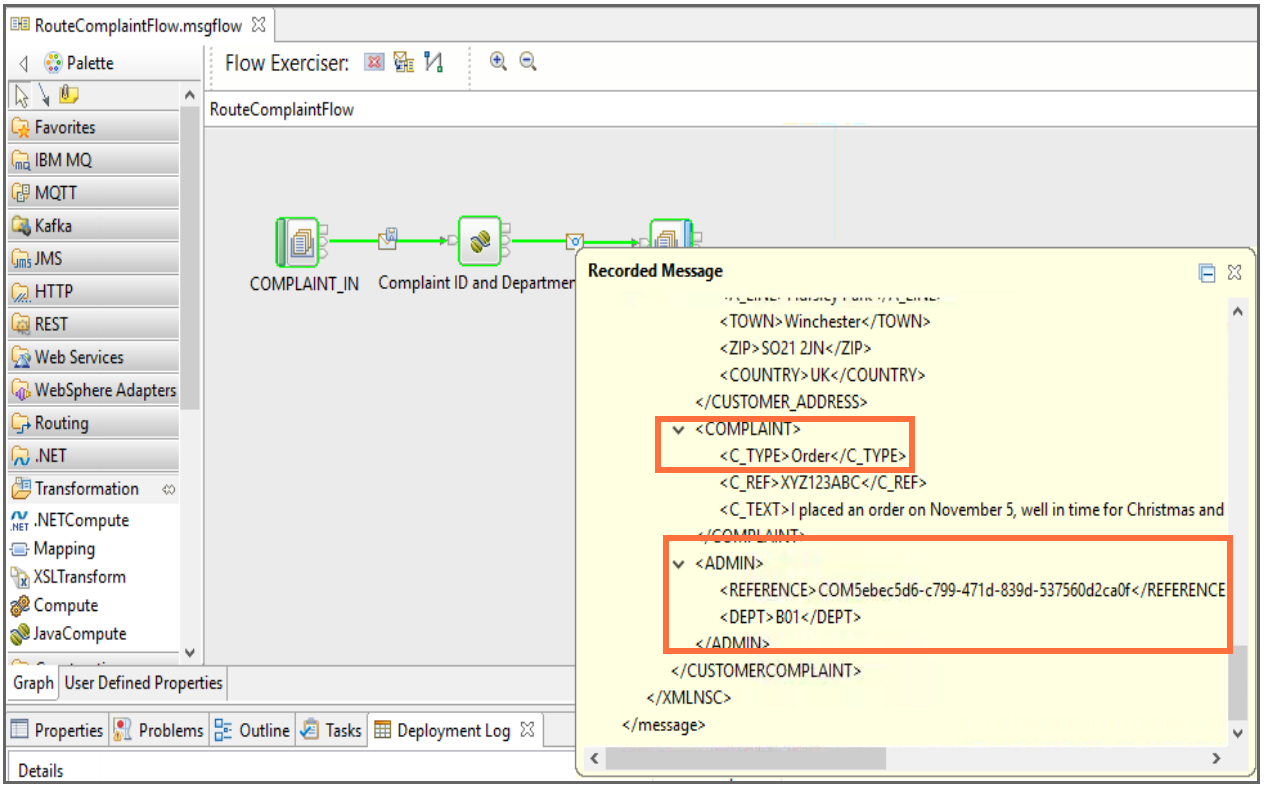
When the **Progress Information** window displays **Stopped**, the test is complete. Click **Close**.



__ 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** 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.

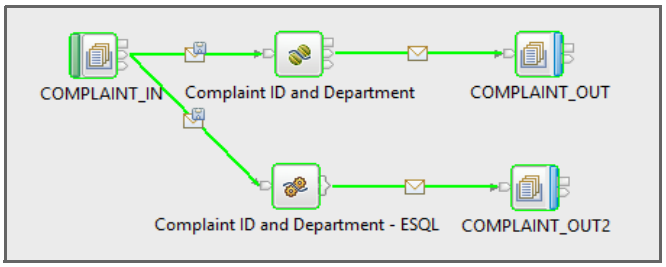
When you test the flow with `Complaint_Misc.xml`, the output message should contain the string `<DEPT>E01</DEPT.>` because the complaint is not about an order or a delivery.

__ 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: `ACEQM`
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. When you run the test, both paths are followed.



Part 5: 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 Explorer view**, right-click the **server1** integration server and click **Delete > Delete all resources**.

__ 4. In IBM MQ Explorer, delete the queues.

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 App Connect Enterprise 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

Estimated time

01:00

Overview

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.

Objectives

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

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. After preparing the environment, in the second 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 third 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 App Connect Enterprise Toolkit interprets custom keywords as information to display in the **Properties** view. In the fourth part of this exercise, you create keywords for the subflow that are displayed in the deployed message flow.

