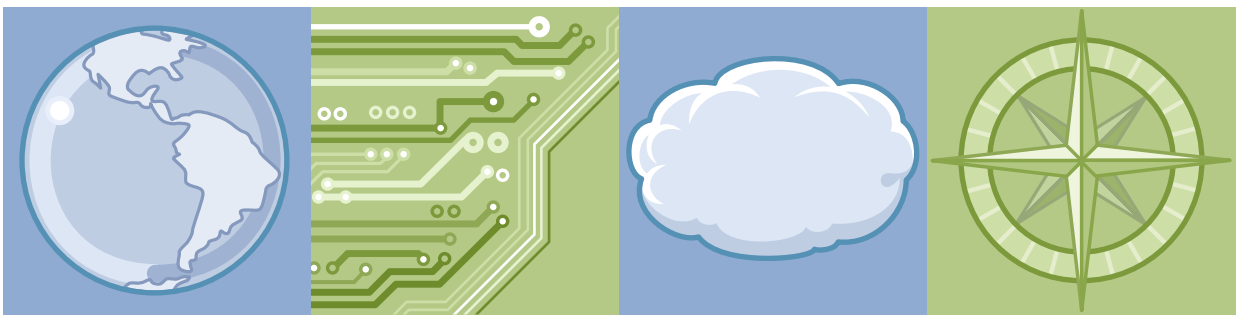
IBM Training

Student Exercises

IBM Integration Bus V10 Application Development II

Course code WM676/ZM676 ERC 1.0



Trademarks

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

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

CICS®	DataPower®	DB™
DB2®	developerWorks®	Express®
IMS™	PartnerWorld®	SurePOS™
Tivoli®	WebSphere®	Worklight®
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.

Sterling Commerce® is a trademark or registered trademark of IBM International Group B.V., an IBM Company.

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

March 2016 edition

Contents

Trademarks

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

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

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

CICS®	DataPower®	DB™
DB2®	developerWorks®	Express®
IMS™	PartnerWorld®	SurePOS™
Tivoli®	WebSphere®	Worklight®
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.

Sterling Commerce® is a trademark or registered trademark of IBM International Group B.V., an IBM Company.

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

Exercises description

This course includes the following exercises:

- Exercise 1. Implementing message aggregation
- Exercise 2. Extending a DFDL model
- Exercise 3. Implementing web services
- Exercise 4. Creating an integration service
- Exercise 5. Creating IBM MQ and database services
- Exercise 6. Creating a decision service
- Exercise 7. Implementing IBM Integration Bus runtime security
- Exercise 8. Recording and replaying message flow data

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

The login ID for the exercise image is: `iibadmin`

The password for the exercise image is: `web1sphere`

Important

Online course material updates might exist for this course. To check for updates, see the Instructor wiki at: `http://www.ibm.com/developerworks/connect/middleware_edu`

Exercise 1. Implementing message aggregation

What this exercise is about

In this exercise, you implement a customer feedback message flow by requesting customer information from a back-end system and aggregating the replies.

What you should be able to do

After completing this exercise, you should be able to:

- Add IBM Integration Bus SYSTEM.BROKER queues to a queue manager
- Use the Aggregate Control node and the Aggregate Request node to generate and concurrently fan-out related requests
- Use the Aggregate Reply node to aggregate messages into a single output message

Introduction

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` for the Windows platform

`/usr/labfiles` for the Linux platform

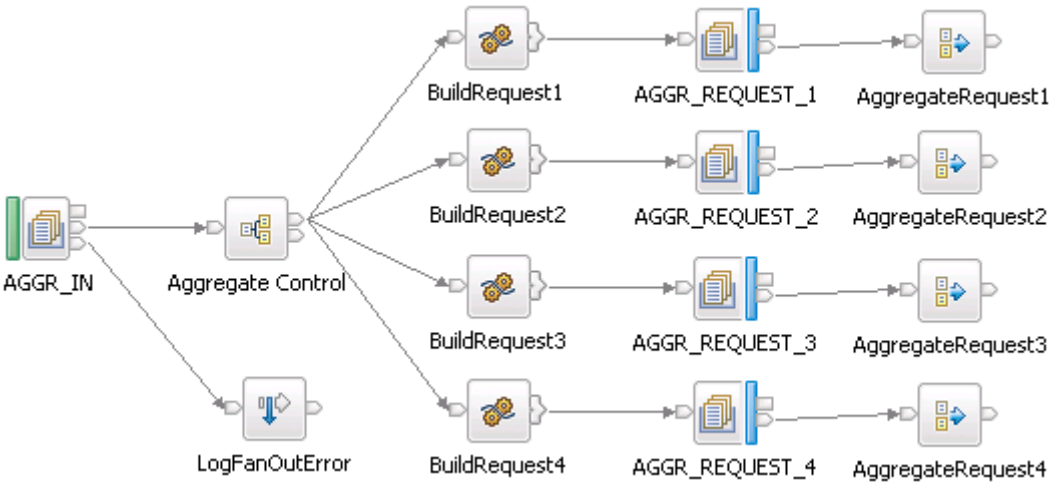
The exercises point you to the lab files as you need them.

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

Other Integration Bus capabilities require that a queue manager with special SYSTEM.BROKER queues is specified on the integration node. For example, event-driven message processing nodes such as the Aggregation node require that a queue manager is specified on the integration node.

In the first part of this exercise, you create a queue manager that is named IIBQM. You then create an integration node that is named IIBNODE_WITHQM. You associate this integration node with the queue manager and add the BROKER.SYSTEM queues to the queue manager.

In the second part of this exercise, you configure a fan-out message flow to generate four different request messages and start the tracking of the aggregation operation. The following figure shows the fan-out message flow.

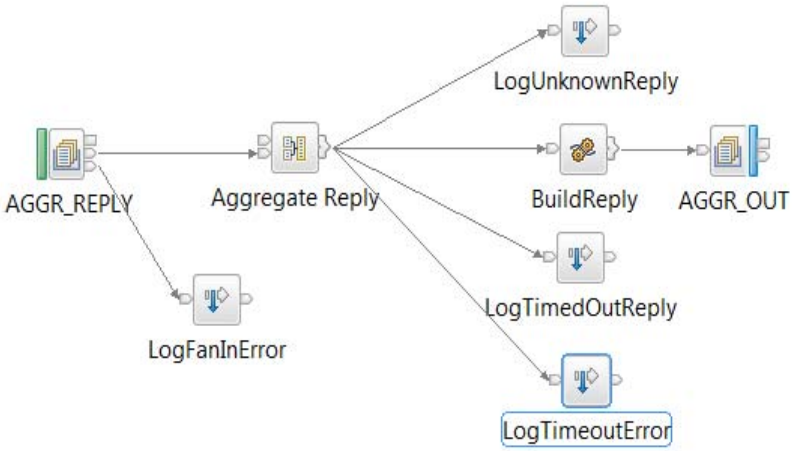


The messages are sent to a request/reply flow to simulate the back-end service applications that typically process the request messages from the aggregation operation. The following figure shows the request/reply flow.

In the third part of the exercise, you configure the fan-in message flow to receive the replies from the request/reply message flow, and aggregate the message into a single output message. The fan-in flow does not aggregate the messages in any specific order.

The output message from the Aggregate Reply node does not contain an IBM MQ message descriptor (MQMD). The flow includes a Compute node to add an MQMD before sending the message to an MQ Output node (AGGR_OUT). The ESQL in the Compute node adds a rudimentary MQMD. It then copies the data from `ComIbmAggregateReplyBody` in the input message into an XML tree in the output message, while maintaining the

aggregate request identifiers and folders. The following figure shows the fan-in message flow.



In the fourth part of this exercise, you create and deploy the BAR file. You test the message flow by using the Integration Toolkit Test Client with **Enqueue** and **Dequeue** flow test events.

A project interchange file that contains the completed flows is provided in the `C:\labfiles\Lab01-Aggr\solution` directory.

Requirements

- IBM Integration Bus V10
- IBM MQ V8 and a basic understanding of IBM MQ administration
- IBM MQ queues AGGR_REPLY, AGGR_OUT, AGGR_IN, and AGGR_REQUEST (created in the exercise)
- Lab exercise files in the `C:\labfiles\Lab01-Aggr` directory

Output message example

The example shows excerpts from the output message that the AggregateReply node generates.

```
<ComIbmAggregateReplyBody>
 <Request2>
  <ReplyIdentifier>X'414d5120494239514d47522020202020564a1b5220006e02'</ReplyIdentifier>
  <SaleEnvelope>
   <SaleList>
    <Invoice>
     <Initial>q</Initial>
     <Initial>F</Initial>
     <Surname>TJnBitwOBVU</Surname>
     <Item>
      <Code>03</Code>
      <Code>07</Code>
      <Code>05</Code>
      <Description>cXYWNui</Description>
      <Category>bHFSN</Category>
      <Price>55.03</Price>
      <Quantity>03</Quantity>
     </Item>
.....
</SaleList>
  </SaleEnvelope>
 </Request2>
 <Request3>
<ReplyIdentifier>X'414d5120494239514d47522020202020564a1b5220006e03'</ReplyIdentifier>
  <SaleEnvelope>
   <SaleList>
    <Invoice>
     <Initial>g</Initial>
     <Initial>o</Initial>
     <Surname>AwqJMgJZpMj</Surname>
     <Item>
      <Code>01</Code>
      <Code>05</Code>
      <Code>03</Code>
      <Description>yDWigPo</Description>
      <Category>tGpHi</Category>
      <Price>40.64</Price>
      <Quantity>02</Quantity>
.....
</SaleEnvelope>
 </Request4>
</ComIbmAggregateReplyBody>
```

Exercise instructions

Part 1: Create an integration node that has a default queue manager

In this part of the exercise, you use IBM MQ Explorer to create a queue manager that is named IIBQM. You then use the Integration Toolkit to create an integration node that is named IINODE_WITHQM that uses the queue manager IIBQM as its default queue manager. Finally, you use an Integration Bus sample command file to add the SYSTEM.BROKER queues and topics to the IIBQM queue manager that the IINODE_WITHQM integration node uses.

__ 1. Start IBM MQ Explorer.

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

__ 2. In IBM MQ Explorer, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ`.

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

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

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

__ d. Click **Finish**.

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

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

__ 3. In IBM MQ Explorer, create a queue that is named `DLQ` for the dead-letter queue.

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

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

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

__ d. Click **Finish**.

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

__ f. Verify that the queue **DLQ** is listed in the **Queues** content view.

__ 4. In the Integration Toolkit, create an integration node that is named `IIBNODE_WITHQM` with an integration server that is named `default`.

__ a. Start the IBM Integration Toolkit by double-clicking the shortcut icon on the desktop or by clicking **All Programs > IBM Integration Bus 10.0.0.0 > IBM Integration Toolkit 10.0.0.0** from the Windows **Start** menu.

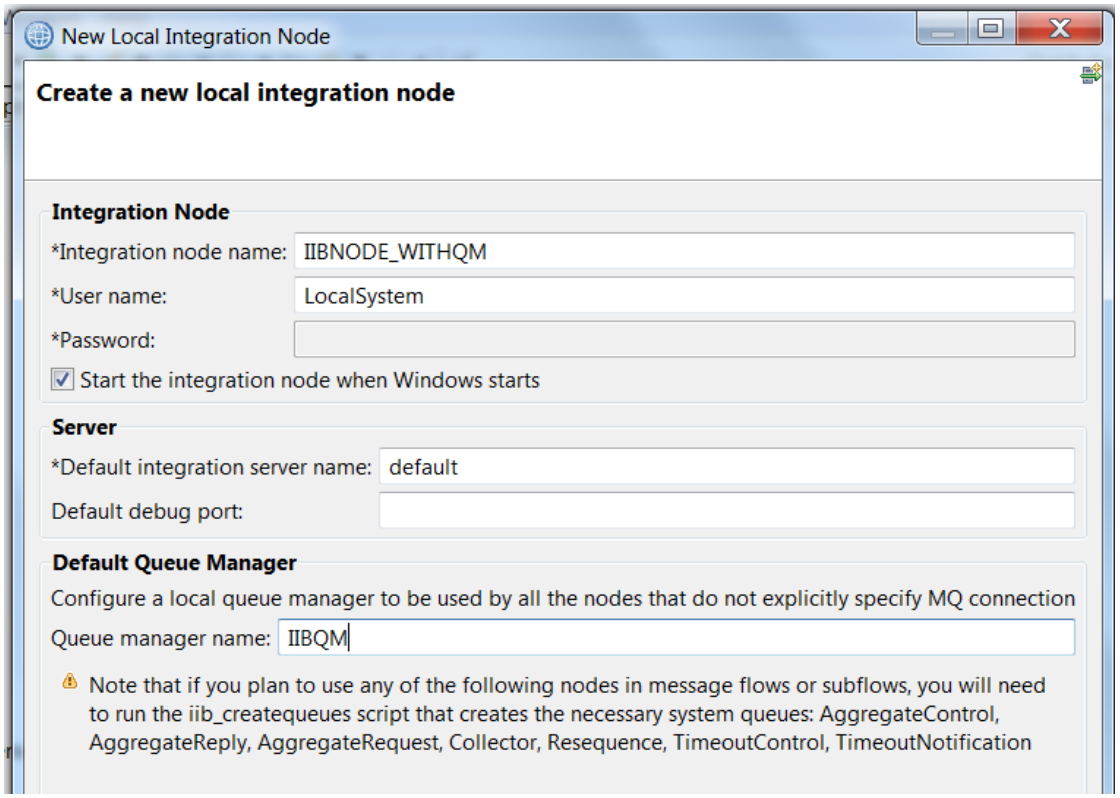
__ b. Close the **Welcome** window.

__ c. In the **Integration Nodes** view, right-click **Integration Nodes** and then click **New > Local Integration Node**.

__ d. For the **Integration node name**, type: `IIBNODE_WITHQM`

__ e. For the **Default integration server name**, accept the default integration server name (**`default`**).

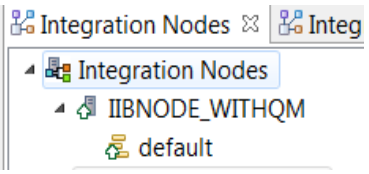
__ f. For the **Queue manager name**, type: `IIBQM`



__ g. Click **Finish**.

__ h. Click **Close** on the **Progress Information** window after the integration node and integration server are created.

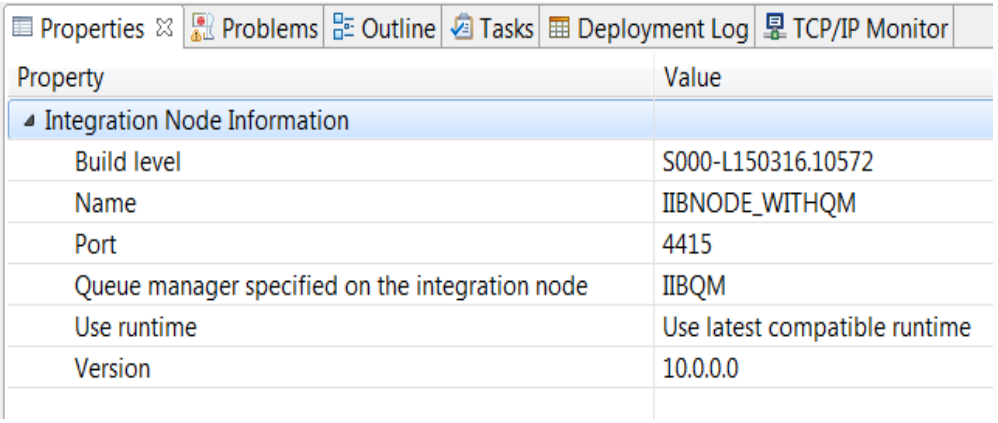
The new integration node and integration server should be shown in the **Integration Nodes** view.



__ 5. Verify the integration node properties.

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

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



__ 6. Create the SYSTEM.BROKERS queues on the integration node queue manager by running the `iib_createqueues.cmd` file.

__ a. Start an IBM Integration Console as an administrator by right-clicking the IBM Integration Console shortcut icon on the desktop and then clicking **Run as administrator.** Click **Yes** to allow the application to update the computer.

__ b. Type the following command to go to the Integration Bus sample directory for IBM MQ. This directory contains the batch file that creates the SYSTEM.BROKER queues.

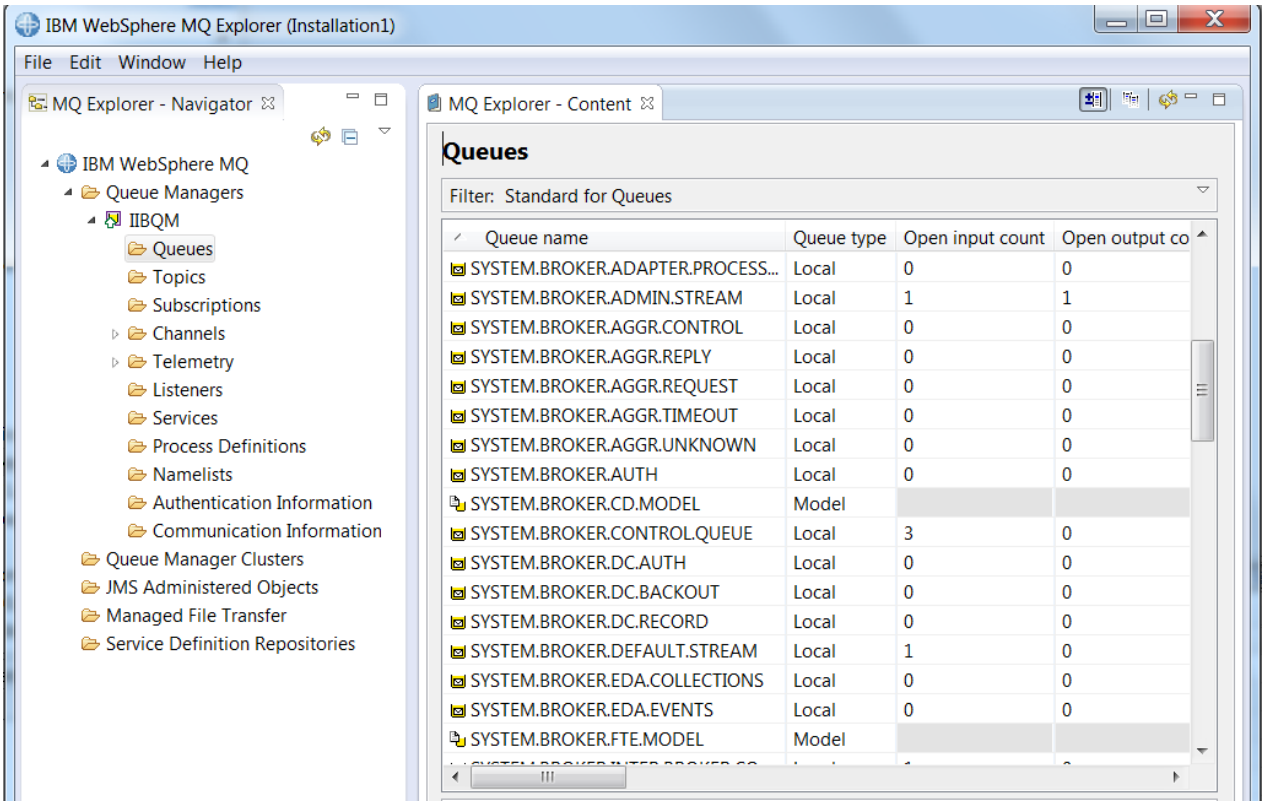
```
cd server\sample\wmq
```

__ c. Type the following command to run the command file on the IIBQM queue manager.

```
iib_createqueues.cmd IIBQM
```

You should see messages that indicate that the SYSTEM.BROKER queues and topics are created. You should also see messages that indicate that security is set on the queues by using the IBM MQ `setmqaut` command.

__ 7. In IBM MQ Explorer, display that the SYSTEM queues on the queue manager IIBQM and verify that the queue manager contains the Integration Bus SYSTEM.BROKER queues.

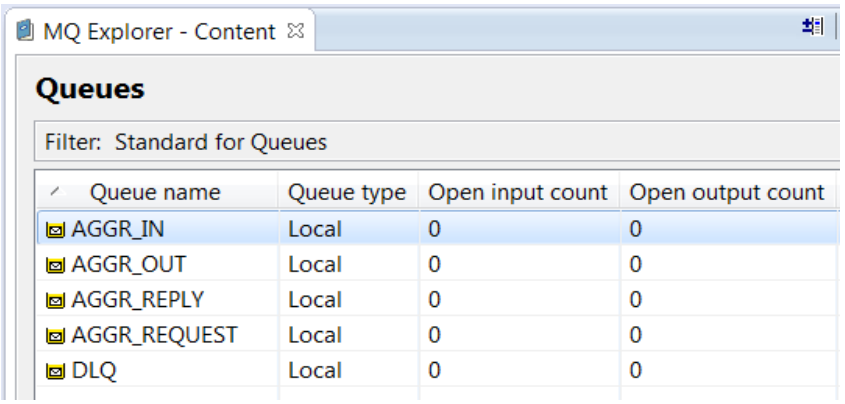


__ 8. Create the application queues that are required for this exercise: AGGR_REPLY, AGGR_OUT, AGGR_IN, and AGGR_REQUEST

In the IBM Integration Console, type:

```
runmqsc IIBQM < C:\labfiles\Lab01-Aggr\resources\Q_Defs.mqsc
```

__ 9. Verify that the application queues AGGR_IN, AGGR_OUT, and AGGR_REPLY were created successfully.



Part 2: Complete the fan-out message flow

In this part of the exercise, you configure the fan-out message flow to generate four different request messages and start the tracking of the aggregation operation.

__ 1. Switch to a new workspace in the Integration Toolkit.

__ a. Click **File > Switch Workspace > Other** from the toolbar.

__ b. For **Workspace**, enter an appropriate value such as `C:\Workspace\Aggr` and then click **OK**.

__ c. Wait for the workspace to open. This operation can take several minutes.

__ d. When the **Welcome** page is displayed, close it to display the **Integration Development** perspective.

__ 2. Import the starting project into the Integration Toolkit workspace.

__ a. Click **File > Import**.

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

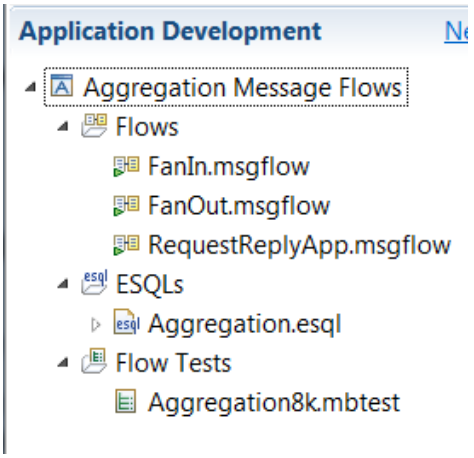
__ c. To the right of **From zip file**, click **Browse**.

__ d. Go to the `C:\labfiles\Lab01-Aggr\resources` directory.

__ e. Click the `AggregationLabStartingPoint_PI.zip` file and then click **Open**. The **Import Projects** dialog box is displayed.

__ f. Ensure that **Aggregation Message Flows** is selected, and then click **Finish**.

The project interchange file includes the **Aggregation Message Flows** application.

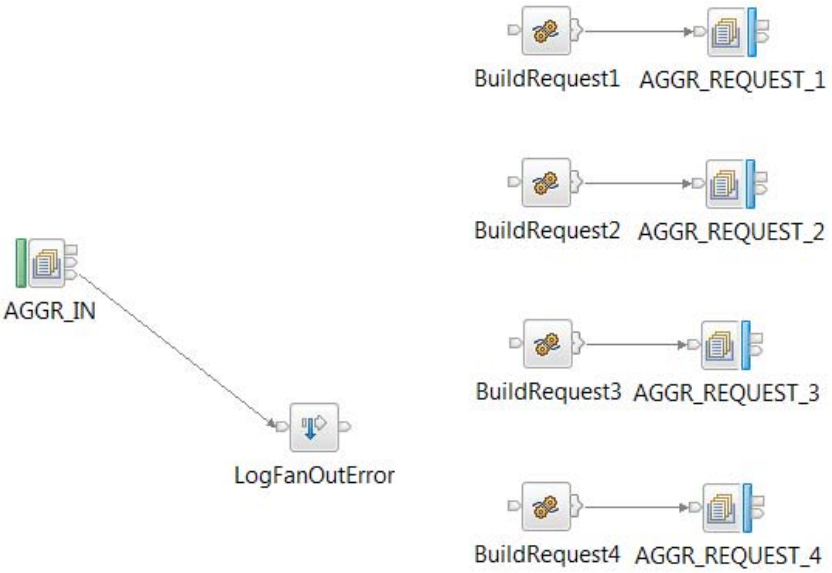


__ 3. Open the **FanOut.msgflow** message flow.

__ a. In the **Application Development** view, expand **Aggregation Message Flows > Flows**.

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

__ 4. Review the current nodes that are provided in the message flow.



- The AGGR_IN MQ Input node gets a message from the AGGR_IN queue from the default queue manager (IIBQM). The **Catch** terminal is wired to the Trace node that is named **LogFanOutError**. If the message flow fails, the Trace node captures the error information in the local error log.
- The BuildRequest Compute nodes build the request messages. Each BuildRequest node is connected to an AGGR_REQUEST_*n* MQ Output node that puts the request message on the AGGR_REQUEST queue on the default queue manager (IIBQM).