In the fifth 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 App Connect Enterprise V11 App Connect Enterprise Toolkit and IBM MQ V9
- Lab files in the `C:\labfiles\Lab11-UDP` directory
- User ace`admin` that is a member of the “mqm” group

Exercise instructions

Part 1: Exercise preparation

__ 1. To avoid any conflicts with the previous exercise, save the artifacts for this exercise in a new workspace that is named `C:\Workspace\Lab11`.

__ a. In the IBM App Connect Enterprise Toolkit, click **File > Switch Workspace > Other**.

__ b. For the **Workspace**, enter `C:\Workspace\Lab11`

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

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

__ 2. Verify that the node and server are running.

__ a. In the **Integration Explorer view**, verify that the integration node **ACENODE1** and **server1** are started.

__ b. Start them if they are stopped.

__ 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.

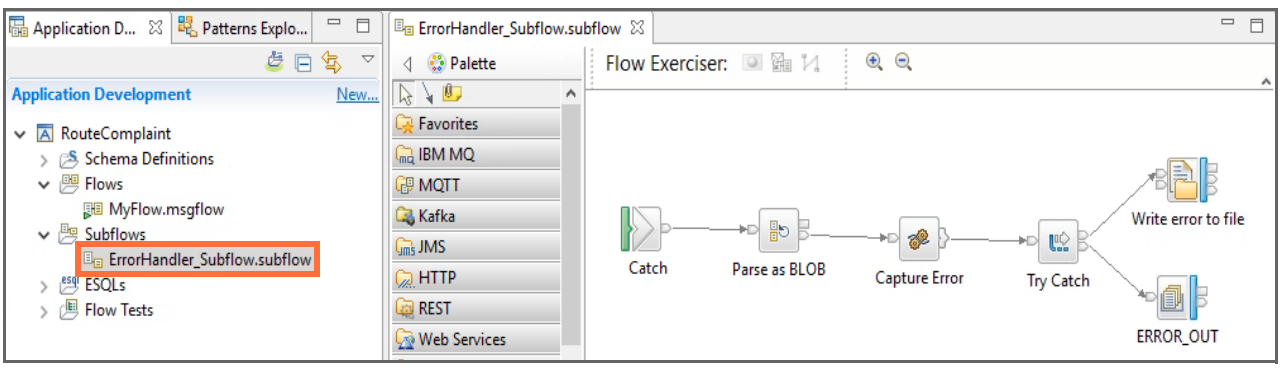
Part 2: Add user-defined property to subflow

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. Add a user-defined (UDP) property to the Error Handler subflow.

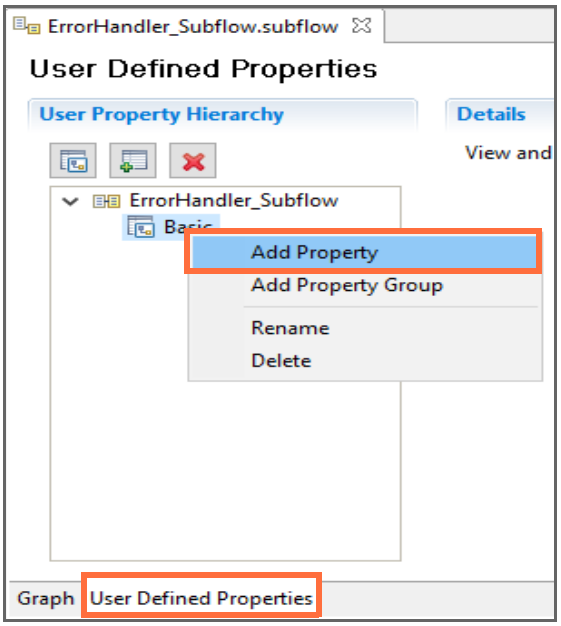
__ a. In the **Application Development** view, expand **RouteComplaint > Subflows**.

__ b. Double-click **ErrorHandler_Subflow** to open it in the Message Flow editor.



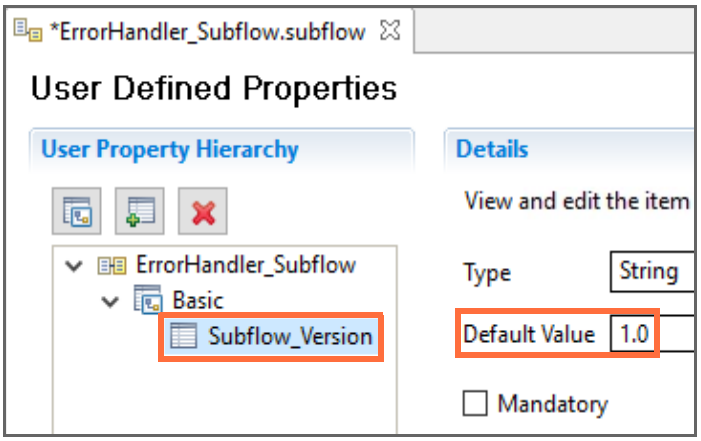
__ c. Click the **User Defined Properties** tab at the bottom of the Message Flow editor view.