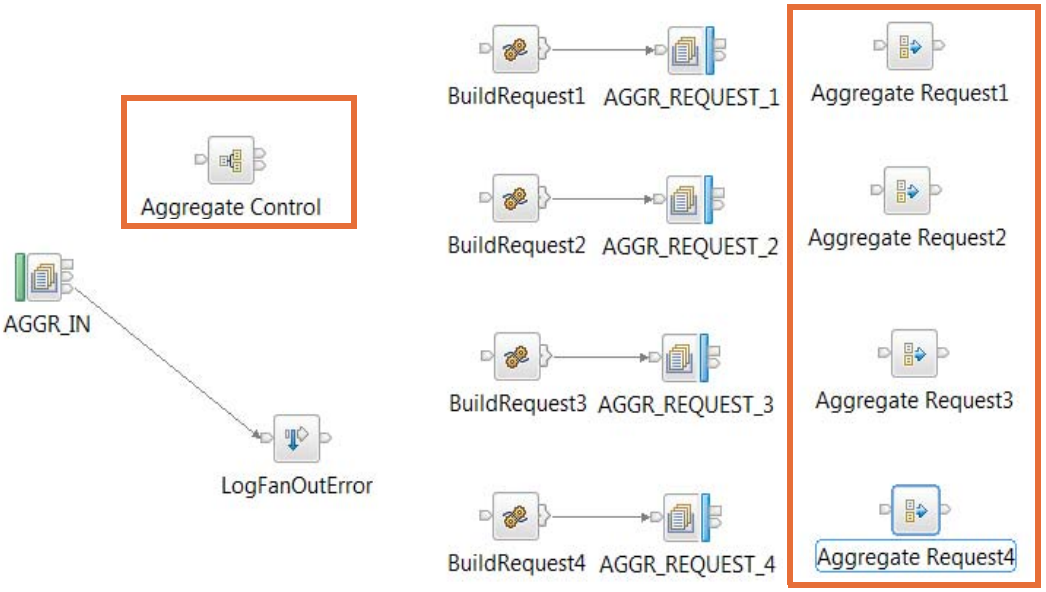
__ 5. Add the following nodes to the message flow from the **Routing** drawer.

__ a. Add an Aggregate Control node between the AGGR_IN node and the BuildRequest Compute nodes.

__ b. Add one Aggregate Request node after each AGGR_REQUEST node.

__ c. Starting with the last Aggregate Request node that you added to the canvas, change the names of the Aggregate Request nodes to match the AGGR_REQUEST node.

For example, change the name for Aggregate Request node for AGGR_REQUEST_4 to: `Aggregate Request4`



__ 6. Wire the new nodes as follows:

__ a. Wire the **Out** terminal of the **AGGR_IN** MQ Input node to the **In** terminal of the **Aggregate Control** node.

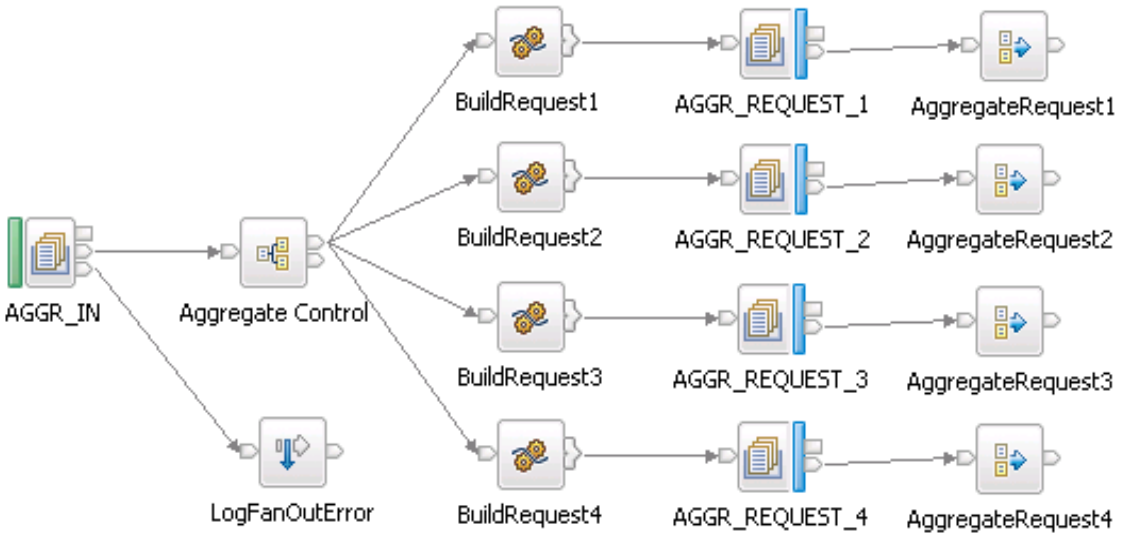
__ b. Wire the **Out** terminal of the **Aggregate Control** node to the **In** terminal of **BuildRequest1**, **BuildRequest2**, **BuildRequest3**, and **BuildRequest4** Compute nodes.

__ c. Wire the **Out** terminal of the **AGGR_REQUEST_1** MQ Output node to the **In** terminal of the **Aggregate Request1** node.

__ d. Wire the **Out** terminal of the **AGGR_REQUEST_2** MQ Output node to the **In** terminal of the **Aggregate Request2** node.

__ e. Wire the **Out** terminal of the **AGGR_REQUEST_3** MQ Output node to the **In** terminal of the **Aggregate Request3** node.

__ f. Wire the **Out** terminal of the **AGGR_REQUEST_4** MQ Output node to the **In** terminal of the **Aggregate Request4** node.



__ 7. Configure the node properties for **Aggregate Control** node.

For the **Aggregate name**, type: `SalesAggr`

__ 8. Configure the **Folder name** property for each of the Aggregate Request nodes.

The **Folder name** is the name that is used as a folder in the Aggregate Reply node's compound message to store the reply to this request.

__ a. For the **Aggregate Request1** node, set **Folder name** to: `Request1`

__ b. For **AggregateRequest2** node, set **Folder name** to: `Request2`

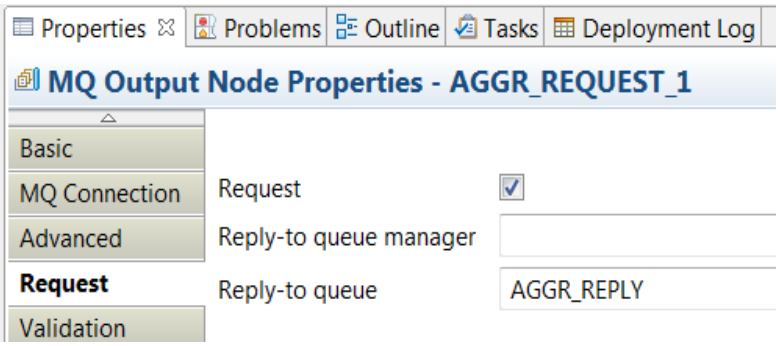
__ c. For **AggregateRequest3** node, set **Folder name** to: `Request3`

__ d. For **AggregateRequest4** node, set **Folder name** to: `Request4`

__ 9. Verify that the messages that the **FanOut.msgflow** generates are request messages.

__ a. Click the **AGGR_REQUEST_1** MQ Output node in the Message Flow editor to display its properties.

__ b. On the **Request** Properties, verify that **Request** is checked and that **Reply-to queue** is set to AGGR_REPLY.



__ c. Verify that the **Request** properties for the nodes that are named **AGGR_REQUEST_2**, **AGGR_REQUEST_3**, and **AGGR_REQUEST_4** are the same as the settings for **AGGR_REQUEST_1**.

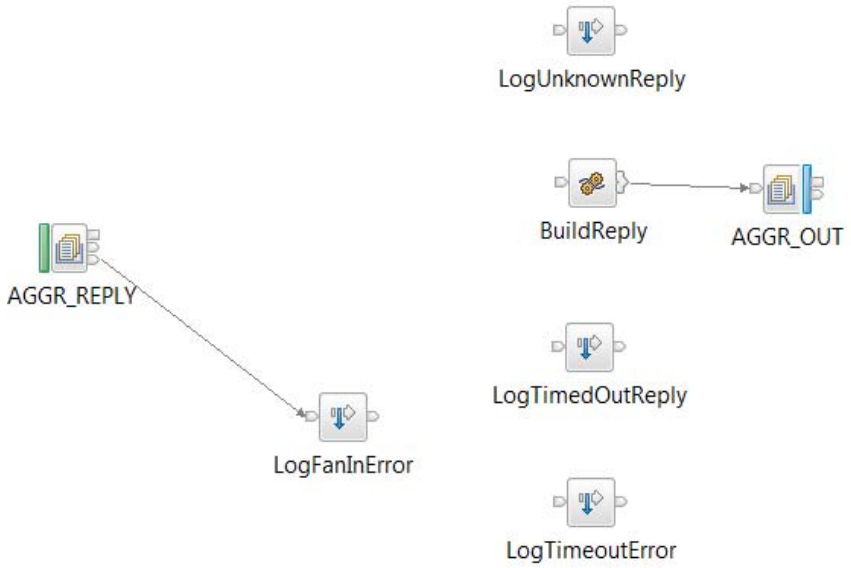
__ 10. Save the message flow.

__ 11. Verify that no problems are listed on the **Problems** tab.

Part 3: Complete the fan-in message flow

In this part of the exercise, you configure the aggregation node for the fan-in message flow.

__ 1. Open **FanIn.msgflow** in the Message Flow editor.



__ 2. Add an Aggregate Reply node after the AGGR_REPLY node. The Aggregate Reply node is in the **Routing** drawer of the Message Flow editor Palette.

__ 3. Wire the Aggregate Reply n**ode** as follows:

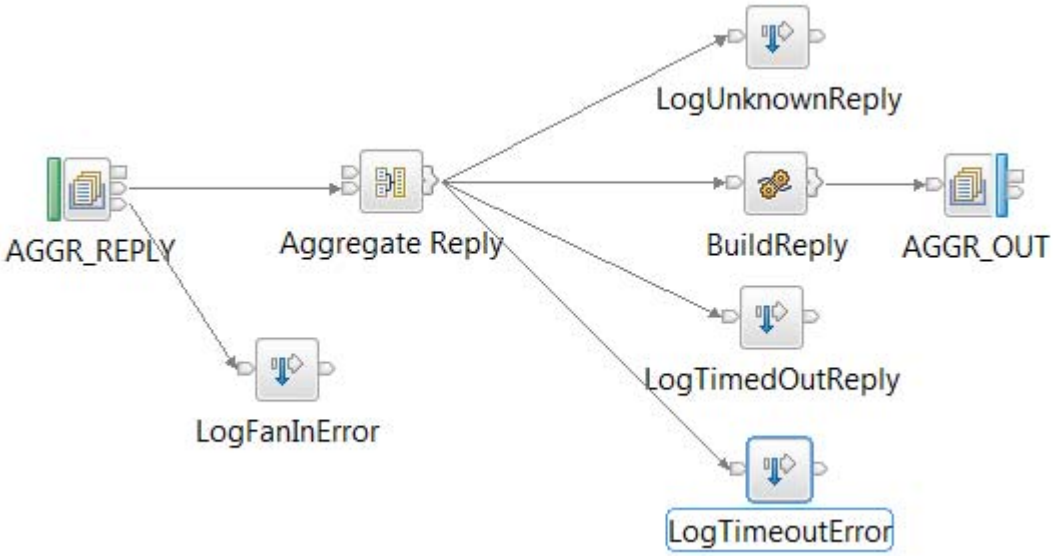
__ a. Wire the **Out** terminal of the **AGGR_REPLY** MQInput node to the **In** terminal of the **Aggregate Reply** node.

__ b. Wire the **Out** terminal of the **Aggregate Reply** node to the **In** terminal of **BuildReply** Compute nodes.

__ c. Wire the **Unknown** terminal of the **Aggregate Reply** node to the **In** terminal of the **LogUnknownReply** Trace node.

__ d. Wire the **Timeout** terminal of the **Aggregate Reply** node to the **In** terminal of the **LogTimedOutReply** Trace node.

__ e. Wire the **Catch** terminal of the **Aggregate Reply** node to the **In** terminal of the **LogTimeoutError** Trace node.

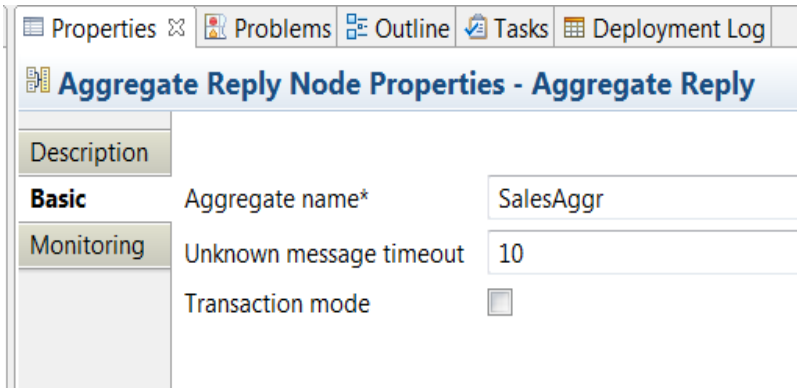


__ 4. Configure the node properties for **Aggregate Reply** node.

__ a. For the **Aggregate name**, enter the same name that you entered on the **Aggregate Control** node in the Fan Out message flow: `SalesAggr`

__ b. For the **Unknown message timeout**, type: `10`

__ c. Clear the **Transaction mode** check box.



__ 5. Save the message flow.

__ 6. Verify that no problems are listed on the **Problems** tab.

Part 4: Deploy and test the aggregation request/reply

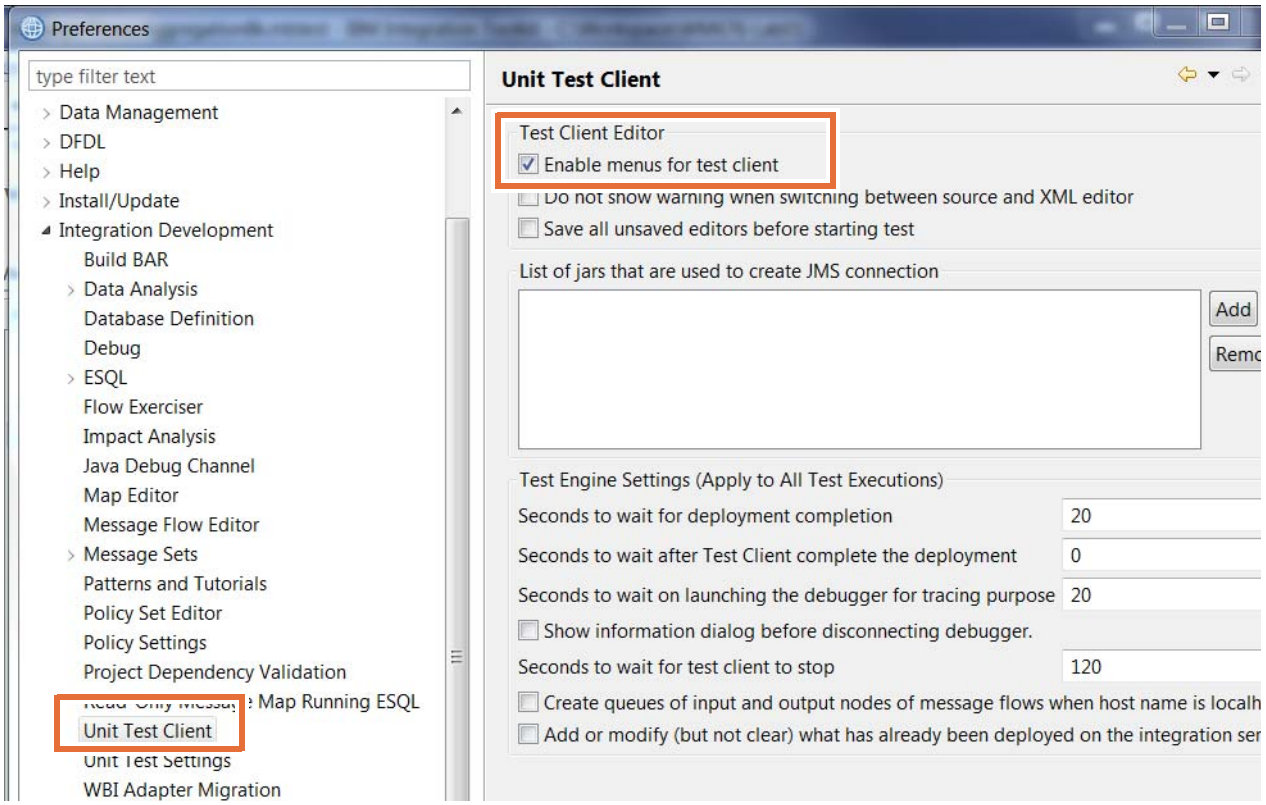
In this part of the exercise, you deploy and test the message flow by using the Integration Toolkit Unit Test Client.

The application includes a simple message flow that is named **RequestReplyApp.msgflow** that simulates the back-end application.

__ 1. Enable the Integration Unit Toolkit Test Client in the Integration Toolkit.

__ a. Click **Window > Preferences**.

__ b. Expand **Integration Development** and then click **Unit Test Client**.



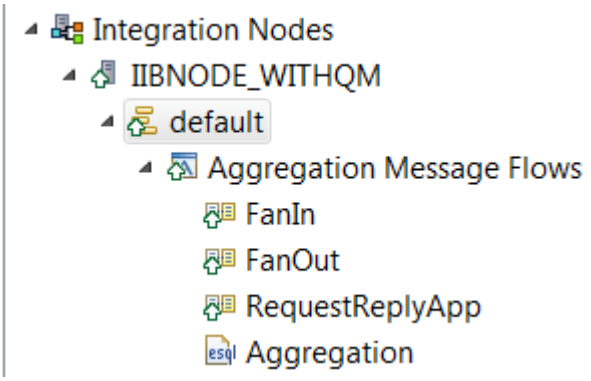
__ c. On the **Unit Test Client** page, click **Enable menus for test client**.

__ d. Click **OK**.

__ 2. Deploy the **Aggregation Message Flow** application to the integration server that is named **default** on the IIBNODE_WITHQM integration node.

Drag the **Aggregation Message Flows** application from the **Application Development** view and drop it on the integration server that is named **default** under the IIBNODE_WITHQM integration node in the **Integration Nodes** view.

__ 3. Verify that the deployment was successful by checking the **Deployment Log** and the **Integration Nodes** view.



__ 4. Open the Unit Test Client with the preconfigured test file.

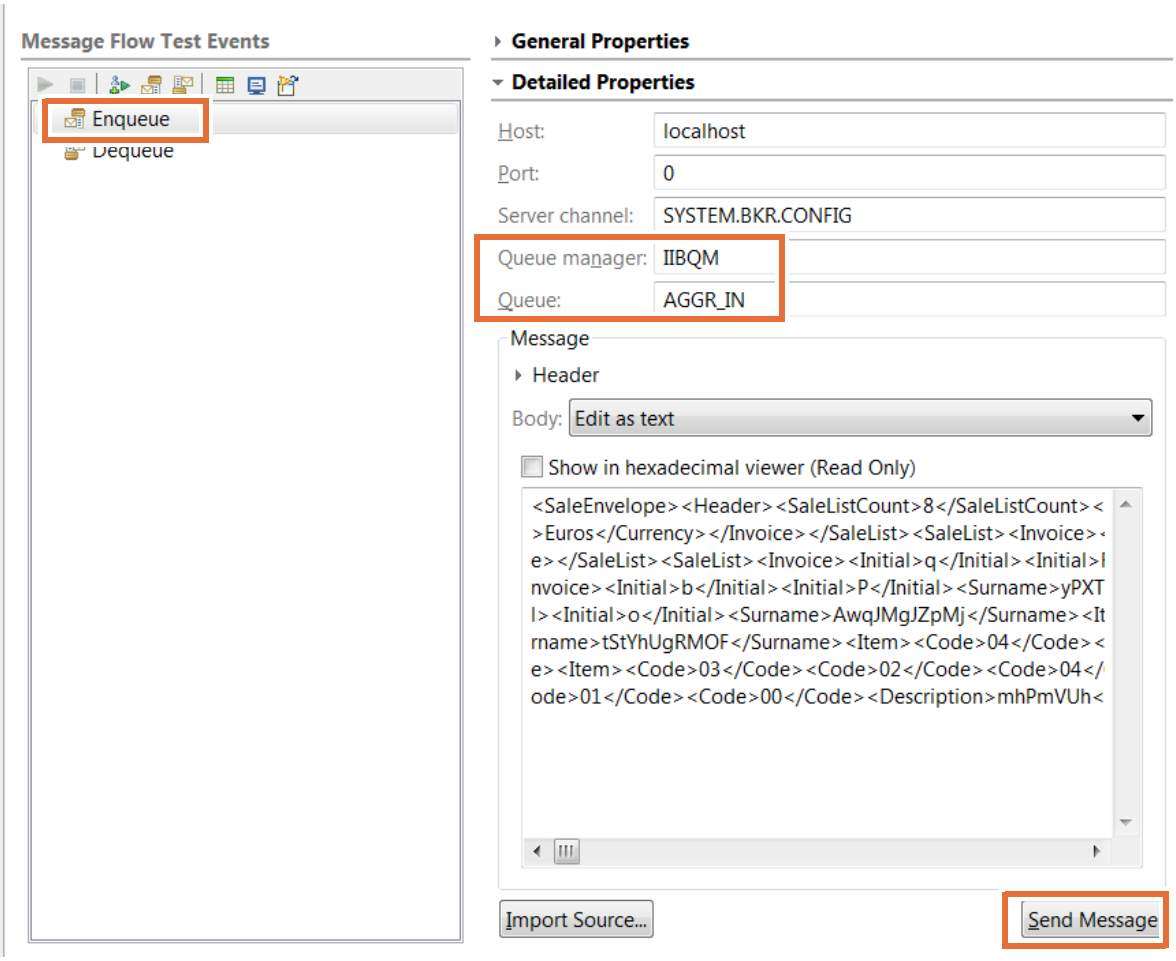
__ a. Expand **Aggregation Message Flows > Flow Tests** in the **Application Development** view.

__ b. Double-click the **Aggregation8k.mbtest** file. The Test Client view opens.

__ 5. Two message flow events are already configured in the test file:

- **Enqueue** puts the message on the queue that is specified under the **Detailed Properties** section (AGGR_IN).
- **Dequeue** gets the message from the queue that is specified under the **Detailed Properties** section (AGGR_OUT).

__ a. Click **Enqueue** and verify that the queue name is set to AGGR_IN in the Unit Test Client **Detailed Properties** and that the <SalesEnvelope> message is loaded in the **Message** section.

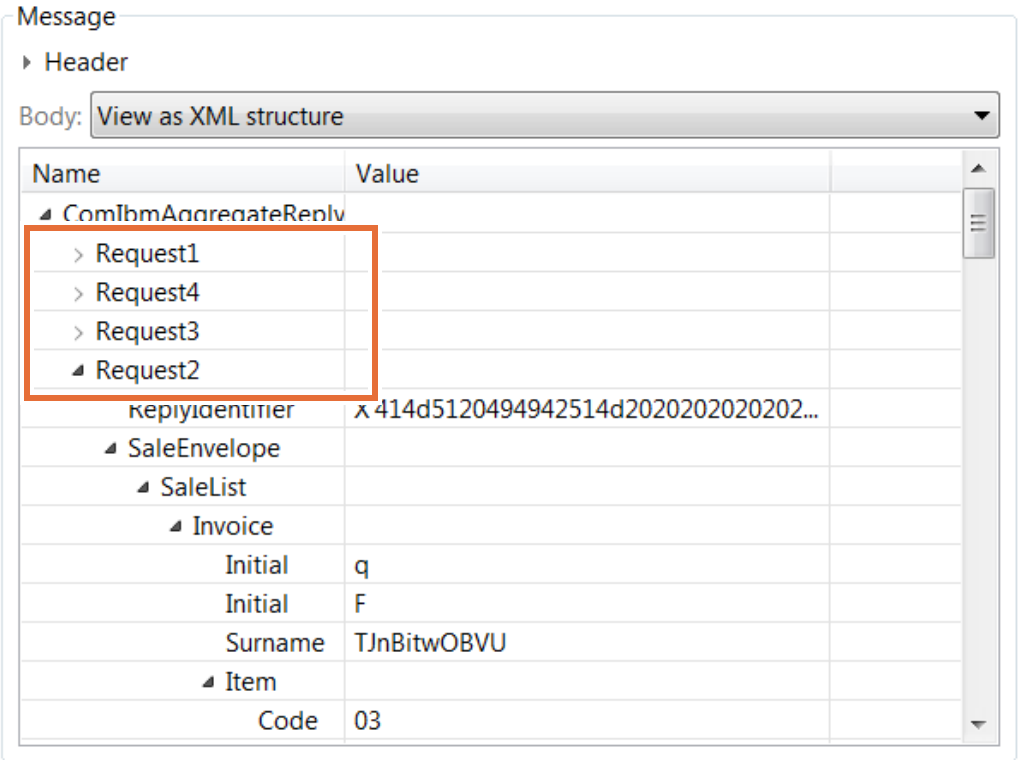


__ b. Click **Send Message**. The Test Client should indicate that the message was sent to AGGR_IN.

__ c. Click **Dequeue** in the message flow test events and verify that queue name is set to AGGR_OUT.

__ d. Click **Get Message**. The aggregated message should be displayed under the message section.

__ e. Scroll down through the message to verify that the output message contains four replies in four folders: Request1, Request2, Request3, and Request4.



Note

Your messages might appear in a different order than is shown in the screen capture.

If the message did not get created, open the Windows Event Viewer **Application** log and check for error messages.

Exercise cleanup

__ 1. Close all open editor windows in the Integration Toolkit. You do not need to save any changes, in case you are prompted.

__ 2. In the **Integration nodes** view:

__ a. Stop the message flow application.

__ b. Right-click the **default** integration server in the IIBNODE_WITHQM integration node and then select **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.

End of exercise

Exercise review and wrap-up

In the first part of the exercise, you used IBM MQ Explorer to create a queue manager that is named IIBQM. You then used the Integration Toolkit to create an integration node that is named IINODE_WITHQM that is associated with the queue manager IIBQM. Finally, you used an Integration Bus command file to add the SYSTEM.BROKER queues and topics to the IIBQM queue manager.

In the second part of the exercise, you configured the fan-out message flow to generate four different request messages and start the tracking of the aggregation operation.

In the third part of this exercise, you configured the aggregation node for the fan-in message flow.

In the fourth part of this exercise, you deployed the application and then tested aggregation by using the Integration Toolkit Unit Test Client.

Having completed this exercise, you should be able to:

- Add IBM Integration Bus SYSTEM.BROKER queues to a queue manager
- Use the AggregateControl node and the AggregateRequest node to generate and concurrently fan-out related requests
- Use the AggregateReply node to aggregate messages into a single output message

.

Exercise 2. Extending a DFDL model

What this exercise is about

In this exercise, you model complex data in DFDL. Complex data includes data with multiple records types, choice groups, and optional components. You also learn how to use discriminators in a DFDL model to optimize the parsing of data.

What you should be able to do

After completing this exercise, you should be able to:

- Extend a DFDL model to add header and trailer records
- Reference a DFDL model in a new DFDL model
- Use discriminators in a DFDL model so that the parser can conditionally process elements as determined by the content of other elements in the data and optimize the parsing of data

Introduction

You can reference a DFDL message model in another DFDL message model.

In the first part of this lab, you extend a DFDL message model to include a header record and a trailer record.

The DFDL standard also provides a mechanism to allow a parser to make parsing decisions that are based on the content of other elements in a message. In this way, the structure and description of a message can be changed, and parsing of data can be optimized.

The DFDL parser is a recursive-descent parser with look-ahead. It can resolve “points of uncertainty”, such as a choice, an optional element, or a variable array of elements. The DFDL parser must speculatively attempt to parse data until an object is either “known to exist” or “known not to exist”. Until that applies, the occurrence of a processing error causes the parser to suppress the error, backtrack, and make another attempt. The use of discriminators can be used to assert that an object is “known to exist”, which prevents incorrect backtracking.

In the second part of this lab, you add discriminators to an existing DFDL model.

Requirements

- IBM Integration Toolkit V10
- Lab exercise files in `C:\labfiles\Lab02-DFDL`

Exercise instructions

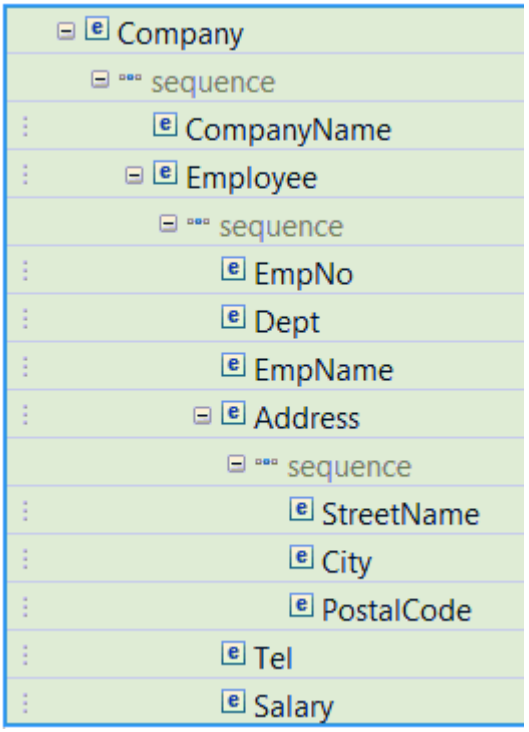
Part 1: Extending the message model

In this part of the lab, you extend a basic DFDL message model to add a header and trailer record.

The starting message model defines the `Company.txt` file (in the `C:\labfiles\Lab02-DFDL\data` directory):

```
Company[compName=My Company
Employee(empNum=111111|dept=500|empName=Alice Wong|Addr:8200 Warden Ave,"Markham, Ont",L3G
1H7|tel=905-347-5649|sal=135599.95)
Employee(empNum=222222|dept=500|empName=James May|Addr:23 The Cuttings,Chatham,CH2
2PR|tel=208-203-1332|sal=189599.95)
Employee(empNum=333333|dept=310|empName=Richard Hammond|Addr:16 Great Windmill,London,W2
3RJ|tel=207-445-2955|sal=599.95)
Employee(empNum=444444|dept=230|empName=Jeremy Clarkeson|Addr:"Rose Cottage, Pea
Dr",Gloucester,GL01 2NM|tel=743-123-4567|sal=75599.95)
Employee(empNum=555555|dept=650|empName=Humphrey Littleton|Addr:416 Regent
Street,London,NW1 1QT|tel=207-883-1238|sal=99999.95)
]
```

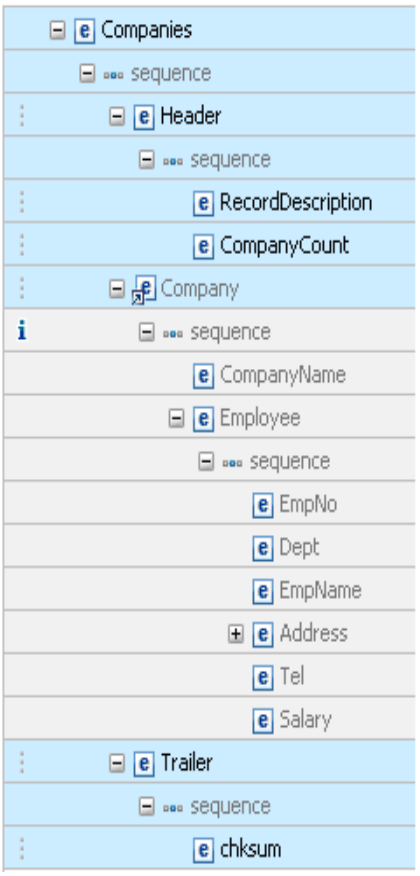
The DFDL model for the `Company.txt` file is defined in the `Company.xsd` schema.



In this exercise, you extend this DFDL structure so that it can parse the `Companies.txt` data file (in the `C:\labfiles\Lab02-DFDL\data` directory), which contains multiple Company records with a header and a trailer record:

```
Header{recDesc:My Company records,compCount:5}
Company[compName=BBC
Employee(empNum=111111|dept=500|empName=Alice Wong|Addr:8200 Warden Ave,"Markham, Ont",L3G
1H7|tel=905-347-5649|sal=135599.95)
Employee(empNum=222222|dept=500|empName=James May|Addr:23 The Cuttings,Chatham, CH2
2PR|tel=208-203-1332|sal=6189599.95)
Employee(empNum=333333|dept=310|empName=Richard Hammond|Addr:16 Great Windmill St,London,W2
3RJ|tel=207-445-2955|sal=599.95)
Employee(empNum=444444|dept=230|empName=Jeremy Clarkeson|Addr:"Rose Cottage, Pea
Dr",Gloucester,GL01 2NM|tel=743-123-4567|sal=5599.95)
Employee(empNum=555555|dept=650|empName=Humphrey Littleton|Addr:416 Regent
Street,London,NW1 1QT|tel=207-883-1238|sal=99999.95)
]
Company[compName=IBM
Employee(empNum=111111|dept=9876|empName=Arnold Buzby|Addr:1000 The Close,Winchester,L3G
1H7|tel=905-345-5649|sal=23.54)
Employee(empNum=222222|dept=2350|empName=Digby Jones|Addr:1 Porstmouth Rd,Southampton,CH2
2PR|tel=208-203-1332|sal=599.95)
]
Trailer{chksum:1234567890}
```

The header record contains a record description and a company count. The trailer record contains a checksum. You must modify the existing DFDL message structure to add the header and trailer records.

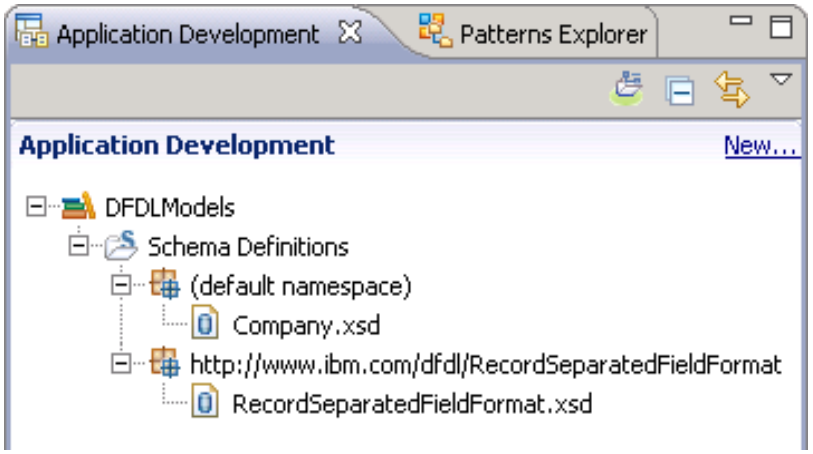


__ 1. Start the IBM Integration Toolkit (if it is not already running) and switch to a new workspace.

__ 2. Import the Project Interchange file that contains the **DFDLModels** library.

The Project Interchange file is `DFDLLab_Startingpoint.zip` and is in the `C:\labfiles\Lab02-DFDL\resources\` directory.

The imported library contains the message model `Company.xsd` that defines the `Company.txt` file and the reference schema `RecordSeparatedFieldFormat.xsd`.



__ 3. Use the New Message Model wizard to create a record-oriented message model schema that is named **Companies** in the **DFDLModels** library that describes the header and trailer records.

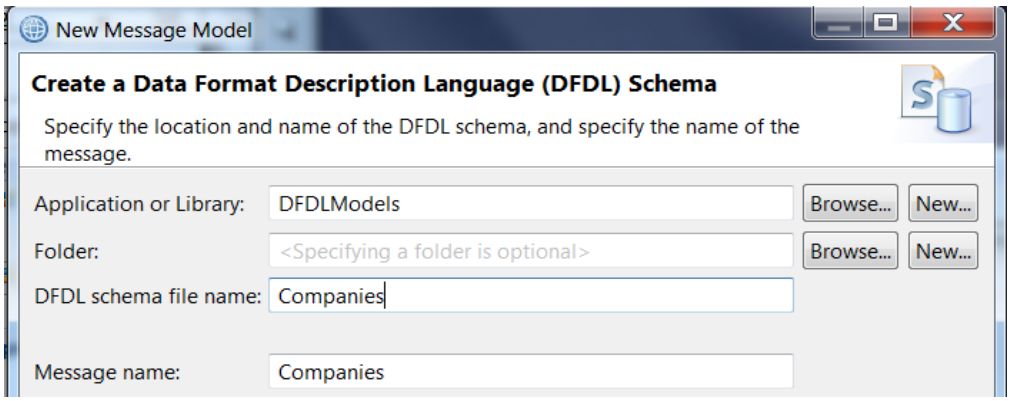
__ a. In the **Application Development** view, click **New > Message Model**.

__ b. In the New Message Model wizard, click **Record-oriented text** and then click **Next**.

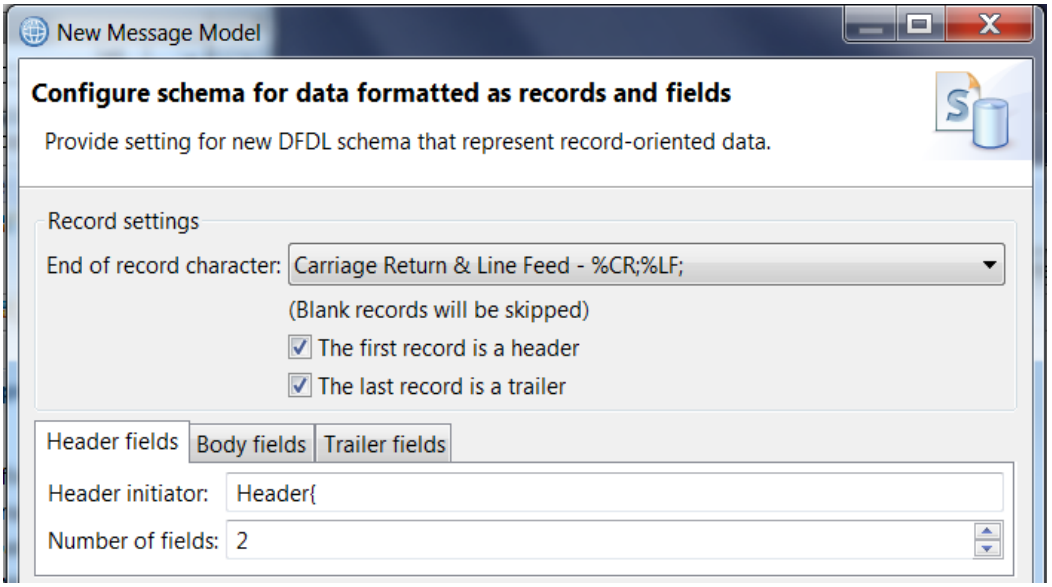
__ c. From the wizard, click **Create a DFDL schema file using the wizard to guide you** and then click **Next**.

__ d. Click **Browse** and select **DFDLModels** for the **Application or Library.**

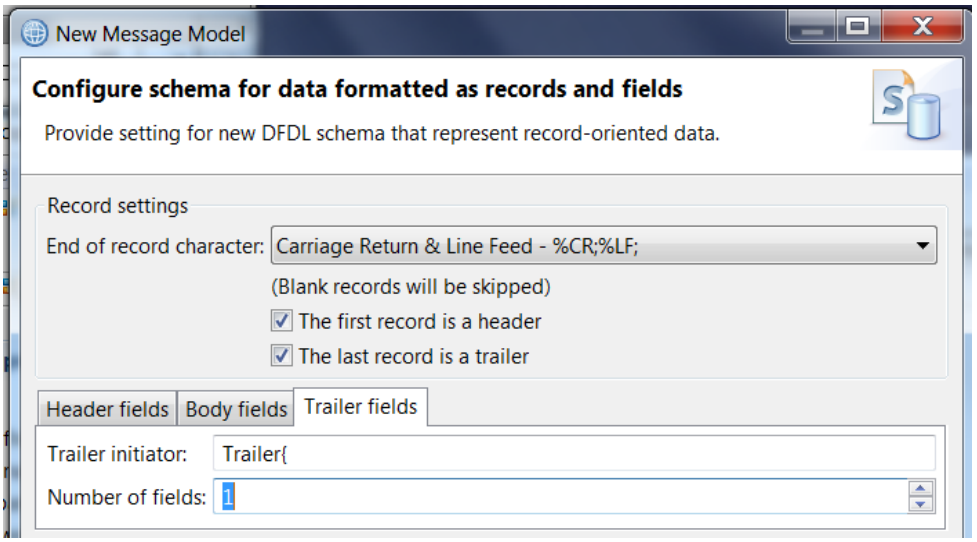
__ e. Enter `Companies` for the **DFDL schema file name** and then click **Next**.



__ f. In the **Header fields** tab, type `Header{` for the **Header initiator** and 2 for the **Number of fields**.



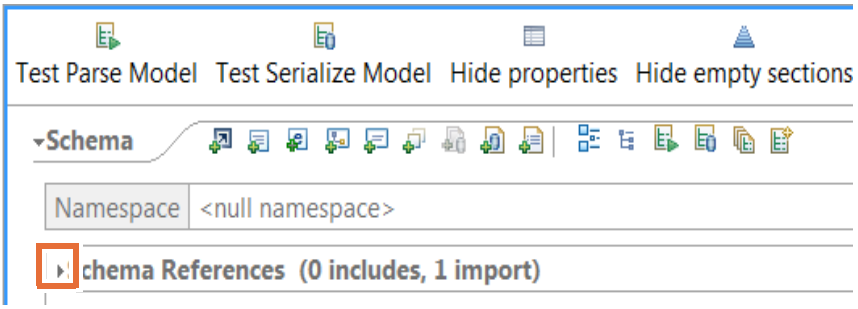
__ g. On the **Trailer fields** tab, type `Trailer{` for the **Trailer initiator** and `1` for the **Number of fields**.



__ h. Click **Finish**. The DFDL Schema editor opens with the generated DFDL schema and the schema file `Companies.xsd` is added to the **DFDLModels** library.

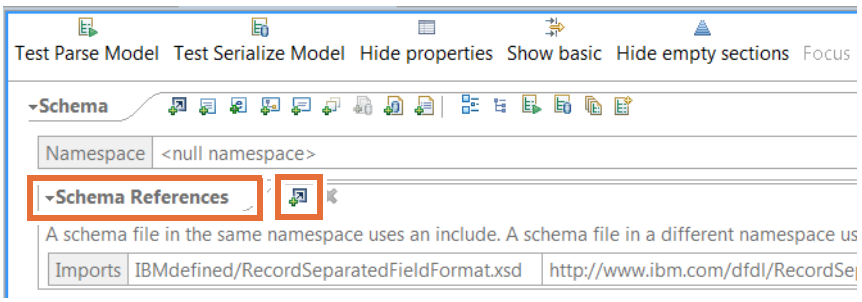
__ 4. Click the **Show all sections** icon in the DFDL Schema editor toolbar.

__ 5. Click the arrow next to the **Schema References** section to expand the references of the DFDL schema file.



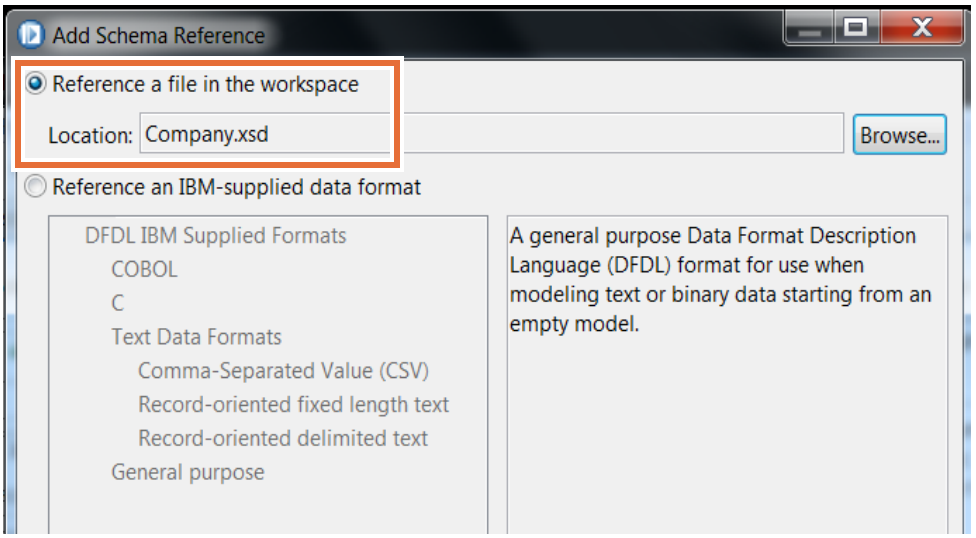
__ 6. The `Companies.xsd` message model builds on the `Company.xsd` message model by creating a reference to that model.

__ a. Click the **Add a reference to another schema** icon in the **Schema References** section.



__ b. In the **Add Schema Reference** window, verify that the **Reference a file in the workspace** option is selected and then click **Browse**.

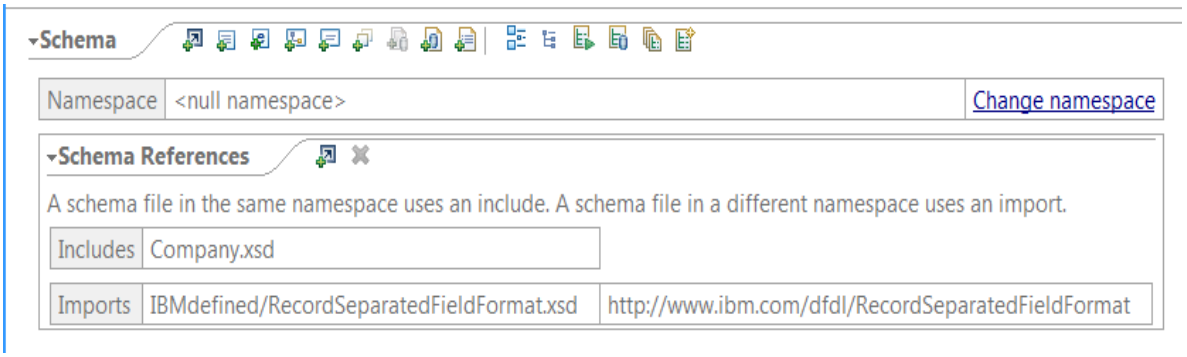
__ c. Select the **`Company.xsd`** schema from the **DFDLModels** library and then click **OK**.



__ d. Click **OK** in the Add Schema Reference window. You now have two **Schema References**:

- A reference to the `Company.xsd` schema, which you just added. You are going to reuse this message model to create a more complex one.

- A reference to the `RecordSeparatedFieldFormat.xsd` schema, which the wizard automatically adds. It contains Record Separated specific defaults for DFDL properties.



__ 7. By default, the **Companies > sequence** element has the **Initiated Content** property set to yes. In this exercise, the `Companies.txt` file does not have an initiator for the file. The record initiators are defined at the element (record) level for each record type.