__ d. In the **User Property Hierarchy** section, right-click **Basic** under **ErrorHandler_Subflow**, and then click **Add Property**.



__ e. Replace the default property name `Property1` by typing: `Subflow_Version`

__ f. Under the **Details** section, change the **Default Value** to: 1.0



__ g. Save the subflow (Ctrl+S).

Part 3: Promote subflow properties to the main flow

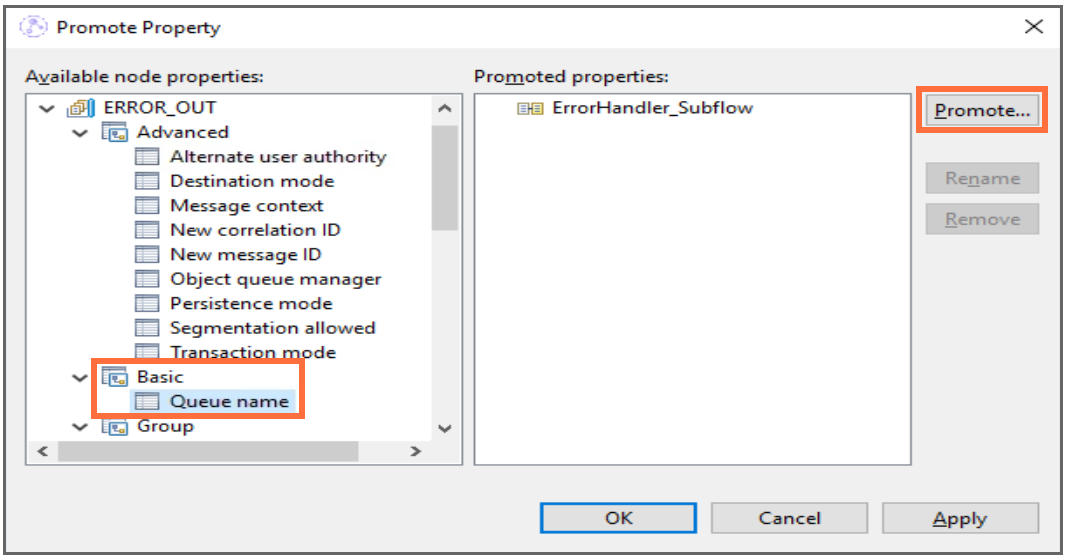
In this part of the exercise, you promote the **Queue name** and **MQ Connection** properties of the ERROR_OUT MQOutput node.

__ 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.

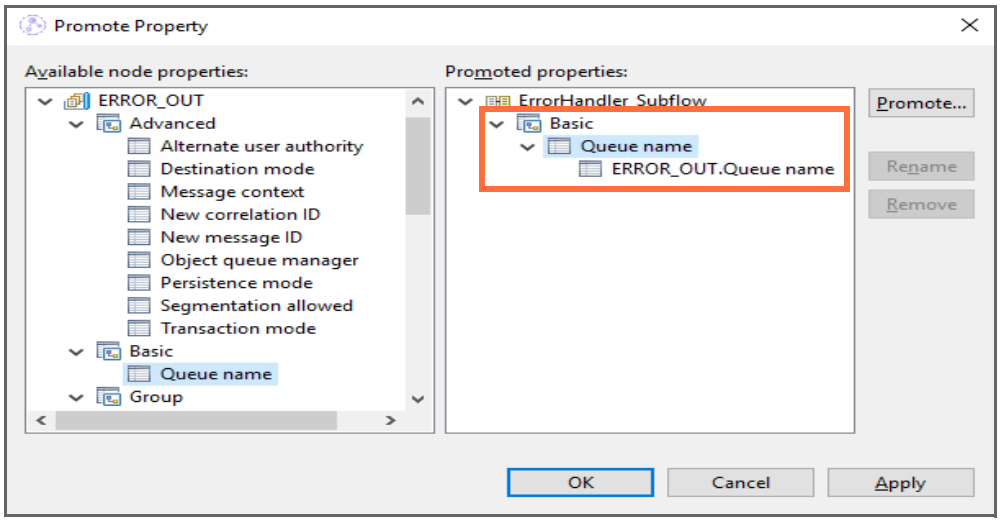
__ b. Right-click the ERROR_OUT MQ Output node, and then click **Promote Property** from the menu.