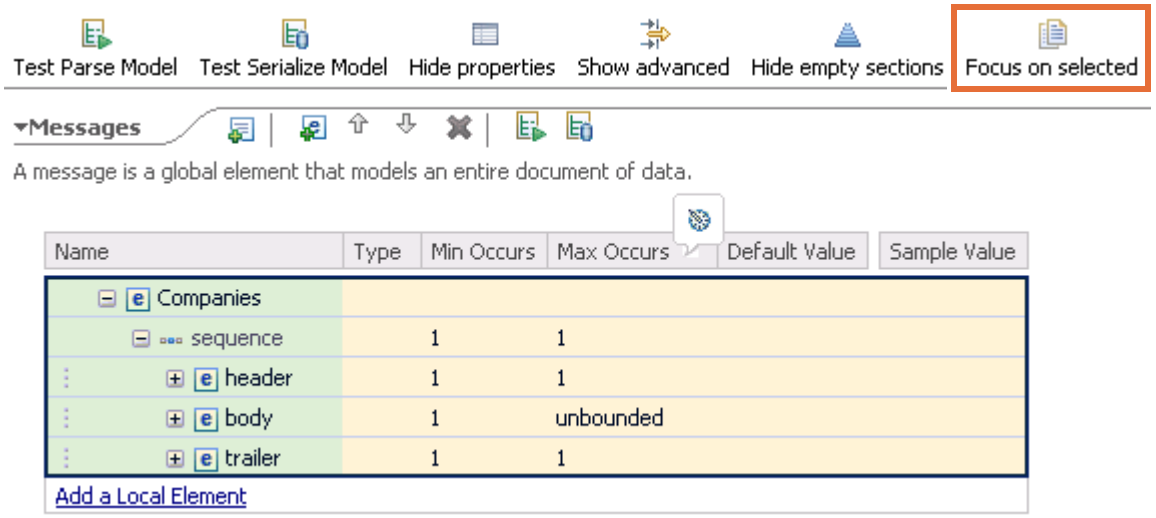
Click the **sequence** element under the **Companies** element in the **Message** section to select it. On the **Representation Properties** tab, change the **Initiated Content** property to no.

__ 8. The header record in the `Companies.txt` file has the following format:

```
Header{recDesc:My Company records,compCount:5}
```

In this step, you modify the **Companies** schema **header** record to define the **RecordDescription** and **CompanyCount** fields.

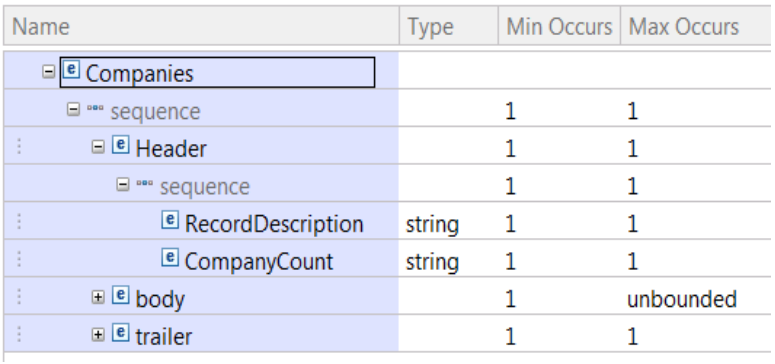
__ a. Click anywhere inside the **Messages** section, and then click the **Focus on selected** icon.



__ b. Expand the **header** element.

__ c. Change the **header** element name to: `Header`

__ d. Change the names of the elements under **Header** to: `RecordDescription` and `CompanyCount`

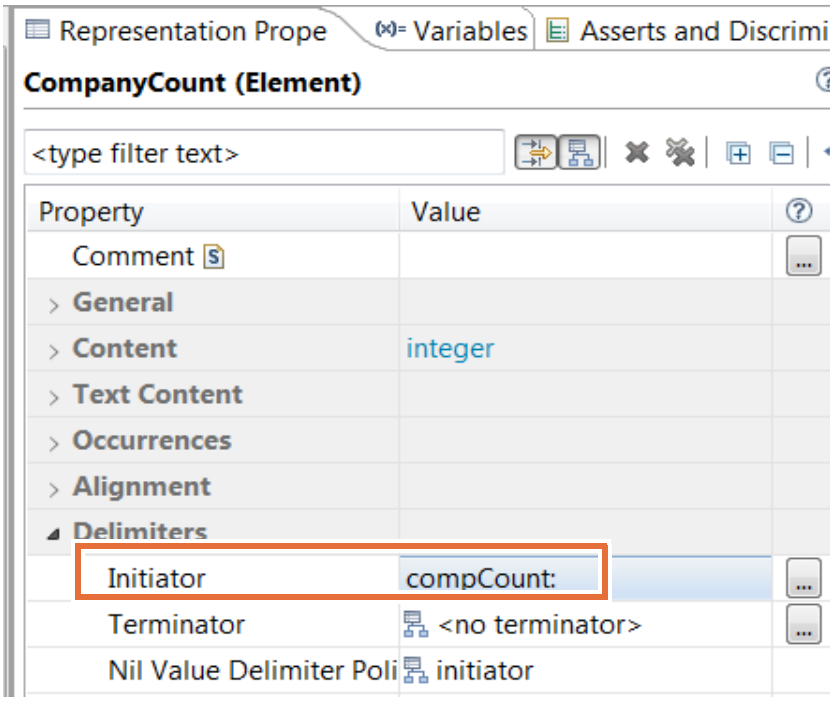


Name	Type	Min Occurs	Max Occurs
Companies			
sequence		1	1
Header		1	1
sequence		1	1
RecordDescription	string	1	1
CompanyCount	string	1	1
body		1	unbounded
trailer		1	1

__ e. Change the **CompanyCount** element type to integer by clicking its **Type** column and then selecting **integer**.

__ f. In the **Representation Properties** tab, change the **Initiator** value (under the **Delimiters** section) for the **CompanyCount** element from `iHead2` to `compCount:`

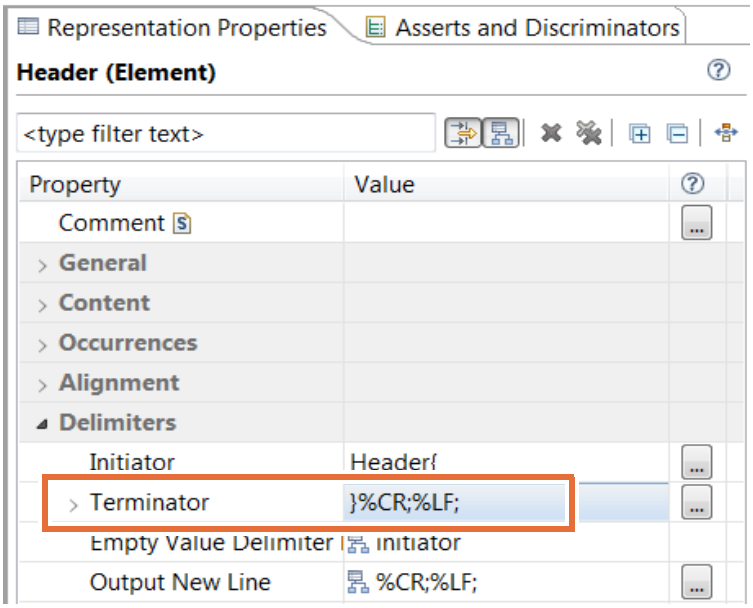
Be sure to enter the colon after the `compCount` string.



__ g. Change the **Initiator** value for the **RecordDescription** element to `recDesc:`

__ 9. Define the terminator for the **Header** element.

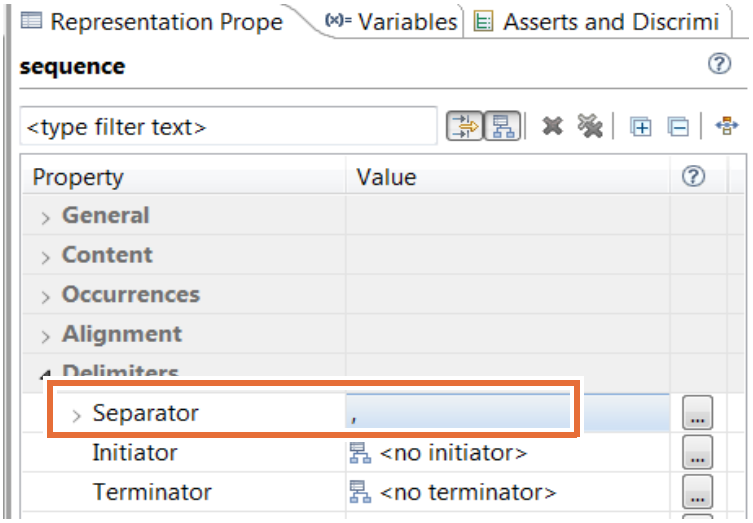
In the **Delimiters** section of the **Representation Properties** tab for the **Header** element, set the **Terminator** property to `}%CR;%LF;`



__ 10. A comma character delimits the fields in the **Header** record. Define the delimiter for the **Header** record.

__ a. Select the **sequence** element under the **Header** element.

__ b. In the **Delimiters** section of the **Representation Properties** tab, set the **Separator** property to: `,` (comma).



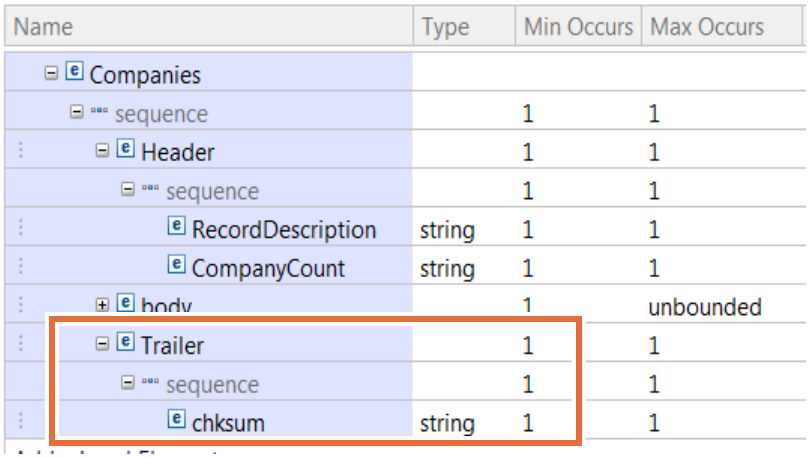
__ 11. The trailer record in the `Companies.txt` file contains a checksum. An example of the trailer record is shown here:

```
Trailer{chksum:1234567890}
```

Modify the **trailer** element and its subelement to define the trailer record.

__ a. Change the **trailer** element name to: `Trailer`

__ b. Change the element under **Trailer** to: chksum



__ c. Change the **chksum** element **Initiator** value from `iTrail1` to chksum:

__ d. Define the terminator for **Trailer** record.

In the **Delimiters** section of the **Representation Properties** tab for the **Trailer** element, set the **Terminator** property to }

__ 12. The Trailer record has one field, so no separator is required.

__ a. Select the **sequence** element under the **Trailer** element.

__ b. In the **Delimiters** section of the **Representation Properties**, delete the **Separator** property value. Press **Enter** to make sure that the value is updated to <no separator>. The wizard automatically added the separator, but it is not required for this model.

__ 13. The wizard automatically adds a separator to the **sequence** element under the **Companies** element. The format of this data does not require a separator for this element.

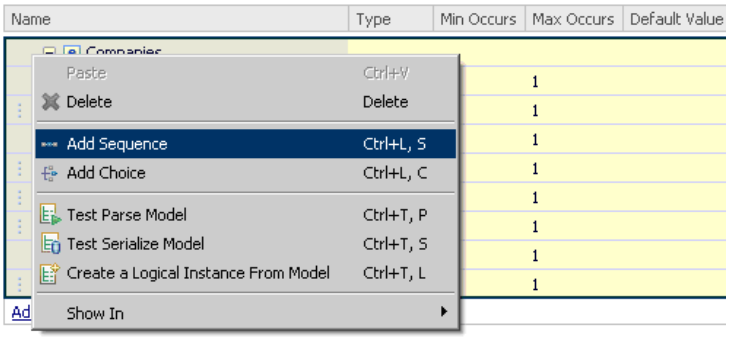
Select the **sequence** element under the **Companies** element.

In the **Delimiters** section of the **Representation Properties,** delete the value of the **Separator** property. Press **Enter** to make sure that the value is updated to <no separator>.

__ 14. Replace the **body** portion of the new schema with a reference to the `Company.xsd` model.

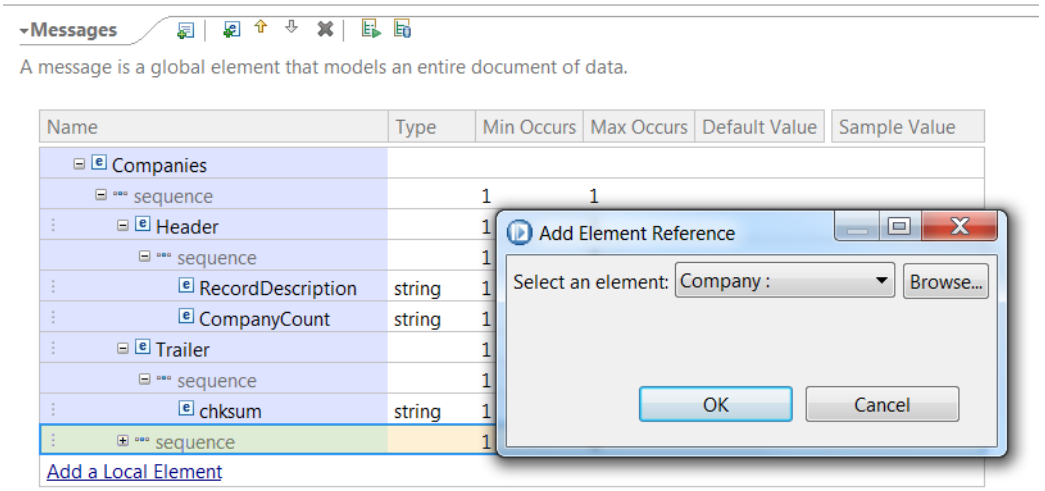
__ a. Delete the **body** element of the schema by right-clicking the **body** element line and then selecting **Delete**. Do not right-click the text of the element name; you see a different menu.

__ b. Right-click the **Companies** element and then select **Add Sequence**. A new sequence element is added to the bottom of the schema.



__ c. Right-click the new **sequence** element and then click **Add Element Reference**.

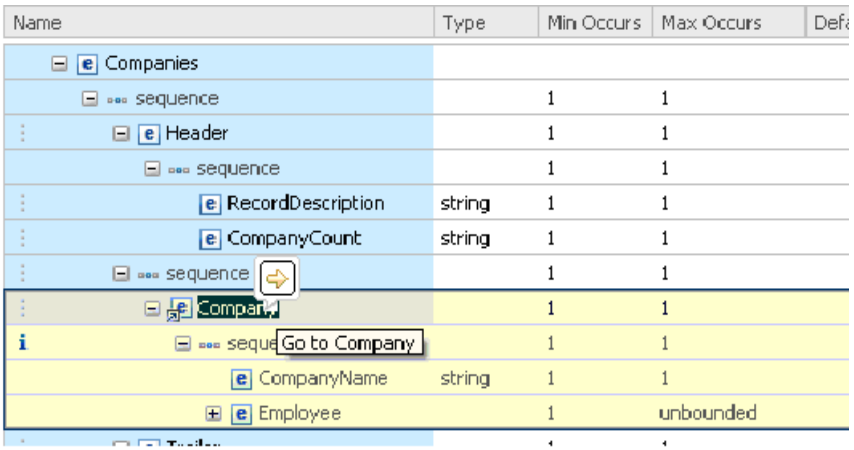
__ d. Select **Company:** from the list and then click **OK**.



Note

The added element reference has a different icon to differentiate it from a regular element; it is only a reference to an existing element in other DFDL schema.

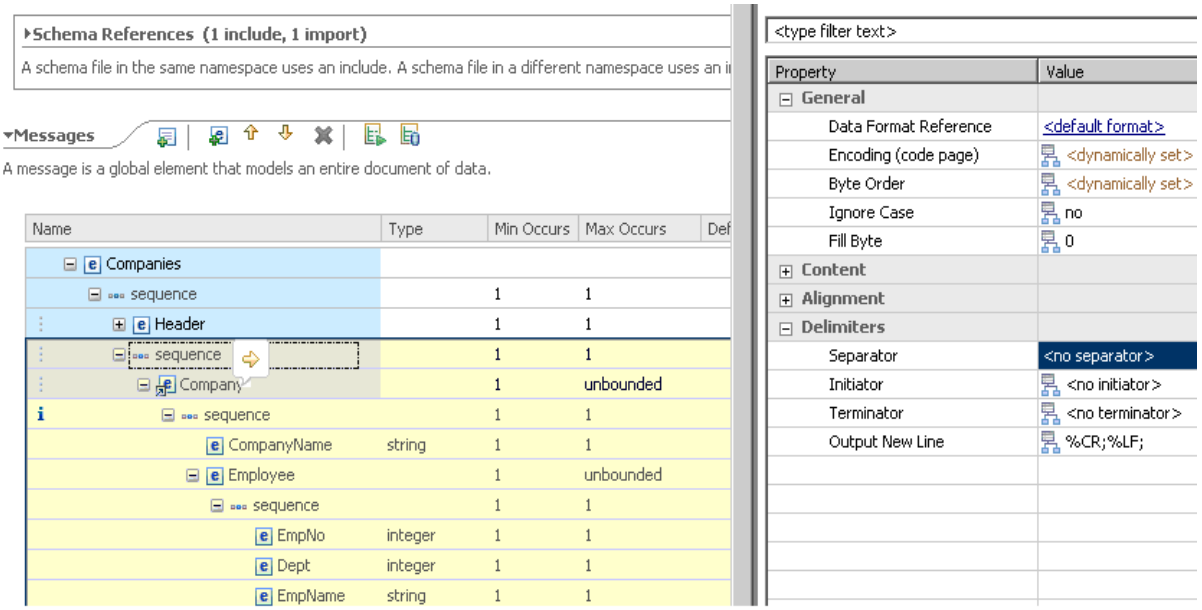
Also, the element name is not available because it is read-only. To modify it, you must open the referenced DFDL schema where the element is defined by clicking the yellow arrow that is displayed when you hover the cursor over the element.



__ e. Right-click the newly created **sequence** element and then select **Move Up** so that it is moved above the **Trailer** element.

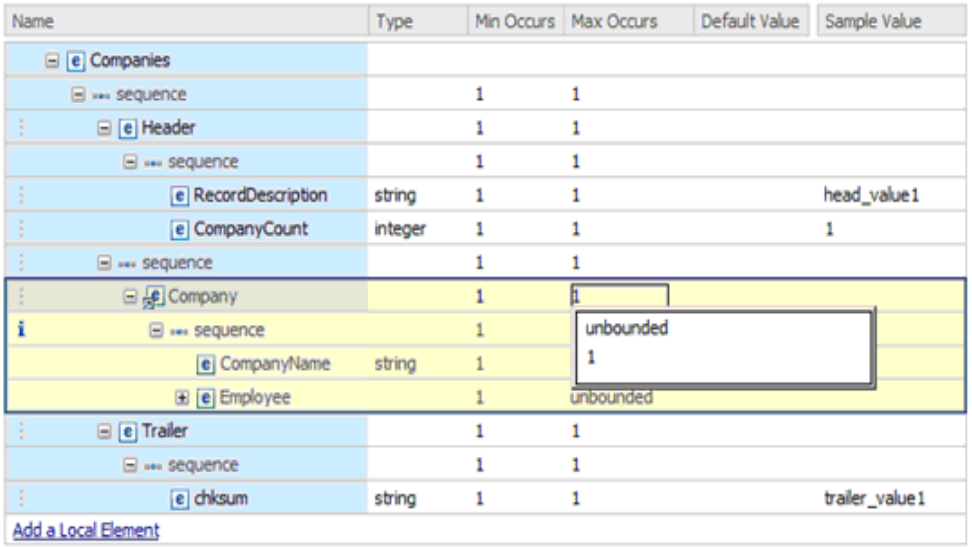
__ 15. Modify the **Delimiters > Separator** property for the new sequence.

Select the new **sequence** element and remove the default value for the **Separator** property. Press Enter to ensure that the existing value is deleted and that the value is set to <no separator>.



__ 16. Select the **Max Occurs** column of the **Company** element reference, and change it to unbounded.

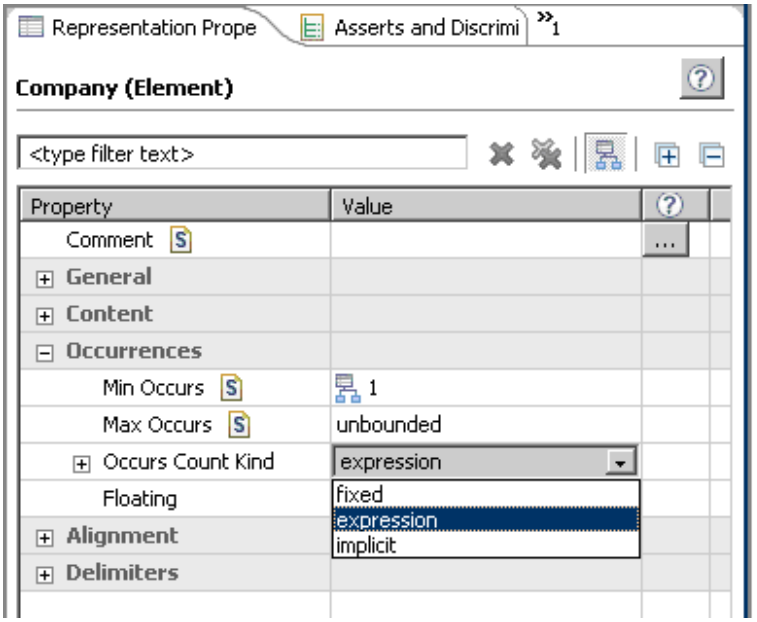
This change allows the **Company** element reference to have unlimited occurrences.



__ 17. Modify the schema so that the **CompanyCount** element in the **Header** dictates the number of occurrences of the **Company** element.

__ a. Select the **Company** element reference.

__ b. In the **Occurrences** section of the **Representation Properties** view, change the **Occurs Count Kind** from `fixed` to `expression`.

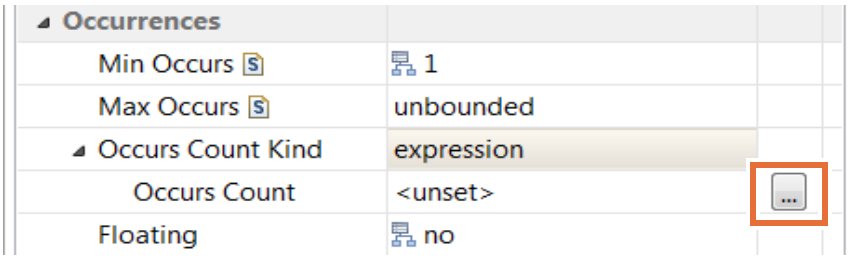


__ c. Expand the **Occurs Count Kind** property to show the **Occurs Count** property.

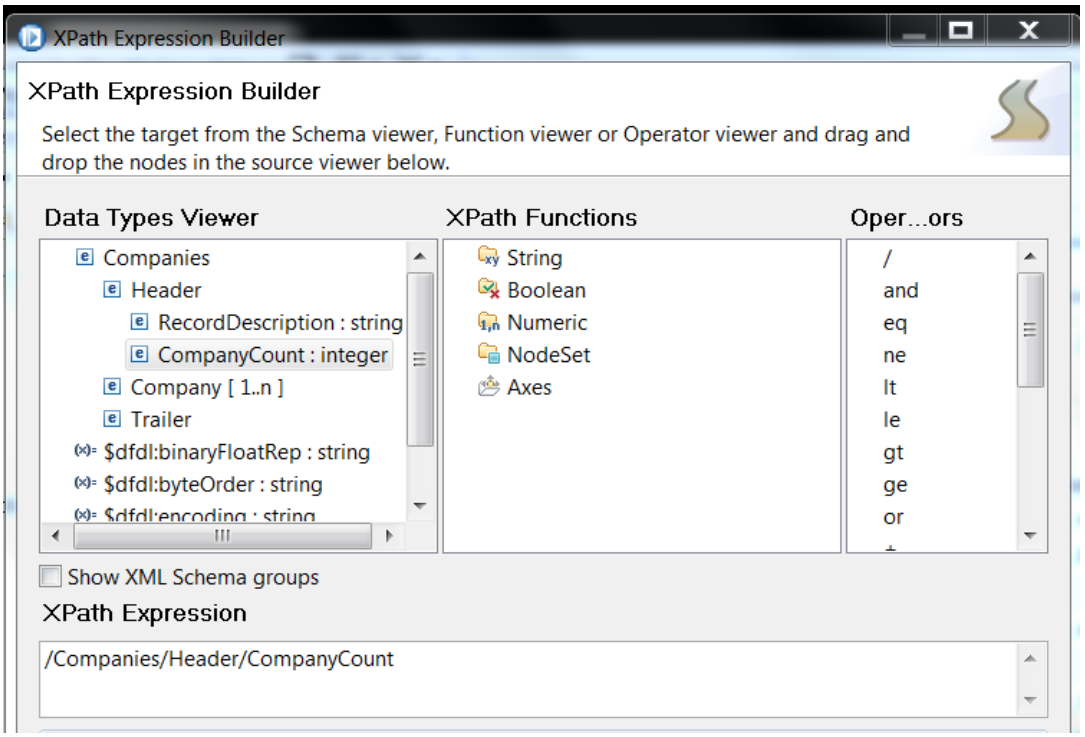
Note

It might be necessary to save the model and close and reopen the DFDL Schema editor to update the Representation Properties so that you can expand the **Occurs Count Kind** property.

__ d. Click the ellipsis next to the **Occurs Count** property to open the XPath Expression Builder.



__ e. In the XPath Expression Builder, expand the **Companies > Header** element and then double-click the **CompanyCount** element.



__ f. Click **Finish**.

__ g. **Save** the changes in the editor.

__ h. Confirm that the Companies element contains no errors.

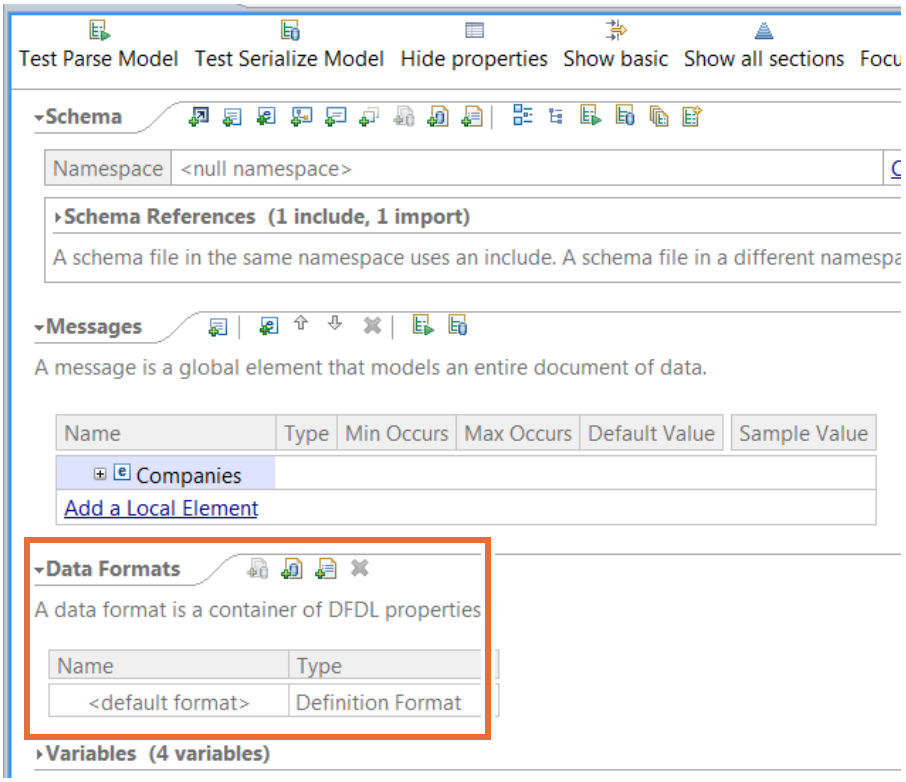
__ 18. The `Companies.txt` input file might have an extra carriage return and line feed at the end of the file. In some cases, the last record might be missing the final “new line” character.

To handle this situation, change the default values for this model so that it can parse successfully, irrespective of whether the final character is present or not.

__ a. In the DFDL Schema editor, collapse the **Companies** model, and then click **Show all sections** in the DFDL Schema editor toolbar.

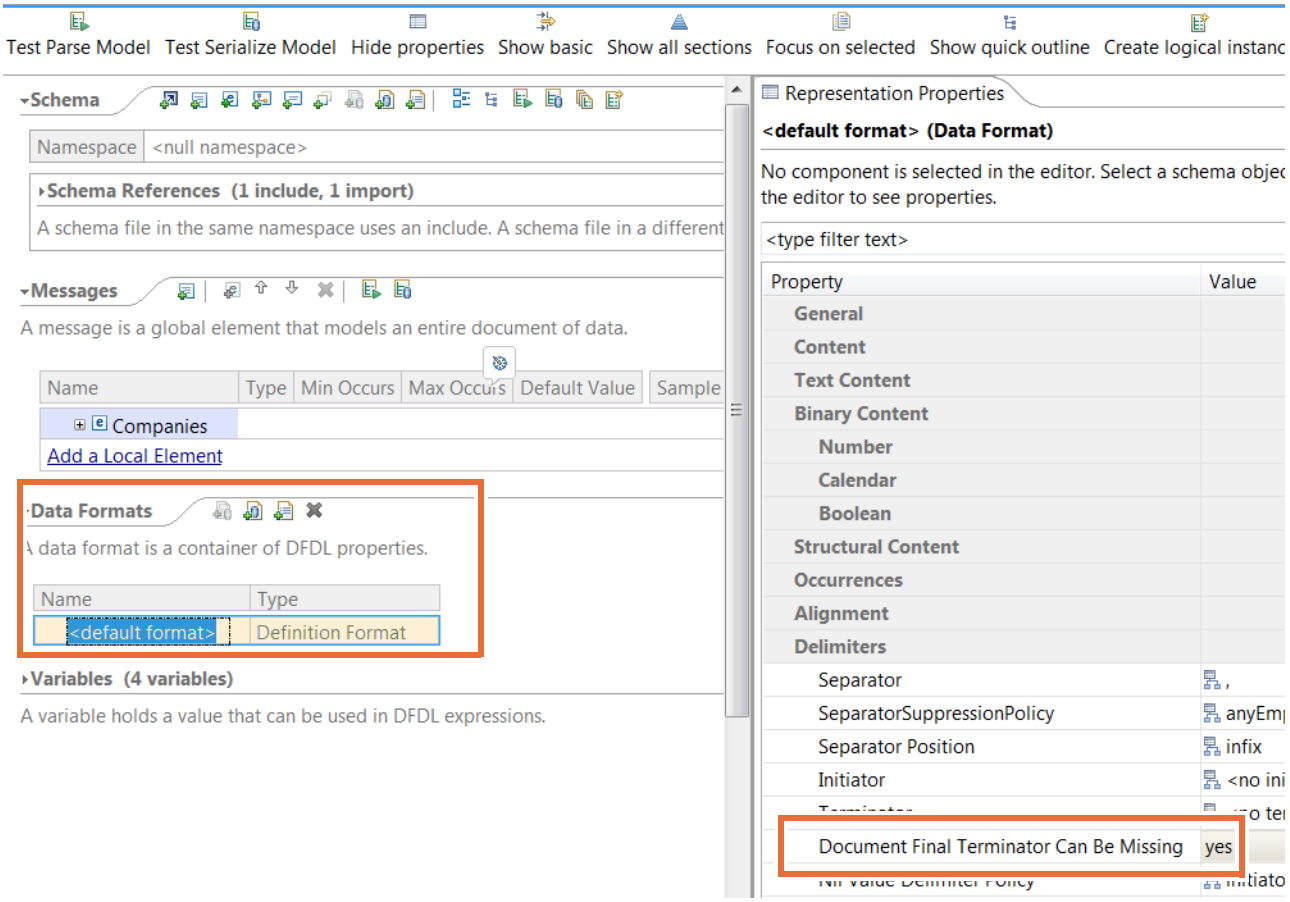
__ b. To show the minimum required content, click **Hide empty sections** in the DFDL Schema editor toolbar.

__ c. Expand the **Data Formats** section.



__ d. Highlight the **<default format>** field under the **Data Formats** section. This section is where you can define many default values for the message model.

__ e. In the **Representation Properties**, expand the **Delimiters** section, and locate the property **Document Final Terminator Can Be Missing**. Set this property to yes.



__ 19. Save the schema.

Check the **Problems** view and verify that it contains no errors.

__ 20. Test the model by parsing the `Companies.txt` file.

__ a. Click the **Test Parse Model** icon.

__ b. In the Parser Input section, select **Content from a data file** and click **Browse**.

__ c. Check **Select an input file from the file system**.

__ d. Go to `C:\labfiles\Lab02-DFDL\data`, select the `Companies.txt` file, and then click **Open**.

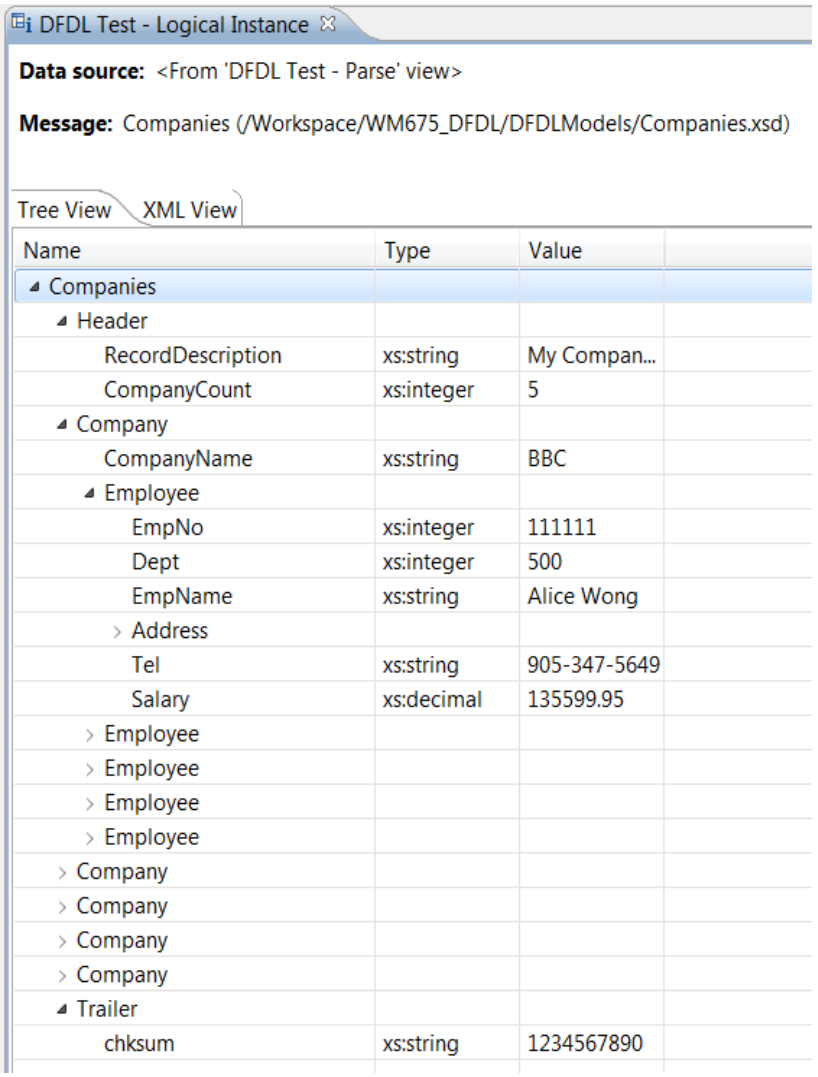
__ e. Click **OK** on the **File Selection** window.

__ f. Click **OK** on the Test Parse Model window.

__ g. Click **Yes** to switch to the DFDL Test perspective.

A message with *Parsing completed successfully* should be displayed.

__ h. Review the Parsed input and the **DFDL – Logical Instance** view to verify that the parsing is correct.



If you have errors, review the parsing errors and the **Trace** tab to identify the problem.

__ 21. When the data parses correctly, return to the **Integration Development** perspective.

__ 22. Close the DFDL Schema editor.

Part 2: Using discriminators to resolve choices

The DFDL parser is a recursive-descent parser with look-ahead used to resolve “points of uncertainty”, such as:

- A choice
- An optional element
- A variable array of elements

The DFDL parser must speculatively attempt to parse data until an object is either “known to exist” or “known not to exist”. Until that applies, the occurrence of a processing error causes the parser to suppress the error, backtrack, and make another attempt.

Discriminators can be used to assert that an object is “known to exist”, which prevents incorrect backtracking. This part of the lab provides a simple example in the use of discriminators.

This part of the lab uses a COBOL copybook that is named `PURCHASES-DISC.cpy` in the `C:\labfiles\Lab02-DFDL\disciminators` directory.

```
01  PURCHASES-DISC.
           03  REQUEST-TYPE                PIC X.
           03  RET-CODE                    PIC XX.
           03  CustomerId                  PIC X(8).
           03  CustomerLastName            PIC X(20).
           03  CustomerFirstName           PIC X(20).
           03  CustomerCompany             PIC X(30).
           03  CustomerAddr1               PIC X(30).
           03  CustomerAddr2               PIC X(30).
           03  CustomerCity                PIC X(20).
           03  CustomerCountry             PIC X(30).
           03  CustomerArea                PIC X(20).
            03  CustomerProvince REDEFINES CustomerArea PIC X(20).
            03  CustomerCounty   REDEFINES CustomerArea PIC X(20).
            03  CustomerRegion   REDEFINES CustomerArea PIC 9(20).
            03  CustomerState    REDEFINES CustomerArea PIC X(20).
           03  CustomerMailCode            PIC X(20).
           03  CustomerPhone               PIC X(20).
           03  CustomerLastUpdateDate      PIC X(8).
           03  PurchaseCount               PIC 9(3) USAGE COMP.
           03  Purchase OCCURS 0 TO 99 TIMES
               DEPENDING ON PurchaseCount.
             04  PurchaseId                PIC 9(5).
             04  ProductName               PIC X(30).
             04  Amount                    PIC 9(2).
             04  Price                     PIC 9(8)V99.
           03  RETURN-COMMENT              PIC X(50).
```

The COBOL copybook contains the REDEFINES clause. The REDEFINES clause enables a single element to contain different types of data (for example, character or binary). It also enables a receiving application to process the element differently, depending on the type of data that is contained in the element.

The REDEFINES clause in the COBOL copybook for this exercise redefines the base element **CustomerArea**.

- The **CustomerProvince** redefines clause is used when `CustomerCountry = 'Canada'.`
- The **CustomerCounty** redefines clause is used when `CustomerCountry = 'UK' or 'Ireland'`.
- The **CustomerRegion** redefines clause is used when `CustomerCountry = 'Russia'`.

  Unlike the other fields, **CustomerRegion** is defined as a numeric value, even though the base element (**CustomerState**) is PIC X (character).
- **CustomerState** is used when `CustomerCountry = 'USA'`
- **CustomerArea** is used by any other country.

Several data files use this copybook. Each data file contains a single record, with data corresponding to the definitions described previously.

__ 1. Create a DFDL message model by importing the COBOL copybook that is named `PURCHASES-DISC.cpy` in the `C:\labfiles\Lab02-DFDL\discriminators` directory.

__ a. In the **DFDLModel**s library that you created in Part 1 of this lab, select **New > Message Model**.

__ b. In the New Message Model wizard, click **COBOL** and then click **Next**.

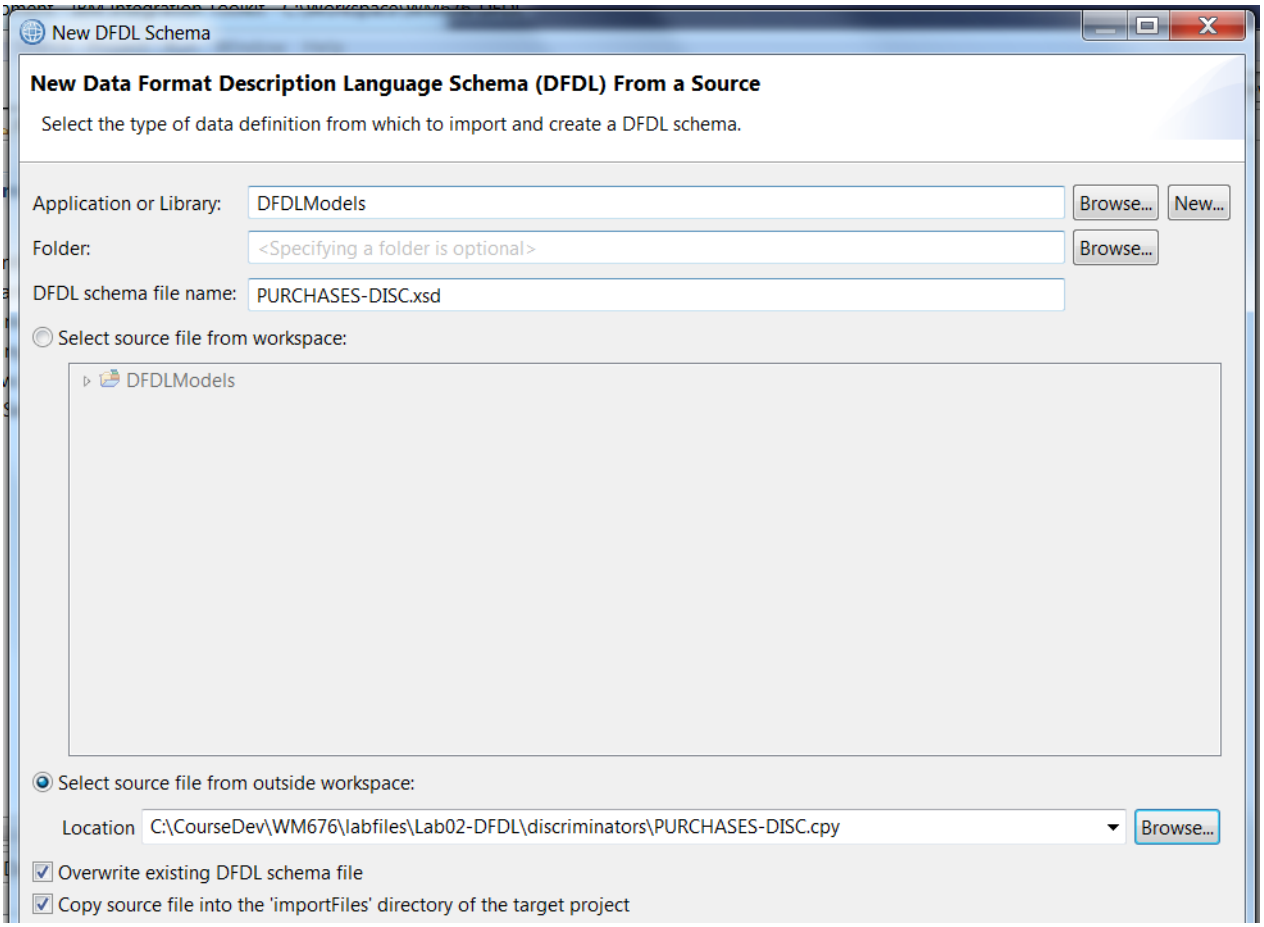
__ c. You can create the new message model by using a wizard, or create an empty DFDL schema and start from scratch.

Leave the default selection to **Create a DFDL schema by importing a COBOL copybook or program** and then click **Next**.

__ d. Select the option to **Select source from outside workspace**.

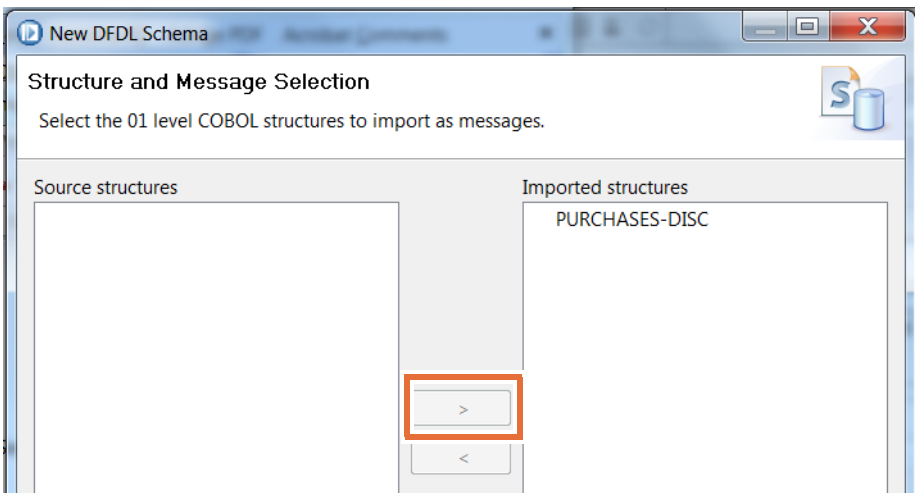
__ e. Click **Browse** and go to the `C:\labfiles\Lab02-DFDL\discriminators` directory.

__ f. Select `PURCHASES-DISC.cpy` file and then click **OPEN**.



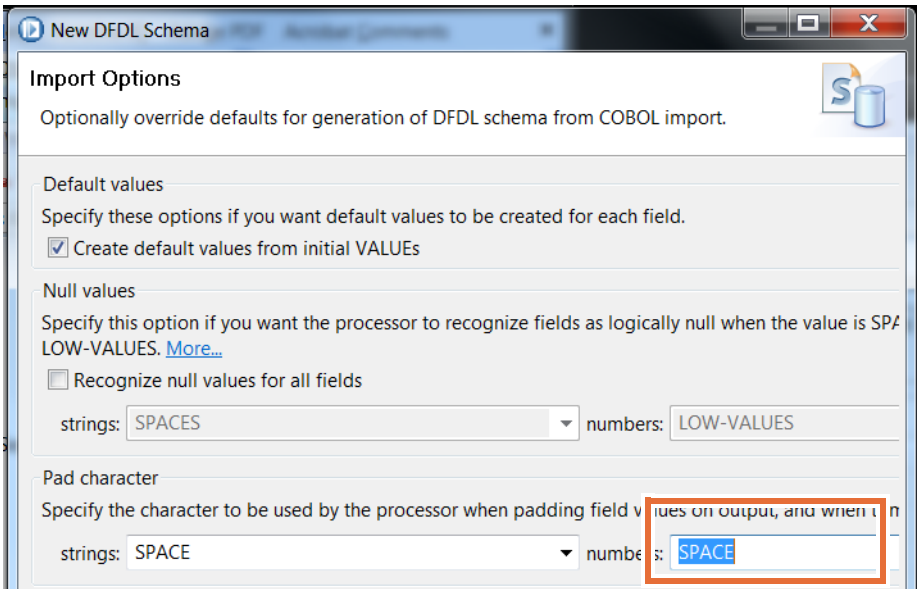
__ g. Click **Next**.

__ h. Click **PURCHASES-DISC** under **Source Structures** and then click the **>** icon to move it to the **Imported structures** pane (use the arrows in the center of the window).



__ i. Click **Next** (do ***not*** click **Finish**).

__ j. In the **Pad character** definition section, select **SPACE** for the **numbers** padding.



__ k. Click **Finish**.

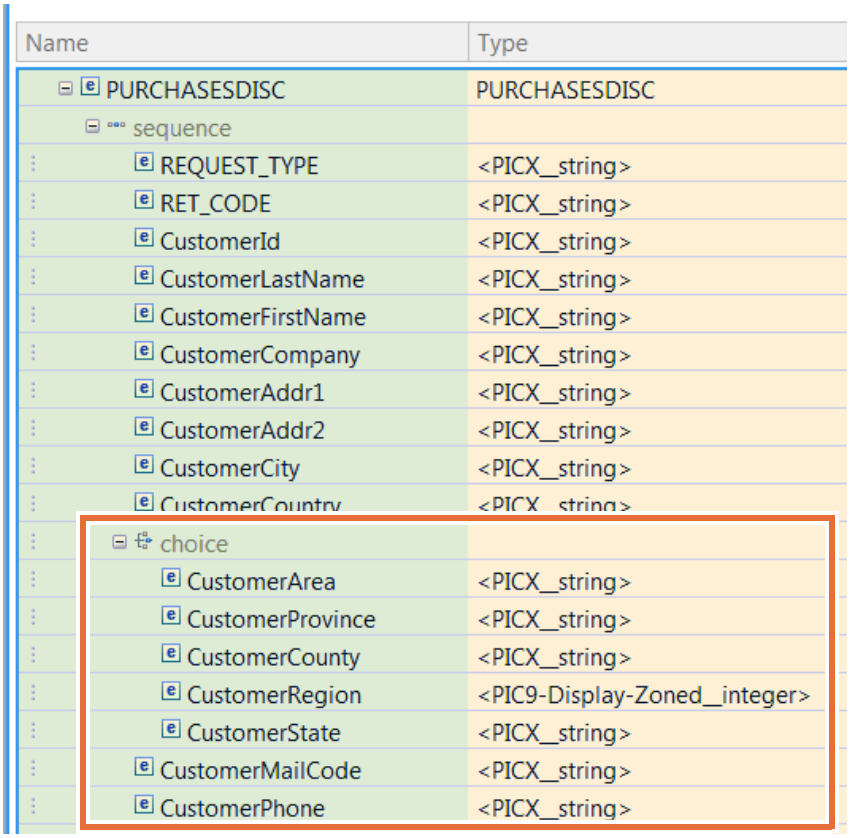
When the wizard finishes, the DFDL Schema editor opens with the generated `PURCHASES-DISC.xsd` schema file.