__ c. In the **Available node properties** window, click **Basic > Queue name** and then click **Promote**.



__ d. On the **Target Selection** window, select **ErrorHandler_Subflow**.

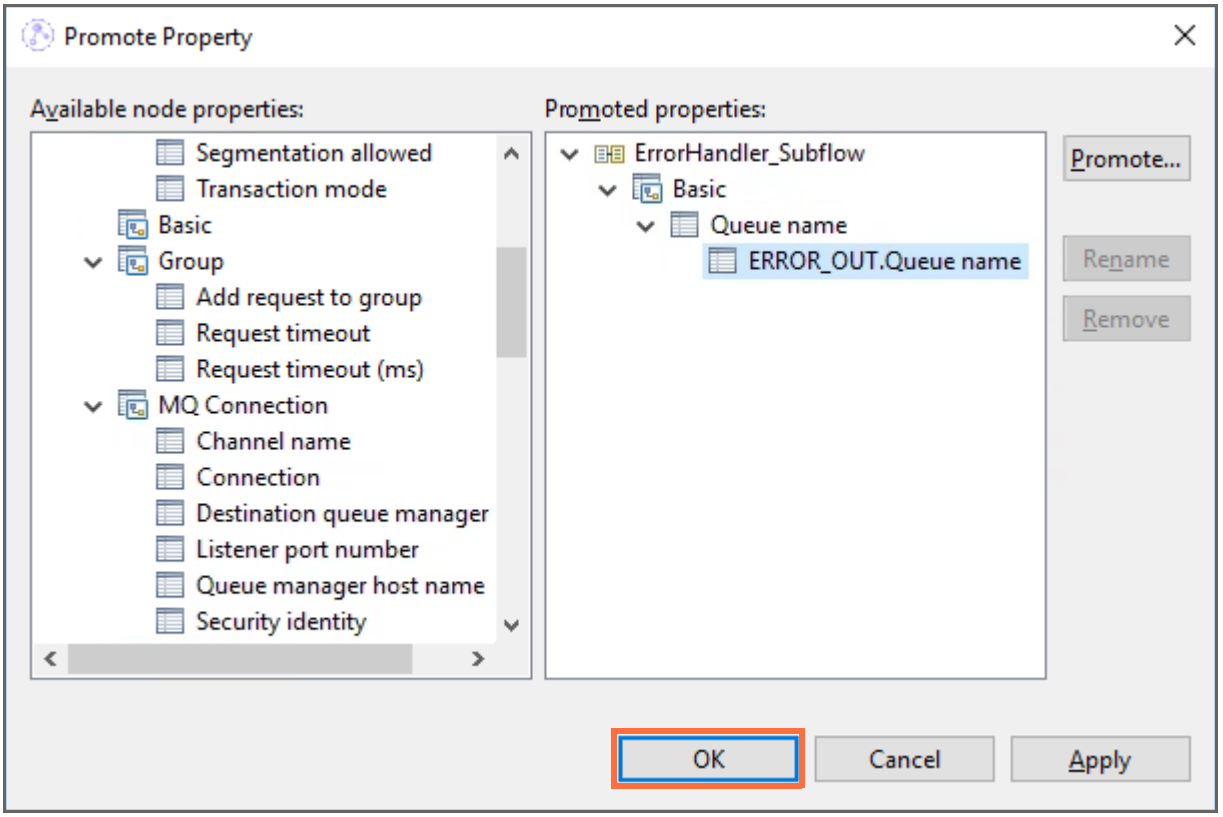
The **Queue name** property is shown in the **Promoted properties** window under the group **Basic**. If you expand the property, you see the property name, prefixed by the node label (ERROR_OUT).



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.