__ 2. The DFDL schema contains a **choice** element that is defined in the COBOL REDEFINES clauses.

__ a. Expand the **choice** element.

The first **choice** element is **CustomerArea**, which corresponds to the primary definition of this item in the copybook.

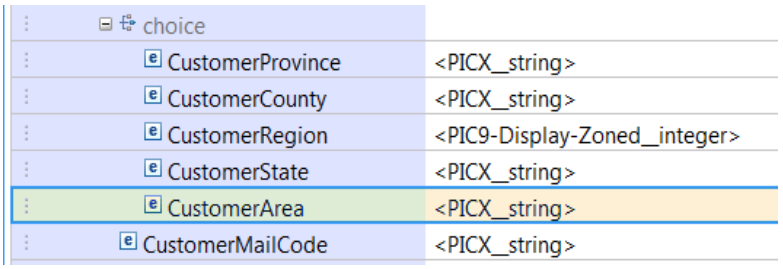
The remaining **choice** elements correspond to the COBOL items in the copybook with the REDEFINES keyword. The order of the elements in the COBOL copybook determines the order of the choice elements in the schema, so the first element is **CustomerArea**.



__ 3. In this exercise, the **CustomerArea** choice is the default. Choice routes are evaluated in order, so this choice must be moved to the bottom of the choices.

__ a. Right-click the element **CustomerArea** (to the left of the element name), and then select **Move Down**. Alternatively, you can highlight the element name and click the yellow down-arrow from the toolbar.

Move the **CustomerArea** element to the bottom of the **choice** element.



__ 4. Save the model and verify that no problems are listed in the **Problems** view.

__ 5. Test the base model by using the Test Parse tool.

__ a. Click **Test Parse Model**.

__ b. Click **Content from a data file** and then click **Browse**.

__ c. Click **Select an input file from the file system** and then click **Browse**.

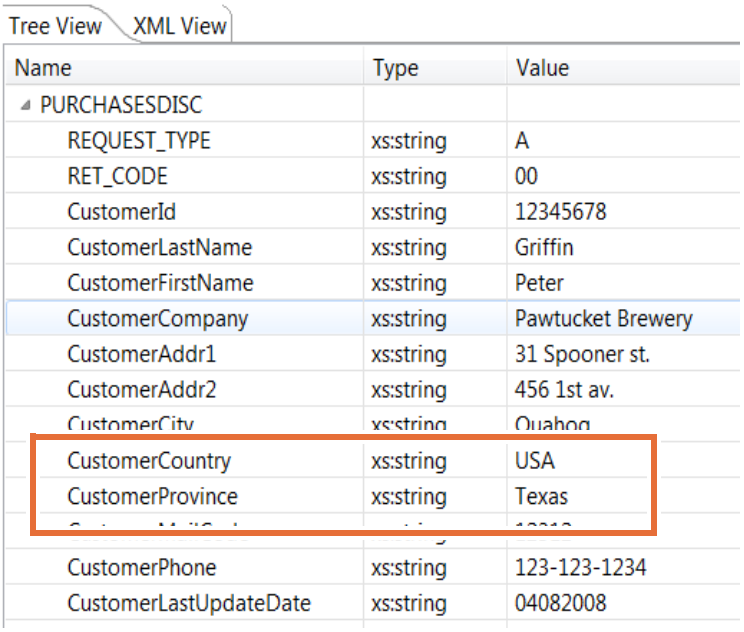
__ d. Go to the `C:\labfiles\Lab02-DFDL\discriminators` directory, click `Purchases_disc_USA.dat` from the file system, and then click **Open**.

__ e. Click **OK** on the **File Selection** window. Click **OK** again to run the test.

__ f. Click **Yes** to switch to the **DFDL Test** perspective.

__ g. The test should complete successfully.

In the **DFDL Test – Logical Instance** view, you should see that the data is fully parsed. In the Logical Instance, the **CustomerCountry** is USA, but the value of Texas is placed into the **CustomerProvince** choice element.



CustomerProvince was selected because the branches of the choice are tried in the declared order until one parses successfully. **CustomerState** and **CustomerProvince** are both declared as PIC X(30), so they end up with the same properties in the schema. The result is that when USA data is parsed, the **CustomerArea** element always matches the **CustomerProvince** data definition.

The results are not what is required, so the model must be modified with some discriminators.

__ 6. Add discriminators to the **CustomerProvince**, **CustomerCounty**, **CustomerRegion**, and **CustomerState** elements in the message model.

Discriminators allow the parser to dynamically select the appropriate elements, which are based on values in the incoming message. The parser still parses branches in the declared order until one is found that parses successfully, but the discriminators ensure that the data matches only one of the branches.

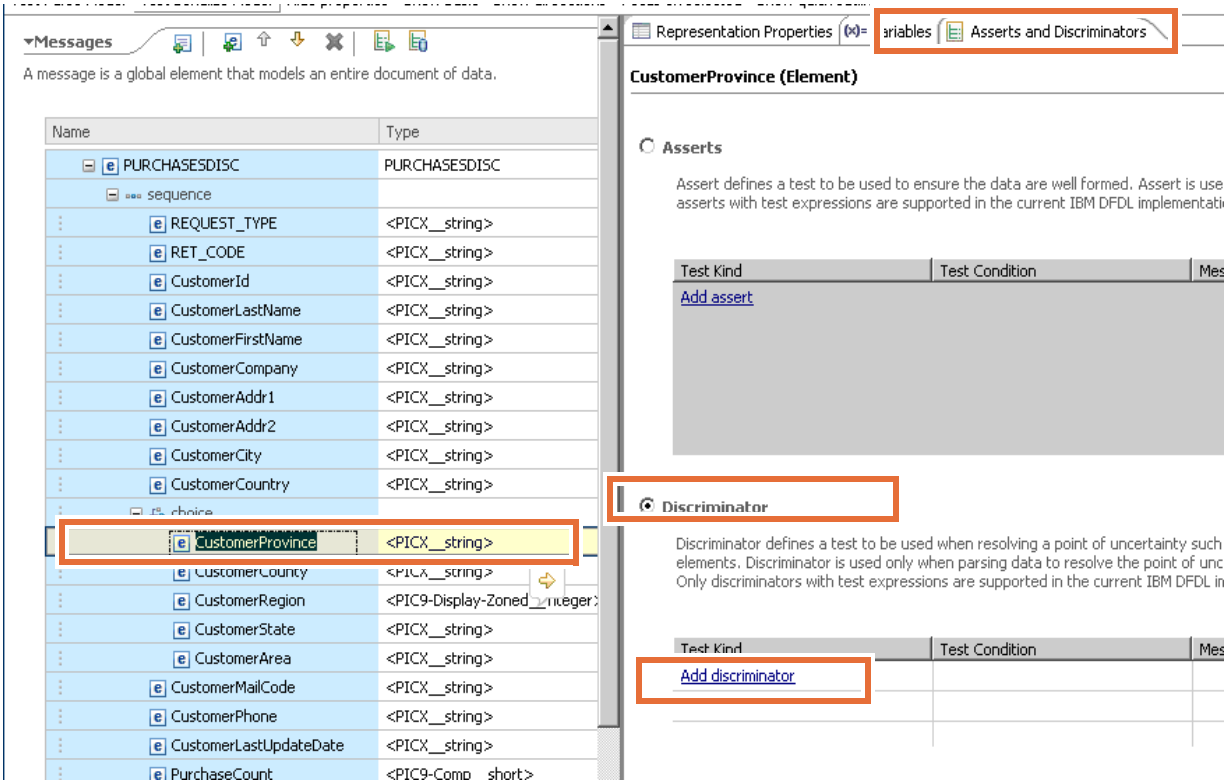
No change is made to the **CustomerArea** element, which acts as a default for all countries that do not have explicit discriminators.

__ a. Switch back to the **Integration Development** perspective.

__ b. In the DFDL editor, expand the **choice** element.

__ c. Click the **CustomerProvince** element, and then click the **Asserts and Discriminators** tab.

__ d. On the **Asserts and Discriminators** tab, click **Discriminator**.



__ e. Click **Add discriminator**.

__ f. Use the Content Assist function to define the **Test condition**.

With the mouse in the **Test Condition** field, press Ctrl+Space and then double-click **DFDL Expression**.

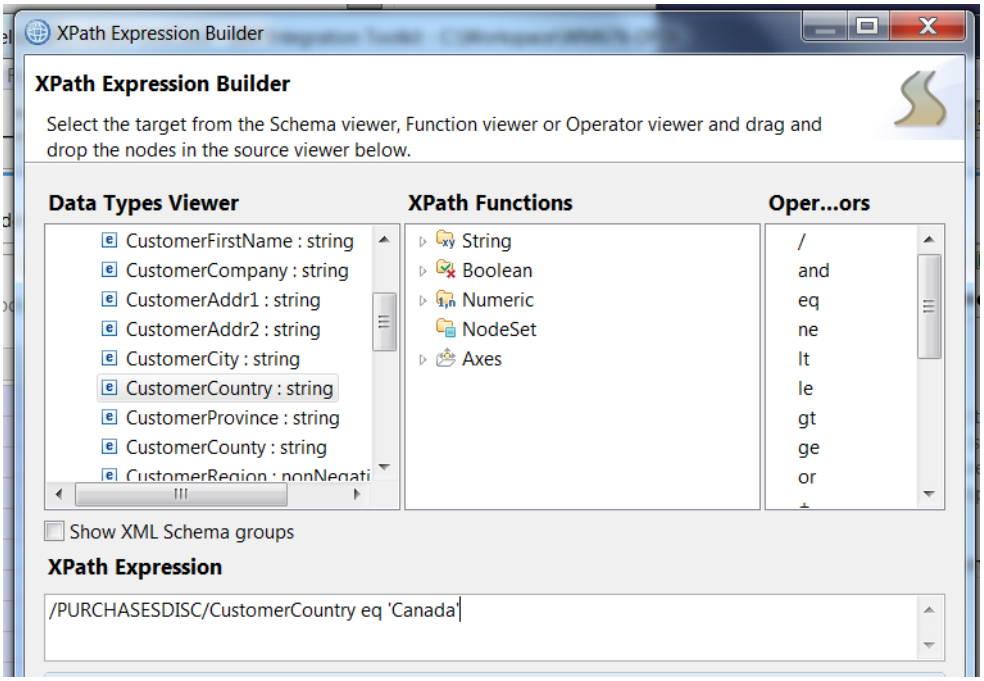
Generate an XPath expression that checks that the **CustomerCountry** element is Canada. When this value is detected, the parser uses the element **CustomerProvince** to parse the data in the **CustomerState** element (redefined by the element **CustomerProvince**).

Expand `PURSHASESDISC : PURCHASESDISC` in the **Data Types Viewer**.

Drag the **CustomerCountry** element to the XPath Expression pane. Complete the XPath expression manually and then click **Finish**.

The final expression should be:

```
/PURCHASESDISC/CustomerCountry eq 'Canada'
```



__ g. Repeat steps d – f for the element **CustomerCounty**.

In this case, **CustomerCounty** should be used to parse the element if the **CustomerCountry** is `UK` or `Ireland`. (Do not confuse **CustomerCountry** with **CustomerCounty**.)

The XPath expression for the element **CustomerCounty** should be:

```
/PURCHASESDISC/CustomerCountry eq 'UK' or
/PURCHASESDISC/CustomerCountry eq 'Ireland'
```

__ h. Repeat the steps to add a discriminator for **CustomerRegion**.

In this case, **CustomerRegion** should be used to parse the element if the **CustomerCountry** is `Russia`.

The XPath expression for the element **CustomerRegion** should be:

```
/PURCHASESDISC/CustomerCountry eq 'Russia'
```

__ i. Repeat the steps to add a discriminator for the element **CustomerState**.

In this case, **CustomerState** should be used to parse the element if the **CustomerCountry** is `USA`.

The XPath expression for the element **CustomerState** should be:

```
/PURCHASESDISC/CustomerCountry eq 'USA'
```

__ 7. Save the model.

__ 8. Retest the model with discriminators.

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

__ b. Select **Content from a data file**, click **Browse**, and select an input file from the file system.

__ c. Browse to the `C:\Labs\Lab02-DFL\discriminator` directory and then select the file `Purchases_disc_USA.dat`. The file should parse successfully.

__ d. In the **DFDL Test – Logical Instance** view, verify that the **CustomerCountry** is USA.

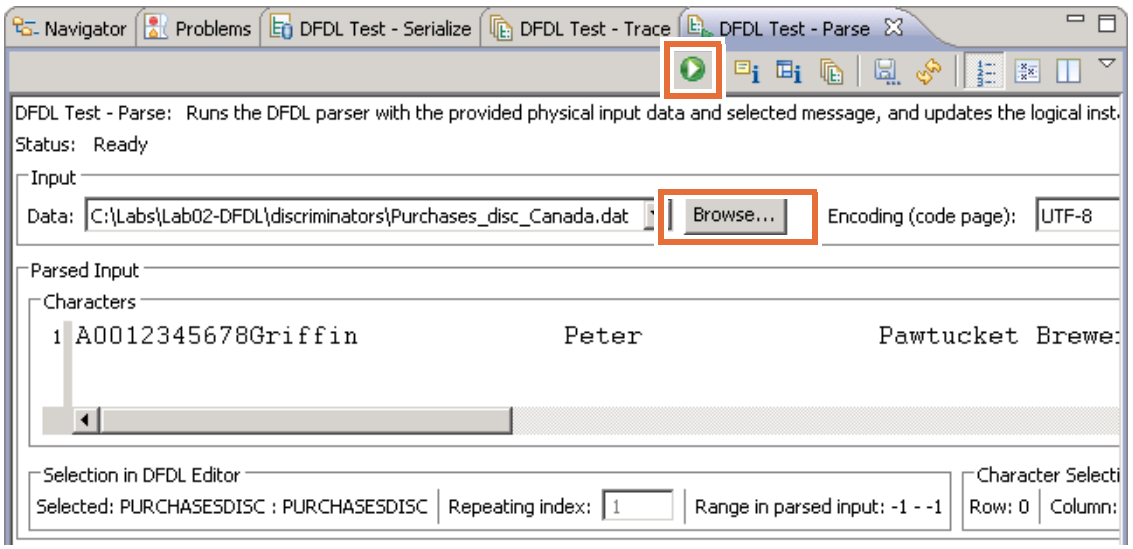
Because the discriminators were specified on the choice, the parser detected that the value of USA matches a discriminator, and placed the value Texas into the element **CustomerState**, replacing the choice element **CustomerArea**.

Name	Type	Value
PURCHASESDISC		
REQUEST_TYPE	xs:string	A
RET_CODE	xs:string	00
CustomerId	xs:string	12345678
CustomerLastName	xs:string	Griffin
CustomerFirstName	xs:string	Peter
CustomerCompany	xs:string	Pawtucket Brewery
CustomerAddr1	xs:string	31 Spooner st.
CustomerAddr2	xs:string	456 1st av.
CustomerCity	xs:string	Quahog
CustomerCountry	xs:string	USA
CustomerState	xs:string	Texas
CustomerMailCode	xs:string	12312
CustomerPhone	xs:string	123-123-1234

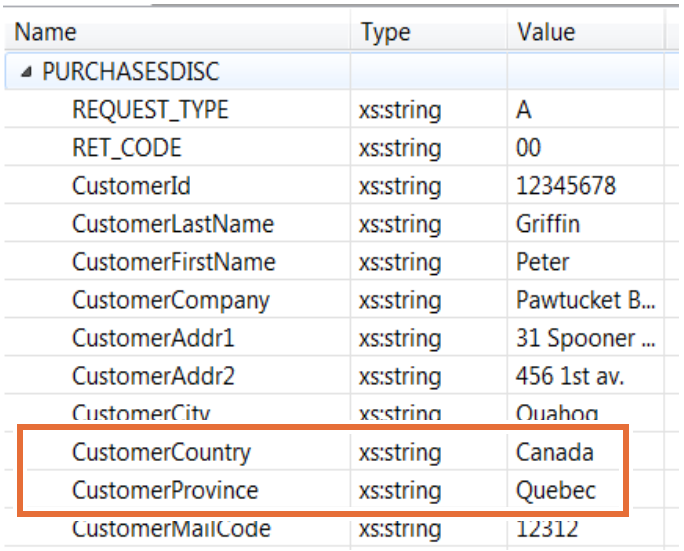
__ e. Test the DFDL model with another file.

On the **DFDL Test – Parse** tab, click **Browse** and then select the input file **Purchases_disc_Canada.dat**.

Click the green **Run Parser** icon.



In this case, the parser determined that the value in this element matches one of the specified discriminators, and used the choice element **CustomerProvince** to represent the appropriate element. It set the value to Quebec.



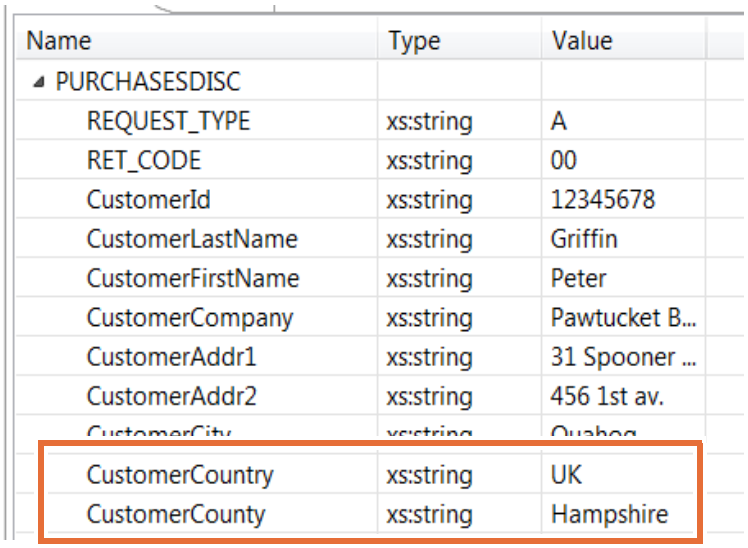
Name	Type	Value
PURCHASESDISC		
REQUEST_TYPE	xs:string	A
RET_CODE	xs:string	00
CustomerId	xs:string	12345678
CustomerLastName	xs:string	Griffin
CustomerFirstName	xs:string	Peter
CustomerCompany	xs:string	Pawtucket B...
CustomerAddr1	xs:string	31 Spooner ...
CustomerAddr2	xs:string	456 1st av.
CustomerCity	xs:string	Quahog
CustomerCountry	xs:string	Canada
CustomerProvince	xs:string	Quebec
CustomerMailCode	xs:string	12312

__ f. Test the model with the Purchases_disc_UK.dat file.

On the DFDL Test – Parse tab, select the input file Purchases_disc_UK.dat and then click the **Run Parser** icon.

In the **Logical Instance** view, you should see that the **CustomerCountry** is UK.

In this case, the parser determined that the value in this element matches one of the specified discriminators, and used the element **CustomerCounty** to parse the appropriate element. It replaced the element **CustomerState** with the element **CustomerCounty**, and set the value to Hampshire.



Name	Type	Value
PURCHASESDISC		
REQUEST_TYPE	xs:string	A
RET_CODE	xs:string	00
CustomerId	xs:string	12345678
CustomerLastName	xs:string	Griffin
CustomerFirstName	xs:string	Peter
CustomerCompany	xs:string	Pawtucket B...
CustomerAddr1	xs:string	31 Spooner ...
CustomerAddr2	xs:string	456 1st av.
CustomerCity	xs:string	Quahog
CustomerCountry	xs:string	UK
CustomerCounty	xs:string	Hampshire

__ g. Run the parser with input file Purchases_disc_Russia.dat. The file that contains data from Russia has the data in the field **CustomerRegion** as numeric, with leading blank characters.

In the **Logical Instance** view, you should see that the **CustomerCountry** is `Russia`. The parser determined that the value in this element matches one of the specified discriminators, and used the element **CustomerRegion** to represent the appropriate element. It set the **CustomerRegion** value to `6938495028`.

__ h. Run the parser with the input file `Purchases_disc_France.dat`.

The message model does not have any special definition for `France`, so the parser does not match any discriminators. The default choice is selected and the **Logical Instance** view shows the element **CustomerArea**, with the value `Paris`.

__ 9. Return to the **Integration Development** view.

__ 10. Close all open editors.

End of exercise

Exercise review and wrap-up

In the first part of this lab, you created a DFDL message model by adding a header record and a trailer record to a reference to an existing model.

In the second part of this lab, you added discriminators to an existing DFDL model.

Having completed this exercise, you should be able to:

- Extend a DFDL model to add header and trailer records
- Reference a DFDL model in a new DFDL model
- Use discriminators in a DFDL model so that the parser can conditionally process elements as determined by the content of other elements in the data and optimize the parsing of data

Exercise 3. Implementing web services

What this exercise is about

In this exercise, you use SOAP nodes to implement web services. You implement message flows to act as both a provider and a user of web services.

What you should be able to do

After completing this exercise, you should be able to:

- Provide a message flow as a web service that uses SOAP/HTTP
- Start a SOAP/HTTP web service asynchronously
- Test a SOAP/HTTP message flow
- Implement WS-Addressing
- Use the TCP/IP Monitor to view the data as it passes through the SOAP/HTTP message transport

Introduction

In this exercise, you implement web services within a set of message flows by using SOAP nodes.

One message flow is the producer of a web service. The other message flow is the consumer of the web service. Both message flows consist of a main flow and a subflow. In the first part of this exercise, you review the flows and subflows in detail. You then deploy and test the message flows with a simple message.

Next, you start the TCP/IP Monitor that is contained within the IBM Integration Toolkit. You can use the TCP/IP Monitor to monitor web service request and response messages.

Finally, you implement WS-Addressing within the message flows, and test them again to observe the differences in the message content and structure from messages that do not use WS-Addressing.

Requirements

- IBM Integration Bus V10
- IBM MQ V8 with a queue manager that is named IIBQM
- Basic understanding of IBM MQ

- Lab exercise files in the `C:\labfiles\Lab03-WebSvcs` directory

Exercise instructions

Exercise setup

__ 1. Make sure that all local IBM Integration Bus components are running.

__ 2. Start the IBM Integration Toolkit (if it is not already running) and switch to a new workspace such as `C:\Workspace\webservices`.

__ 3. This exercise uses the Integration Toolkit Unit Test Client to test the web services flow. Enable the Unit Test Client menus.

__ a. In the Integration Toolkit, click **Window > Preferences**.

__ b. In the Preferences window, expand **Integration Development**.

__ c. Click **Unit Test Client**.

__ d. On the **Unit Test Client** page, click **Enable menus for test client**, click **Apply**, and then click **OK**.

__ 4. The message flows in this exercise require access to IBM MQ application queues on a queue manager that is named `IIBQM`.

You created a queue manager that is named `IIBQM` in Exercise 1.

If you did not complete Exercise 1, open IBM MQ Explorer and create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ`.

__ 5. Create the IBM MQ queues for this exercise on the queue manager that is named `IIBQM`: `SOAPSAMPLE_FAULT`, `SOAPSAMPLE_IN`, and `SOAPSAMPLE_OUT`

The lab exercise files include an IBM MQ script that creates the local queues.

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

```
runmqsc IIBQM < C:\labfiles\Lab03-WebSvcs\resources\Q_Defs.mqsc
```

__ b. Confirm that the script created the local queues for the application.

Part 1: Import and review the web services message flows

In this part of the exercise, you import and review the SOAP nodes web service application.