__ e. In the **Promote Property** window, click **OK**.

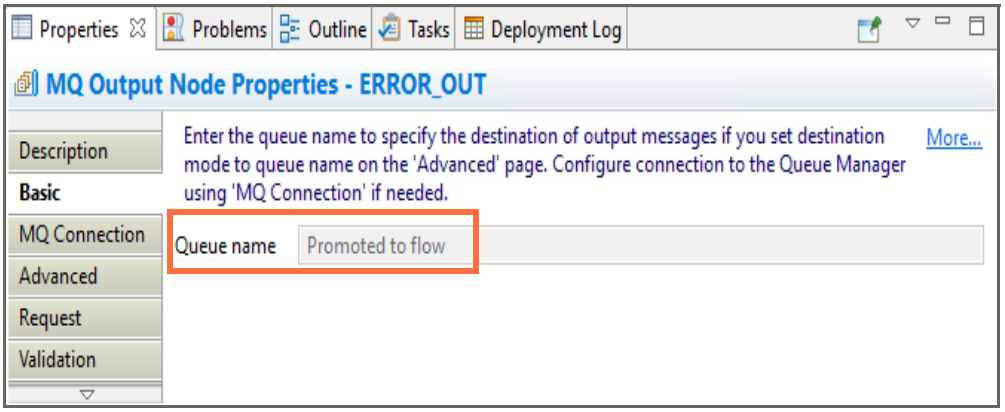


__ f. Save the subflow.

__ 2. Review the promoted properties.

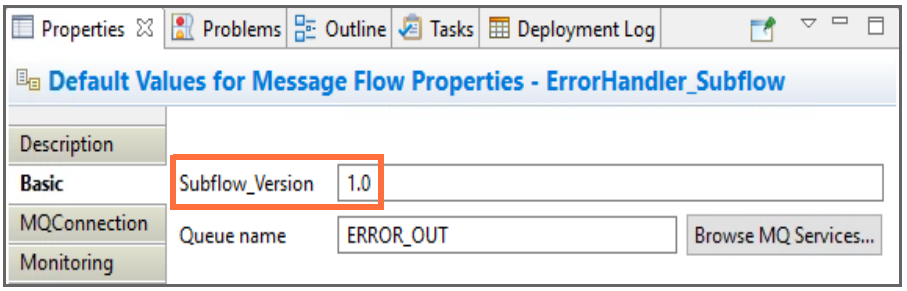
__ a. Click the ERROR_OUT node to display the node Properties view.

Observe that the **Queue name** on the **Basic** tab now shows 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.



__ b. Click in the white space in the Message Flow editor drawing canvas to display the message flow properties.

The **Queue name** property that you promoted in the previous step is displayed, as is the Subflow_Version user-defined property. Although you are not changing the values at this time, this location is where you would do so.



Part 4: Define custom keywords

You can define custom keywords to provide more information about message flow, such as the message flow author and version. The IBM App Connect Enterprise Toolkit interprets custom keywords as additional information to be displayed, in the **Properties** view.

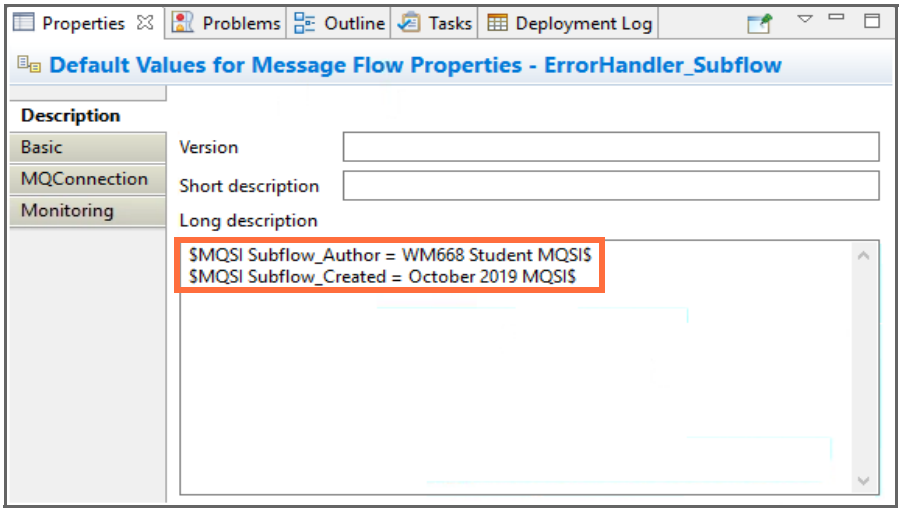
In this part of the exercise, you create keywords for the subflow to be displayed in the deployed message flow.

__ 1. Define custom keywords

__ a. In the **Properties** tab of the ErrorHandler_Subflow, click the **Description** tab.

__ b. In the Long Description property, type the following lines:

```
$MQSI Subflow_Author = WM668 Student MQSI$
$MQSI Subflow_Created = October 2019 MQSI$
```



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.

__ c. Save the message flow.

Part 5: 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.