__ 1. Import the **`SOAPNodesWebSvc.zip`** project interchange file into the Integration Toolkit workspace.

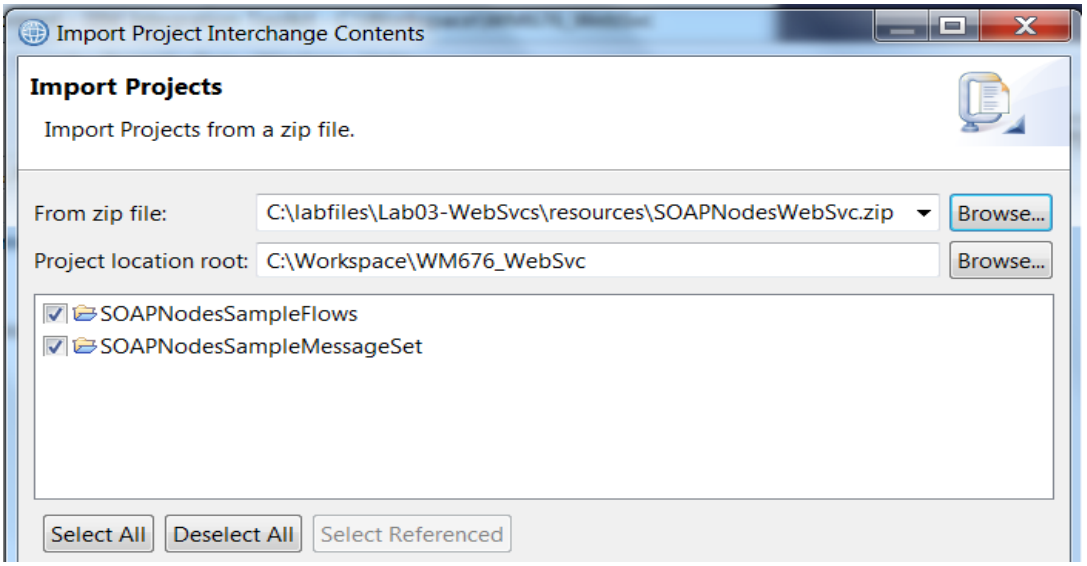
__ a. From the main menu bar, click **File > Import**.

__ b. In the Import wizard, click **IBM Integration > Project Interchange** and then click **Next**.

__ c. Click **Browse**.

__ d. Go to the **`C:\labfiles\Lab03-WebSvcs\resources\`** folder, click the **`SOAPNodesWebSvc.zip`** file, and then click **Open**.

__ e. In the **Import Projects** window, verify that both projects are selected and then click **Finish**.



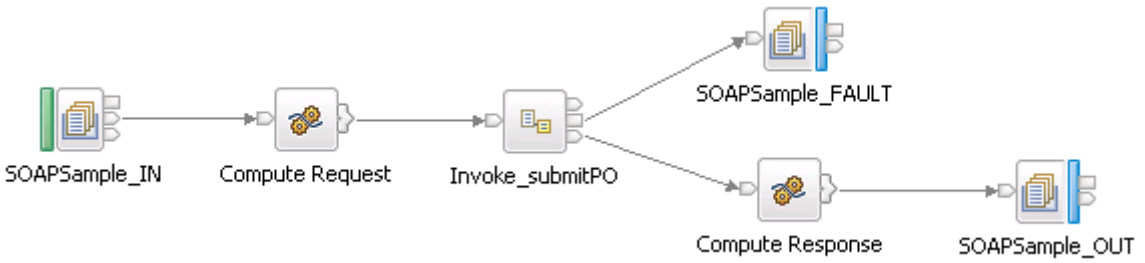
__ 2. Review the web service consumer message flow.

__ a. Expand **Independent Resources > SOAPNodesSampleFlows > Flows**.

The project contains two message flows: a web service consumer message flow and a web service provider message flow.

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

The **SOAPNodesSampleConsumerFlow** is the main flow of the message consumer.

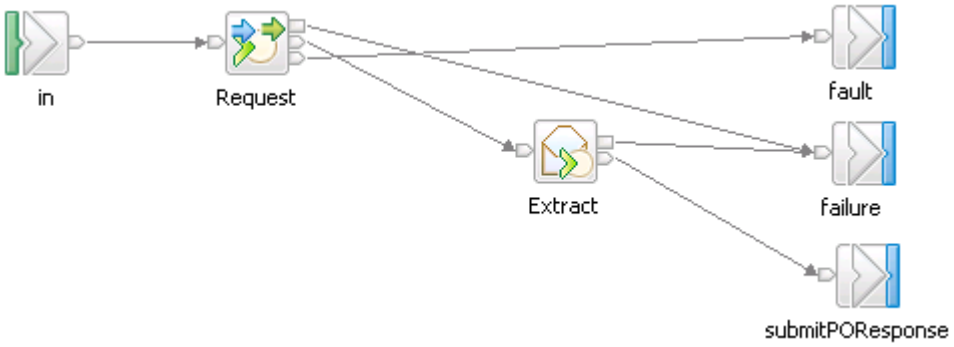


- **SOAPSample_IN** is an MQ Input node. It accepts an XML message from the SOAPSAMPLE_IN queue on the IIBQM queue manager to start the message consumer flow.
- **Compute Request** is a Compute node that modifies the incoming XML message in preparation for sending it to the web service.
- **Invoke_submitPO** calls the subflow that sends the SOAP request. This subflow is explained further in an upcoming step.
- **SOAPSample_FAULT** is an MQ Output node that writes a fault message to the SOAPSAMPLE_FAULT queue on the IIBQM queue manager when a failure occurs in the subflow.

- **Compute Response** is a Compute node that formats an output XML message that was returned from successful invocation of the SOAP service in the subflow.
- **SOAPSample_OUT** is an MQ Output node. It writes the formatted message to queue SOAPSAMPLE_OUT on the IIBQM queue manager.

__ 3. Examine the **submitPO** message flow.

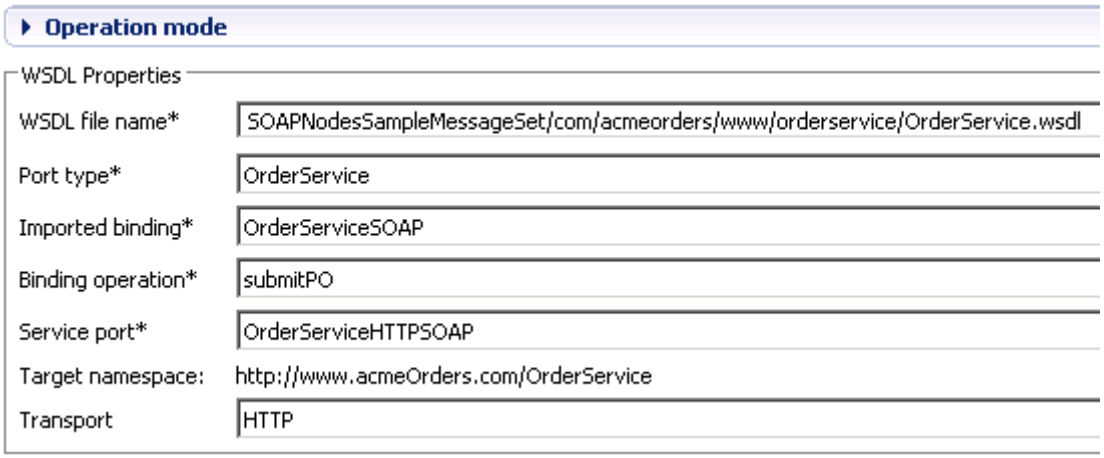
__ a. Double-click the **Invoke_submitPO** subflow node in the **SOAPNodesSampleConsumerFlow** message flow to open the subflow in the Message Flow editor.



__ b. Examine the **Request** node in the **submitPO** subflow.

The **Request** node is a SOAP Request node. It creates a SOAP message and sends it to the web service as defined in the node properties.

__ c. Examine the **Basic** properties of the **Request** node.



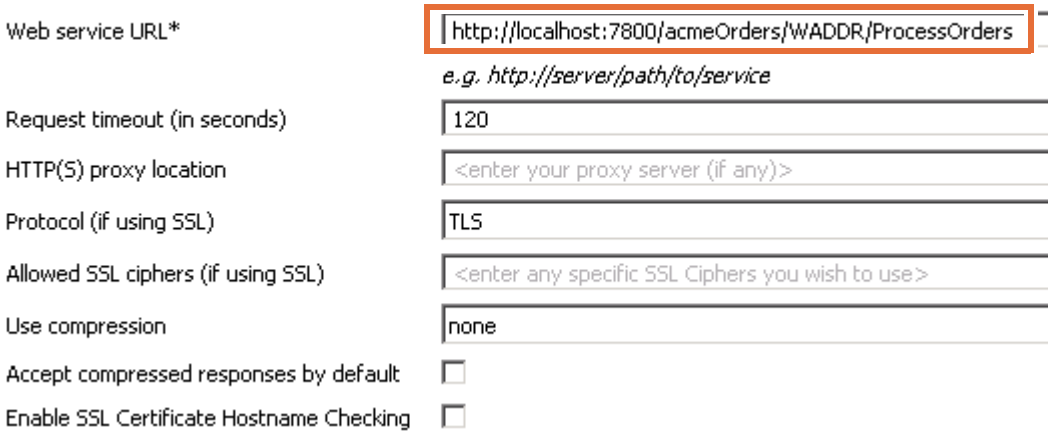
Note

These properties are configured automatically when you take any of the following actions:

- You create a message flow application by using the **Start from WSDL and/or XSD files** quick start wizard.
- You add a SOAP node to the canvas and then access a WSDL file by clicking **Browse** in the properties.

- You add a SOAP node to the canvas and then drag a WSDL file onto the node in the canvas.

__ d. Click the **HTTP Transport** property tab for the **Request** node and note the **Web service URL** value.



The SOAP Request node sends a web service request message to the service at the specified address: `http://localhost:7800/acmeOrders/WADDR/ProcessOrders`.

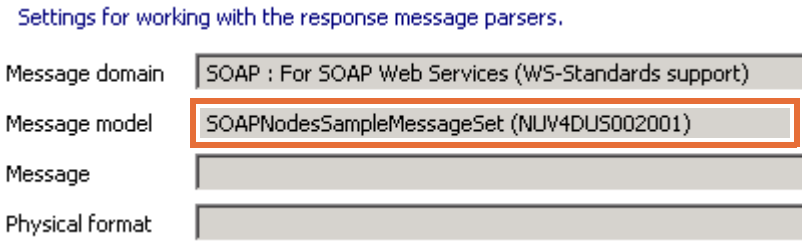
Information

By default, SOAP Input, SOAP Reply, and SOAP AsyncResponse nodes use the integration server embedded listener. The `HTTPConnector` listener object controls the runtime properties that affect the handling of HTTP messages. Each connector has its own assigned port, which is allocated from a range of numbers, as required. The default range for the HTTPConnector is 7800 – 7842.

This exercise assumes that the integration server is using the `HTTPConnector` port 7800.

__ e. Click the **Response Message Parsing** property tab for the **Request** node.

The **Message model** property identifies the message set that defines the data: `SOAPNodesSampleMessageSet`.



__ f. Examine the properties for the **Extract** node in the **submitPO** subflow.

The **Extract** SOAP Extract node removes the SOAP envelope from the (valid) response message and converts it to the XMLNSC domain.

__ g. Examine the properties of the **submitPOResponse** Label node in the **submitPO** subflow. It receives the message from the SOAP Extract node, and then exits the subflow.

__ 4. Review the web service provider message flow.

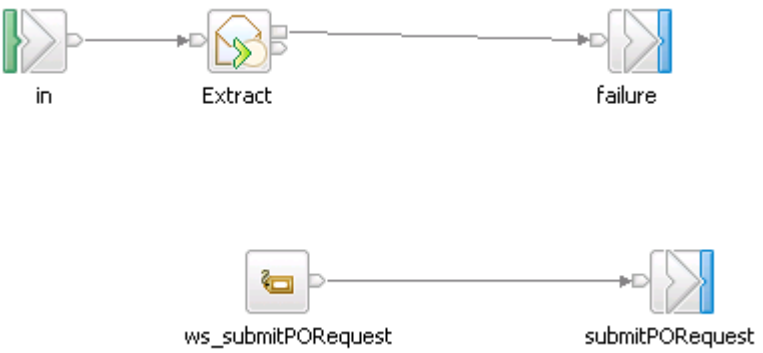
In the **Application Development** view, double-click **SOAPNodesSampleProviderFlow.msgflow** to open the message flow.

The **SOAPNodesSampleProviderFlow** message flow is the main flow of the message provider.

- **Input** is a SOAP Input node. It receives an incoming SOAP message, validates it, and then passes it on.
- **OrderService_Extract** calls the subflow that extracts the SOAP envelope from the message. The subflow is explained in more detail later in this exercise.
- **Compute Response** is a Compute node that creates XML data that simulates a valid response. In an actual business application, you might reference the SOAP message as XML, and modify or transform it as needed for the application.
- **Reply** is a SOAP Reply node. It creates a valid SOAP message from the data that is passed to it from the preceding Compute node. It then returns the message to the web service consumer that started the initial web service call.

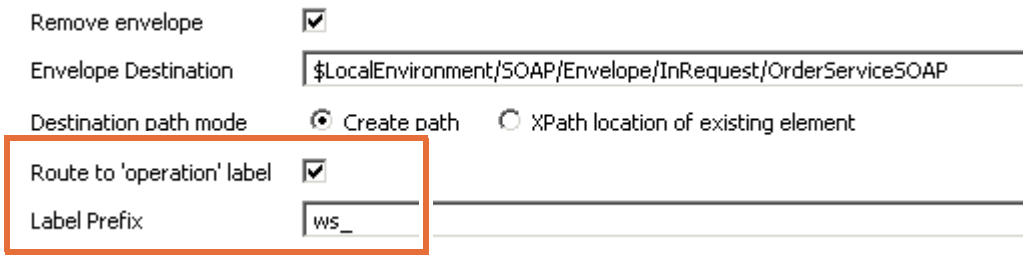
__ 5. Examine the **OrderService** subflow.

__ a. Double-click the **OrderService_Extract** subflow node in the message flow. This operation opens the subflow of the message producer in the Message Flow editor.



__ b. Examine the **Extract** node **Basic** properties.

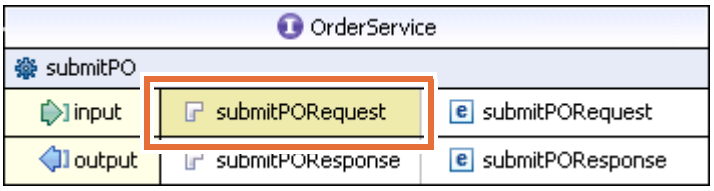
The SOAP Extract node **Extract** removes the SOAP envelope. Removing the SOAP envelope leaves an XML message that can be altered or transformed, by using a Compute or Mapping node, for example.



When the **Route to 'operation' label** property is selected, the message node routes the message to a Label node that matches the operation name in the SOAP request message.

The **Label Prefix** property concatenates a value to the operation name. In this example, the prefix that is attached to the operation name is "**ws_**".

You can create a multi-way branch in the message flow with the **Route to 'operation' label** property. This feature maps the request message name for a WSDL operation to a Label node. In this exercise, the request message name for the **submitPO** operation is **submitPORequest**.



The node adds the label prefix **ws_** to the operation name to match the Label node **ws_subPORequest**.

__ c. Examine the properties of the Label node **ws_submitPORequest**. This node is the target of the flow of control from the SOAP Extract node.

__ 6. Close the open message flows.

Part 2: Run the web services message flows

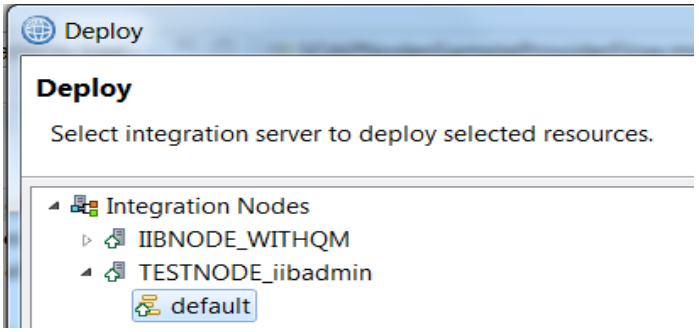
In this part of the exercise, you run the imported web services application.

__ 1. Deploy the SOAP nodes sample application to the SOAPNodesSample integration server.

__ a. In the **Application Development** view, expand the **BARs** project.

__ b. Right-click **SOAPNodesSampleBAR.bar > SOAPNodesSampleFlows/SOAPNodesSampleBAR.bar** and then click **Deploy**.

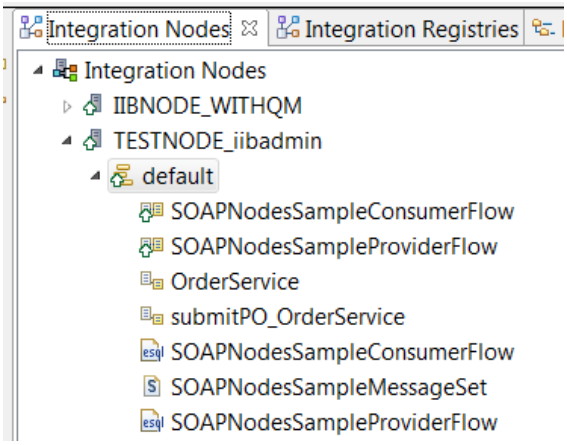
__ c. Click the integration server that is named **default** on the integration node that is named **TESTNODE_iibadmin**.



__ d. Click **Finish**.

__ e. Verify that the application deployed successfully in the **Deployment Log** view.

__ 2. Confirm that the SOAP node message flows and message sets are deployed in the **Integration Nodes** view.



__ 3. Using the **`mqsireportproperties`** command, verify that the integration server HTTPConnector is running on port 7800 (the port value that is specified in the web services URL in the message flow).

In an IBM Integration Console, type:

```
mqsireportproperties TESTNODE_iibadmin -e default -o HTTPConnector -r
```

The command should return `7800` for the HTTPConnector port.

Information

The SOAP Request node in the web service consumer message flow sends web service requests to the following web address:

`http://localhost:7800/acmeOrders/WADDR/ProcessOrders`

The web service provider message flow accepts SOAP request messages at port 7800.

You can also check that the integration server is listening on port 7800 by using the `netstat` command, for example: `netstat –a | findstr 7800`

__ 4. Run the sample by using the Integration Toolkit Unit Test Client.

__ a. In the Integration Toolkit **Application Development** view, expand **Independent Resources > SOAPNodesSampleFlows > Flow tests**.

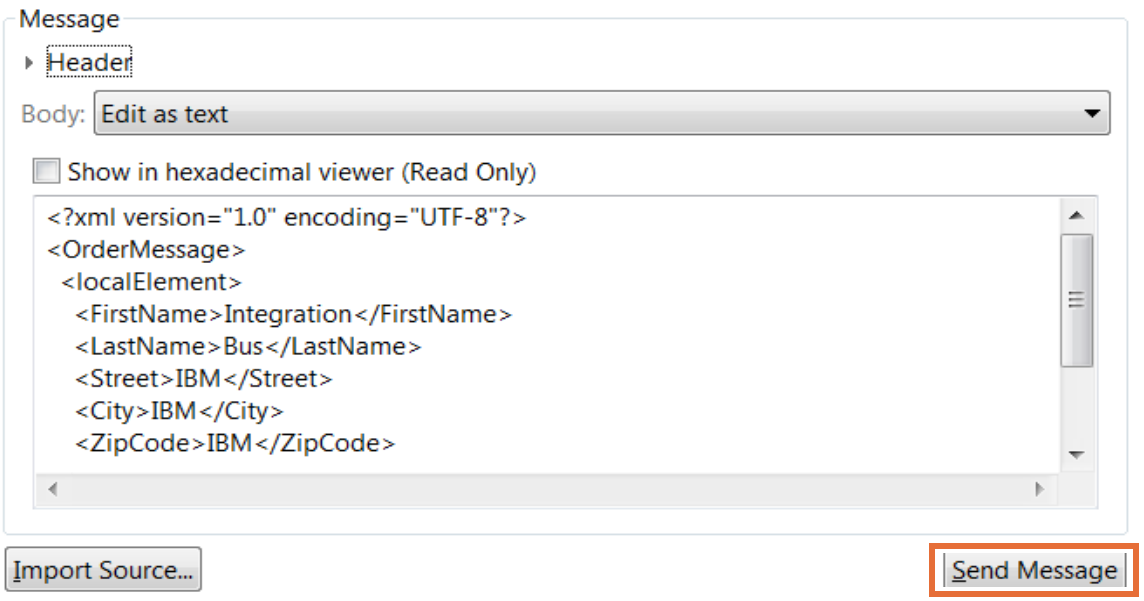
__ b. Double-click **SOAPNodesSampleConsumerFlow.mbtest**. The Test Client window opens.

__ c. Review the message that you are sending. You can enlarge the window to do so.

Note

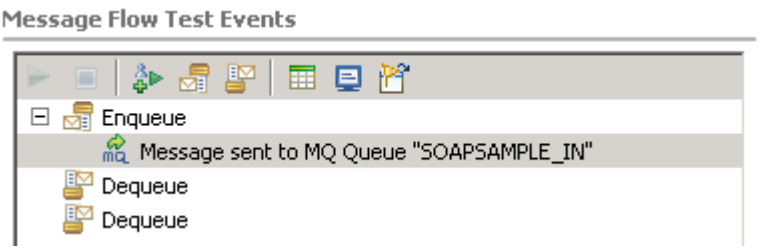
To maximize the Unit Test Client window, double-click the tab. Double-click its tab again to return to the default size.

__ d. Click **Send Message**.



__ e. The Test Client indicates that a message was queued to the SOAPSAMPLE_IN queue.

This queue is the one that the message consumer uses to send the request.



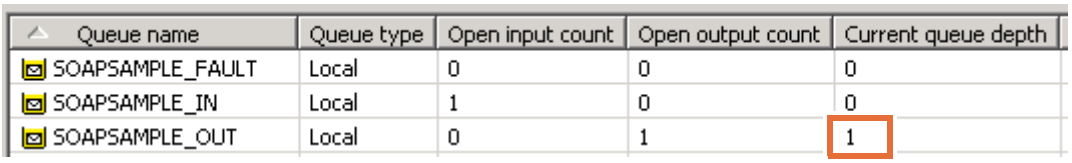
__ 5. If the application runs successfully, a message is enqueued on the SOAPSAMPLE_OUT queue. Use the IBM MQ Explorer to view the queues.

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

__ b. Expand **IBM WebSphere MQ > Queue Managers > IIBQM**.

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

__ d. In the **Queues** content view, check the **Current queue depth** column for the SOAPSAMPLE_OUT queue.

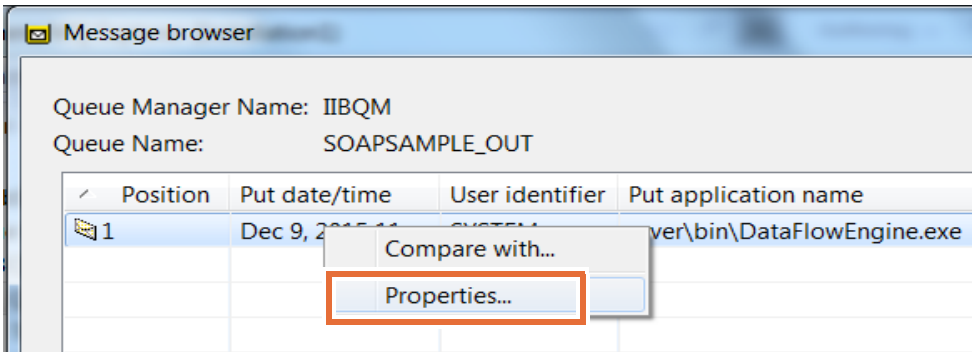


Queue name	Queue type	Open input count	Open output count	Current queue depth
SOAPSAMPLE_FAULT	Local	0	0	0
SOAPSAMPLE_IN	Local	1	0	0
SOAPSAMPLE_OUT	Local	0	1	1

The queue should contain one message.

__ e. To browse the message, right-click **SOAPSAMPLE_OUT**, and then click **Browse Messages**.

__ f. To view the detailed message, right-click the row in the table that contains message 1, and then click **Properties**.