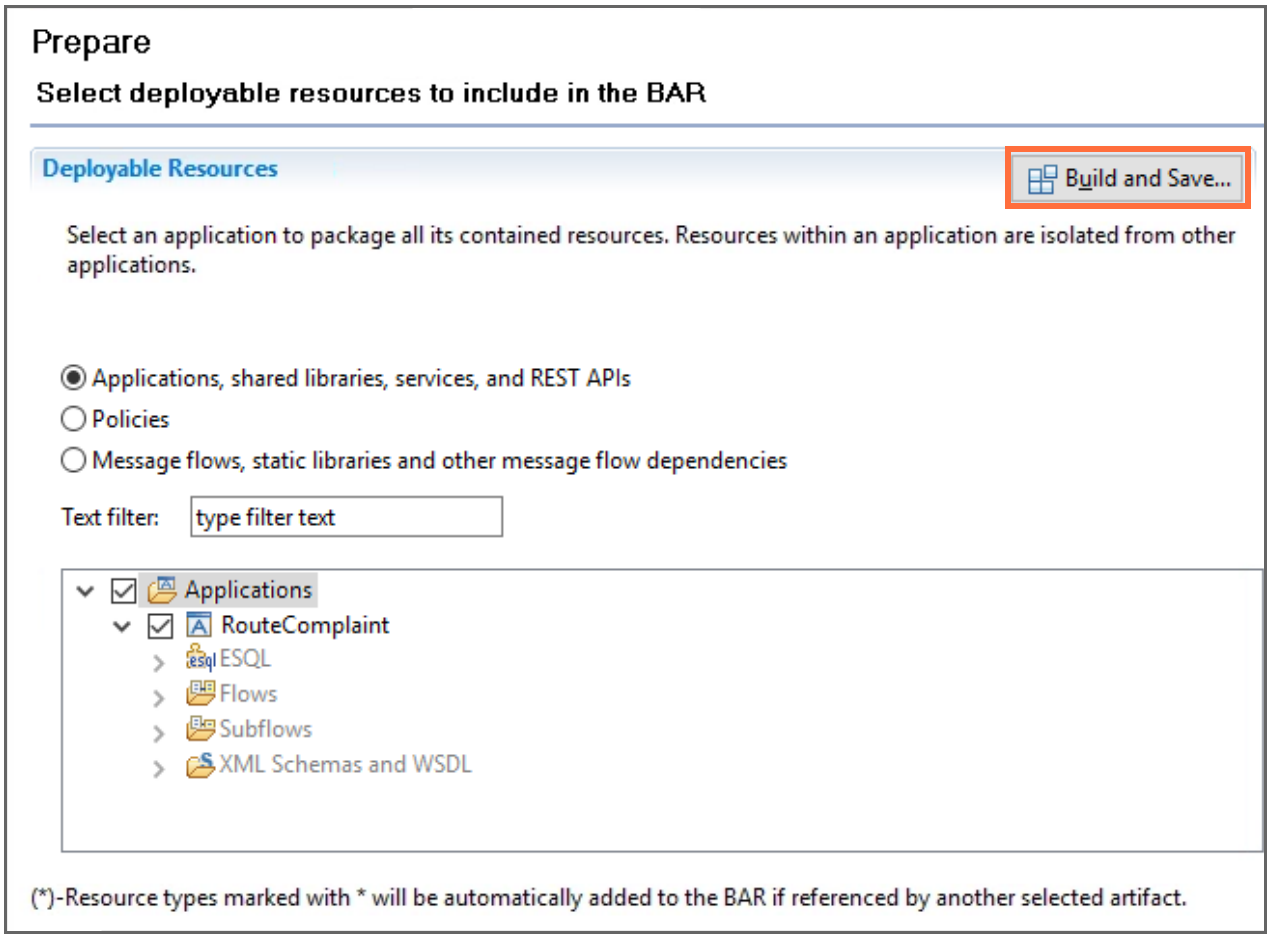
__ a. Click **File > New > BAR** file. The New Bar File windows opens.

__ 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** tab.

__ d. In the BAR file editor **Prepare** tab, select **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.

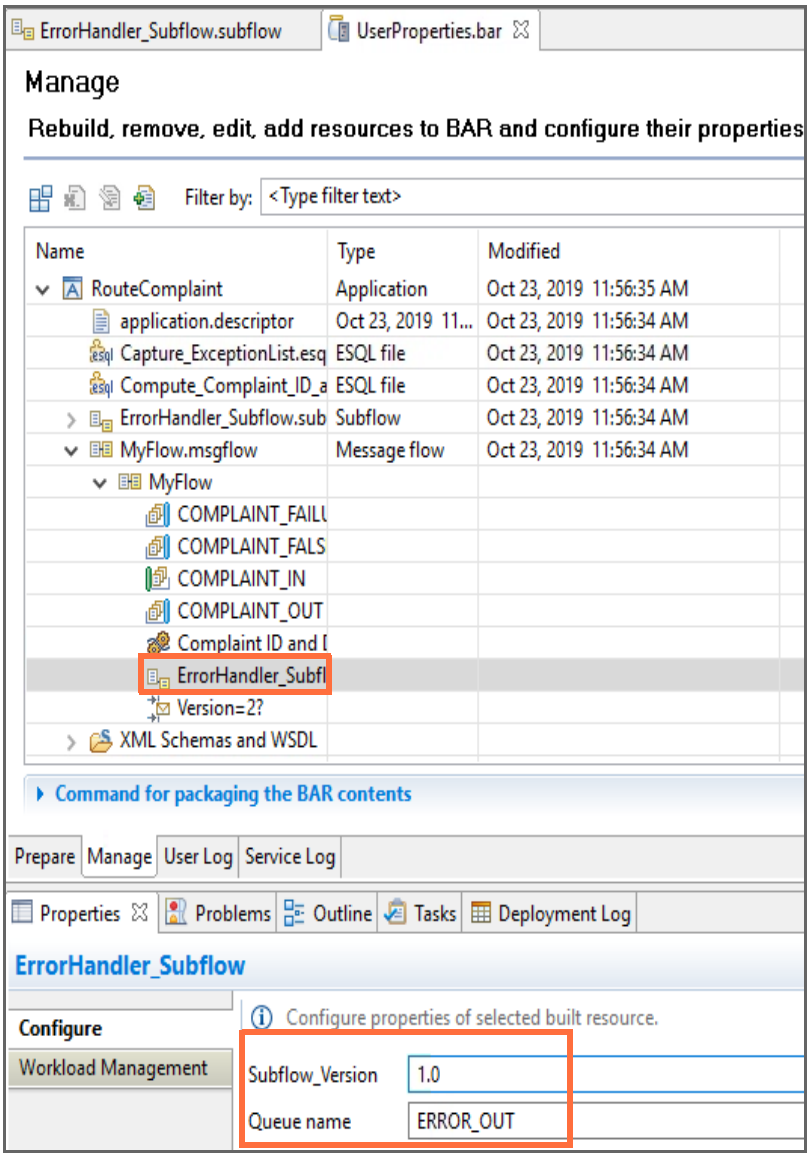
__ 2. Verify that the properties you promoted to the main flow from the subflow in Part 3 of this exercise are shown as properties of the main flow

__ a. Click the **Manage** tab at the bottom of the BAR File editor to go to the BAR File editor **Manage** tab.

__ b. In the BAR file editor **Manage** tab, expand **RouteComplaint**, then expand the main message flow **MyFlow.msgflow**.

__ c. Click **ErrorHandler_Subflow** under the **MyFlow** folder.

__ d. Verify that the properties you promoted to the main flow from the subflow are shown as properties of the main flow.



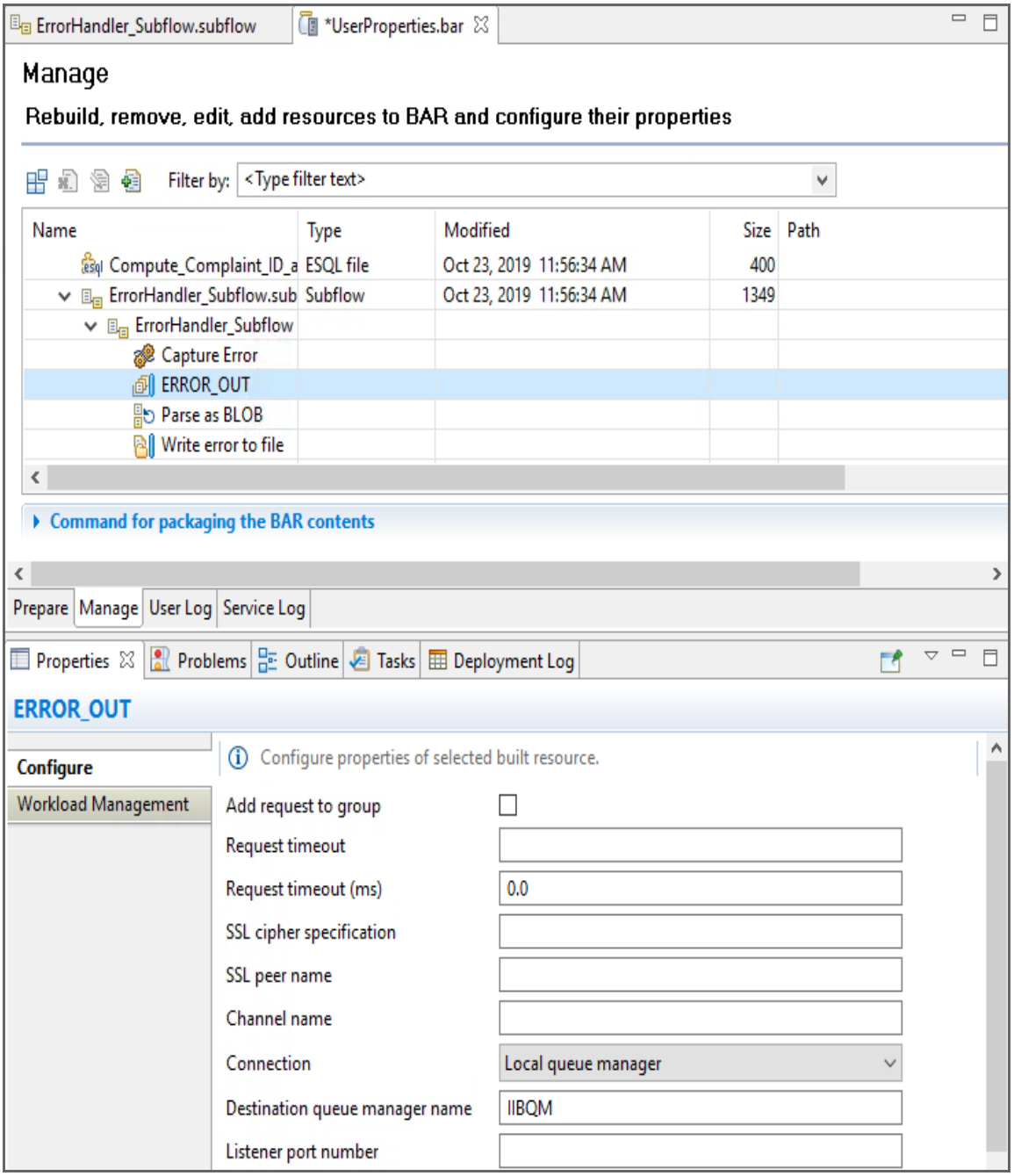
__ e. In the **ErrorHandler_Subflow** properties view, set the **SubFlow_Version** property to **1.2**.

__ f. Save your work.

__ 3. Check the subflow user-defined properties and promoted properties.

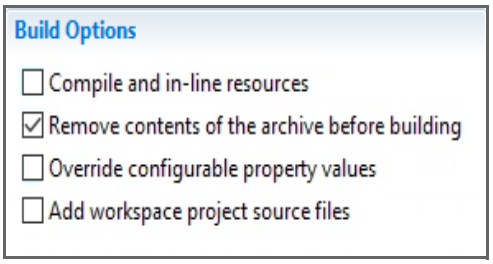
__ a. While on the **Manage** tab on the BAR file editor, expand **ErrorHandler_Subflow.subflow** and click **ERROR_OUT**.

__ b. Verify that the **Queue name** property is not available in the subflow.



__ 4. Deploy the BAR file.

__ a. On the **Prepare** tab, clear the **Override configurable property values** checkbox to use the configurable properties that you modified on the **Manage** tab.



__ b. Click **Build And Save**. The components are added to the BAR file.

__ c. Click **OK** on the Adding to BAR File confirmation window

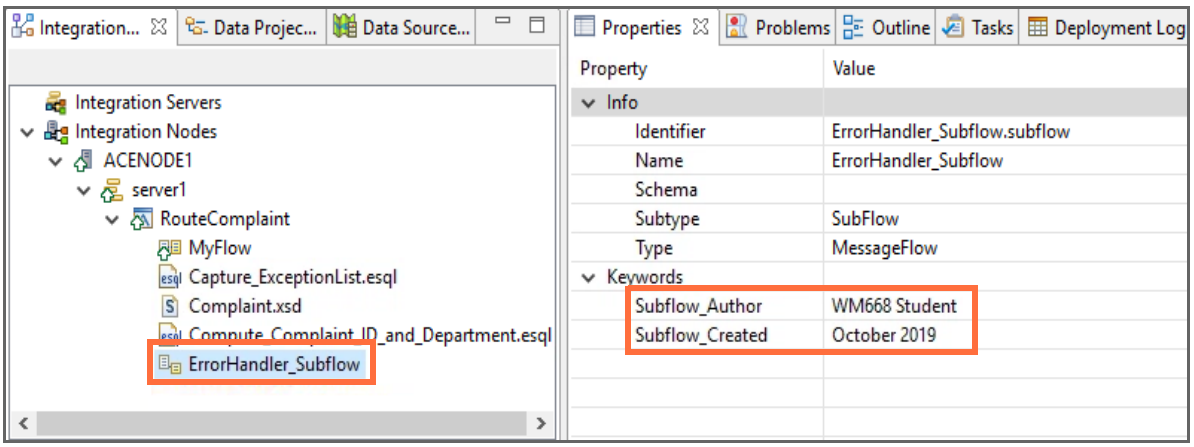
__ d. Drag the **UserProperties.bar** file to the **server1** integration server on the ACENODE1 Integration node.

__ 5. Review the BAR file attributes.

__ a. In the **Integration Explorer view**, expand the **RouteComplaint** application under **server1**.

__ 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 earlier in the exercise.



Part 6: Exercise clean-up

__ 1. Close all open editors.

__ 2. In the **Integration Nodes** view, right-click the **server1** integration server and click **Delete > all resources**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you prepared the development environment by importing a solution. Then, you added user-defined properties to a message flow.

In the third part of this exercise, you promoted subflow properties to the main flow.

In fourth part of this exercise, you created keywords for the subflow.

In the fifth 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

IBM Training