__ g. In the **Properties** window, click **Data** in the left pane. The message is displayed in hexadecimal and character formats in the right pane. Review the message that was enqueued.

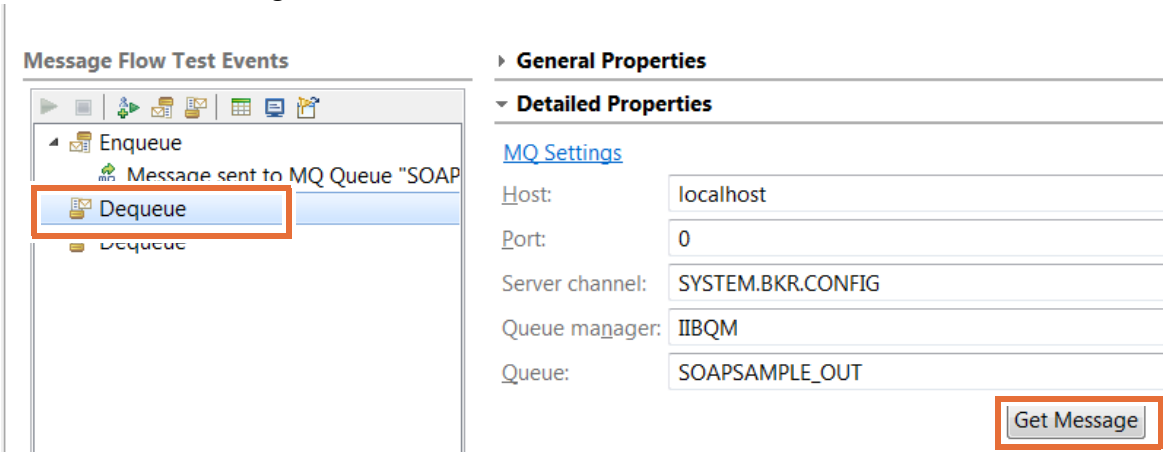
__ h. Close the **Message Browser** window.

__ 6. Use the Unit Test Client to view the message.

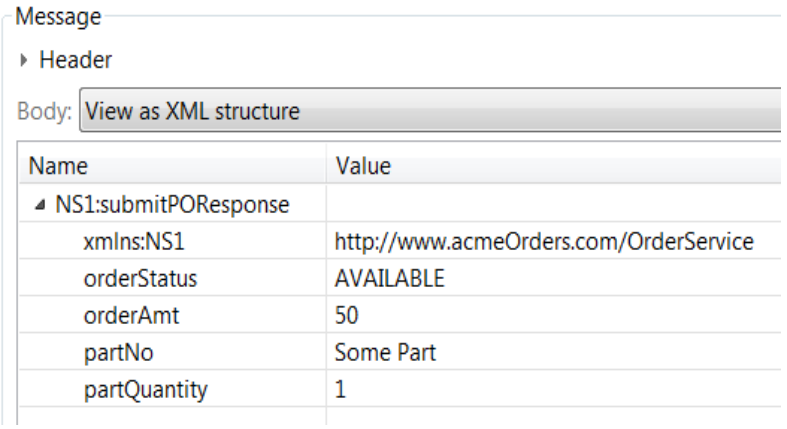
__ a. In the **Message Flow Test Events** pane, click **Dequeue**.

__ b. Verify that **MQ Settings** under the **Detailed Properties** section identify the queue manager `IIBQM` and the output queue SOAPSAMPLE_OUT.

__ c. Click **Get Message**.



__ d. Inspect the contents of the message from the SOAPSAMPLE_OUT queue. The data represents the response from the `SubmitPO` web service operation.



Part 3: Using the TCP/IP Monitor

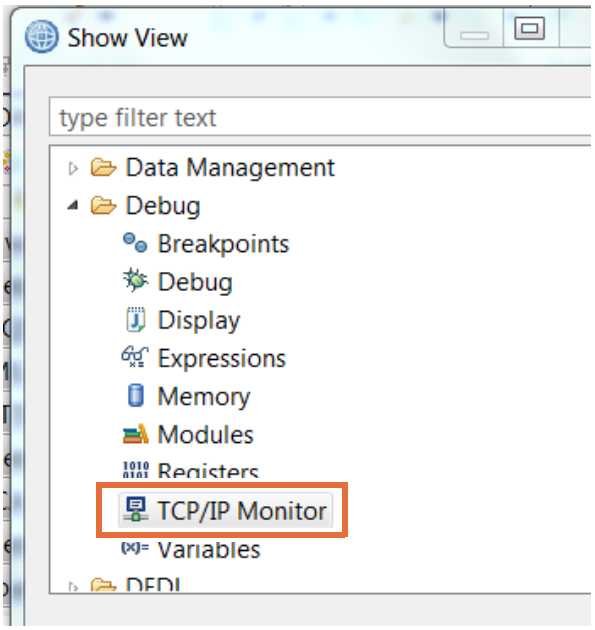
The Integration Toolkit includes Eclipse Workbench tools. The TCP/IP Monitor tool is an Eclipse tool that you can use to view data as it passes through the SOAP/HTTP message transport.

In this part of the exercise, you extend the sample by using the TCP/IP Monitor to view the message that is sent from the SOAP Request node to the SOAP Input node. The TCP/IP Monitor also shows the message that the SOAPReply node returns to the SOAPRequest node.

__ 1. Open the TCP/IP Monitor view.

__ a. From the IBM Integration Toolkit toolbar, click **Window > Show View > Other**. The **Show View** menu is displayed.

__ b. Expand **Debug**, click **TCP/IP Monitor**, and then click **OK**.

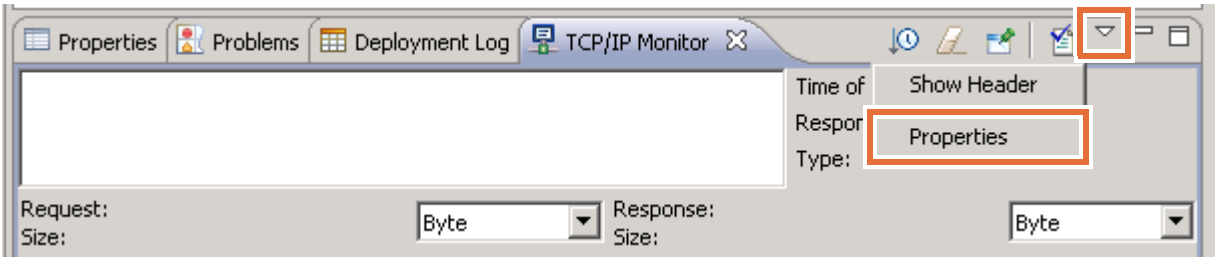


__ c. Confirm that Integration Toolkit opens the TCP/IP Monitor view.

__ 2. Configure the TCP/IP Monitor settings.

__ a. In the **TCP/IP Monitor** view toolbar, click the **View Menu** icon (the downward-pointing triangle in the view toolbar).

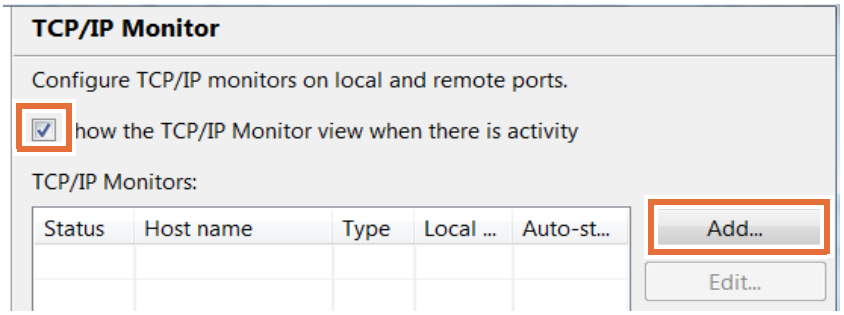
__ b. Click **Properties**.



The **Preferences** dialog box is displayed.

__ c. In the **Preferences** window, verify that the **Show TCP/IP Monitor view when there is activity** option is selected.

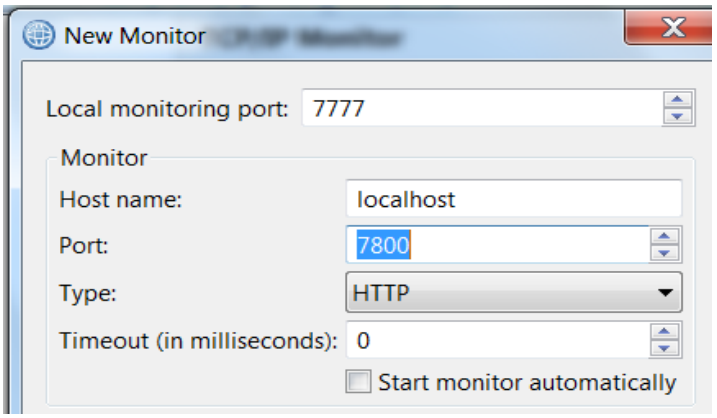
__ d. Click **Add**.



The **New Monitor** dialog box is displayed.

__ e. Set the following parameters:

- Specify a unique port number on your local machine for a local monitoring port. Set **Local monitoring port** to: `7777`
- Specify the host name or IP address of the machine where the server is running by setting **Host name** to: `localhost`
- Specify the port number of the remote server by setting **Port** to: `7800`



Information

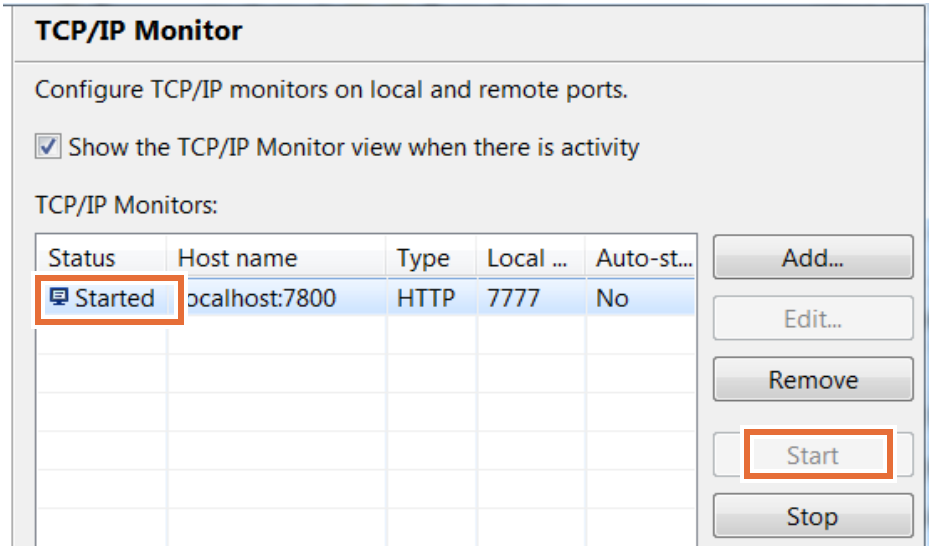
In the TCP/IP Monitor, the **Type** field specifies whether the request type is sent via HTTP or TCP/IP.

- If the **HTTP** option is selected, the requests are modified so that the HTTP header points to the remote machine and separated if multiple HTTP requests are received in the same connection.
- If the **TCP/IP** option is selected, all the requests are sent byte for byte.

The **Timeout** property specifies the time to wait before attempting to connect again.

__ f. Click **OK**. The monitor information is added to the list of monitors.

__ g. Click the row that was added, and then click **Start**. The **Status** column changes from **Stopped** to **Started**.



__ h. Click **OK** to close the TCP/IP configuration dialog box.

__ 3. Rerun the application test in the Integration Toolkit Unit Test Client.

__ a. In the Unit Test Client **Message Flow Test Events** window, click the **Enqueue** operation.

__ b. Click **Send Message**.

The application runs again. Another message is displayed in the Message Flow Test Events window, showing that another message was enqueued to the SOAPSAMPLE_IN queue.

__ c. Check the TCP/IP Monitor view. No activity is recorded in the TCP/IP Monitor.

The SOAPRequest node in the message flow is not sending the request through the TCP/IP Monitor because you configured the TCP/IP Monitor to use port 7777 as the local monitoring port.

You cannot set the local monitoring port in the TCP/IP Monitor to the same value as the port that is being monitored. Instead, you must change the port that the SOAP Request node uses so that it sends the message through the monitored port.

__ 4. Configure the SOAPRequest node to send messages to the monitored port.

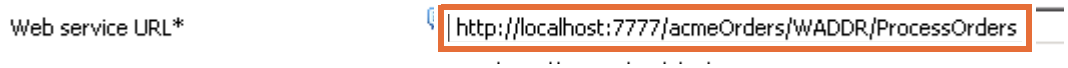
__ a. Open **SOAPNodesSampleConsumerFlow.msgflow**.

__ b. In the Message Flow editor, double-click the **Invoke_submitPO** subflow node.

__ c. In the **submitPO** subflow, click the SOAPRequest node **Request** to display the node properties.

__ d. Click the **HTTP Transport** properties tab.

__ e. Change the port number in the **Web service URL** from 7800 to 7777 (the port that TCP/IP Monitor watches).

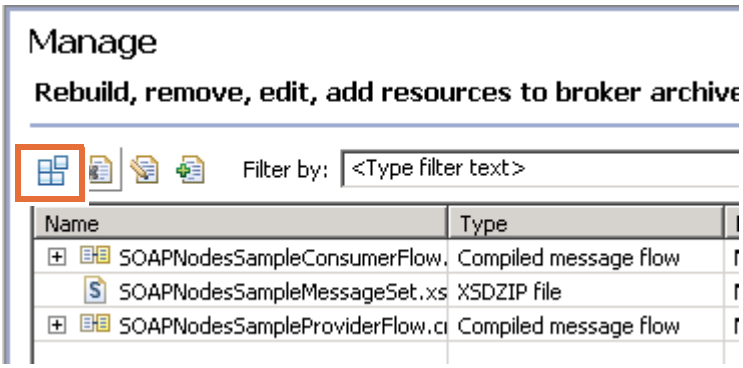


__ f. Save all unsaved changes in the workspace by typing Ctrl + Shift + S.

__ 5. Rebuild and redeploy the BAR file.

__ a. Open the **SOAPNodesSampleBAR.bar** in the BAR File editor.

__ b. Click the **Build** icon in the BAR File editor toolbar (the first icon in the toolbar).



__ c. If you receive messages about modified resources, click **Yes** to save them.

__ d. If you receive a message about overriding configurable properties, click **OK** to allow the override to occur. When the **Operation completed successfully** message is displayed, click **OK**.

__ 6. Deploy the BAR file to the integration server that is named **default** on the **TESTNODE_iibadmin** integration node.

Wait for the deployment operation to complete. Verify that the deployment was successful by reviewing the **Deployment** tab for a success message for the current date and time.

__ 7. Run the application test.

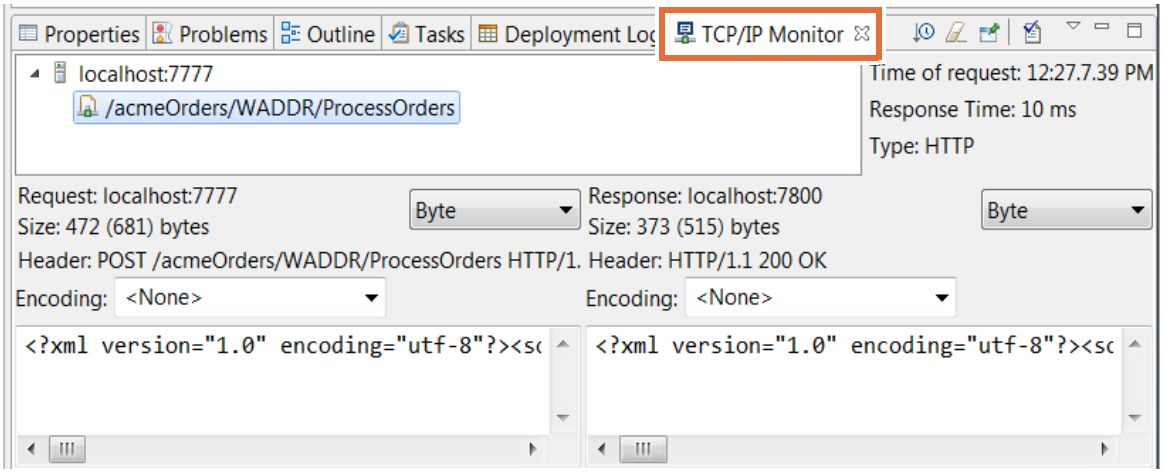
__ a. In the Test Client view, click the **Enqueue** operation In the Message Flow Test Events window.

__ b. Click **Send Message**.

The application runs again. Another message is displayed in the **Message Flow Test Events** window, showing that another message was enqueued to the SOAPSAMPLE_IN queue. On this test, the TCP/IP Monitor also captured the message traffic.

__ 8. Review the web service request and response messages in the TCP/IP Monitor.

__ a. Click the **TCP/IP Monitor** tab.

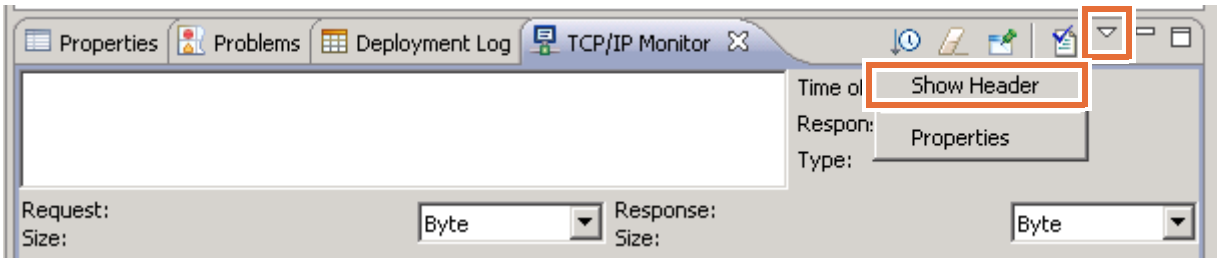


In the top part of the TCP/IP Monitor, you see the monitored port (7777) and the URL that is captured. The lower left section shows the request message (on port 7777), and the lower right section shows the response message (on port 7800).

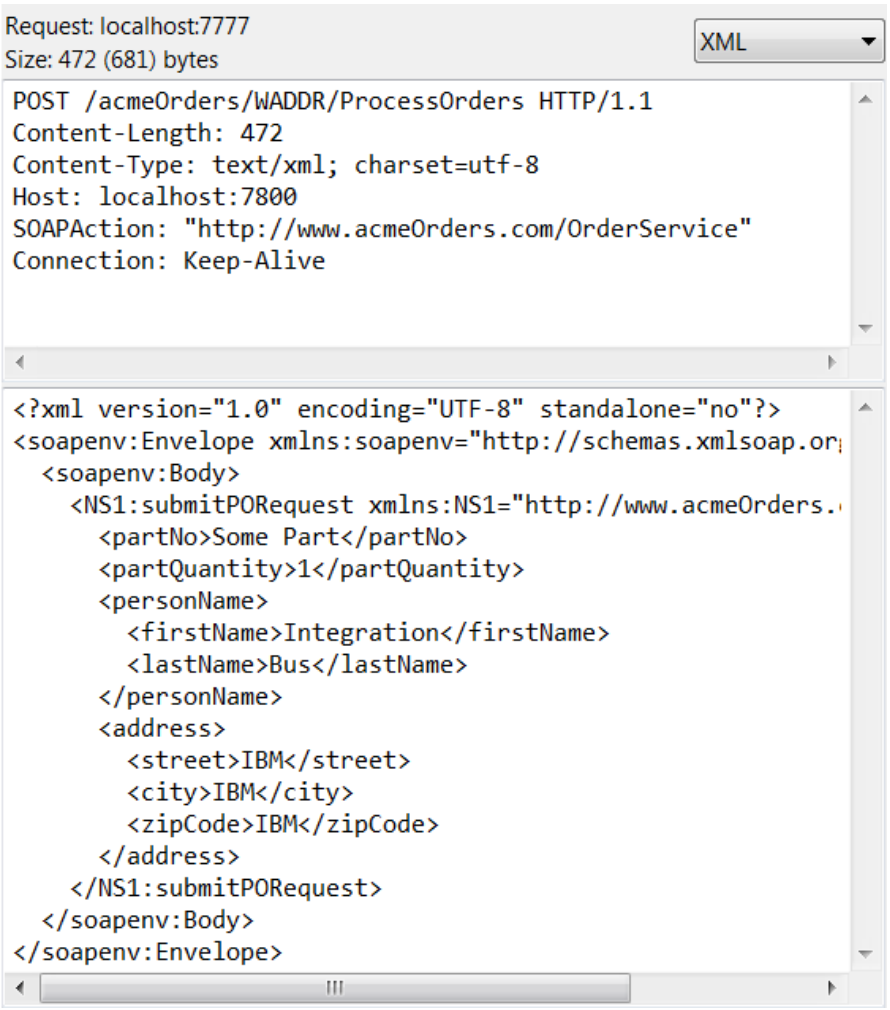
__ b. Change the display format to XML by clicking the field that currently shows **Byte** and selecting **XML**. Change this display setting for both the request message and the response message.

__ c. Maximize the viewing space by double-clicking the **TCP/IP Monitor** tab.

__ d. Enable the message header view. Click the **View menu** icon, and then click **Show Header**.

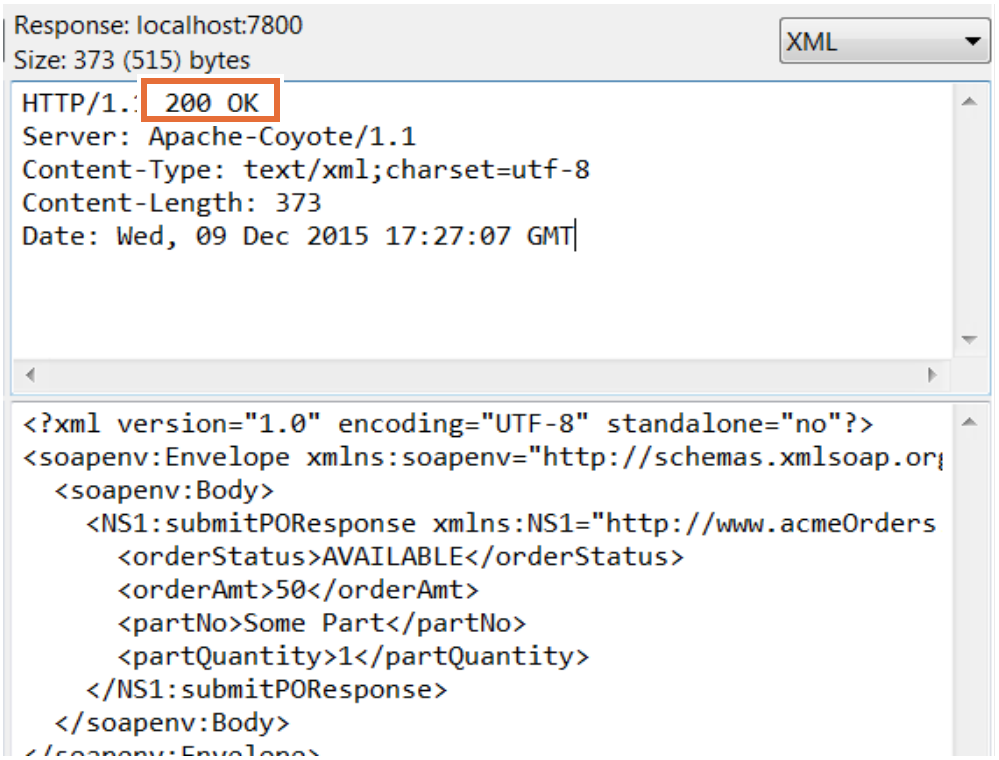


__ e. Review the request message with its header.



The SOAPAction header is automatically set to `http://www.acmeOrders.com/OrderService`. This value came from the WSDL file for this operation.

__ f. Review the response message with its header.



Along with the body of the response message, observe the HTTP **200** status code, which indicates the request was successful.

__ 9. Restore the **TCP/IP Monitor** pane to its normal size by double-clicking the tab.

Part 4: Set the reply message path with WS-Addressing

When used with a SOAP Request node, WS-Addressing allows a reply to be sent back to a different web service client as specified in the WS-Addressing properties.

In this part of the exercise, you modify the application to use WS-Addressing for both the provider and the consumer message flows.

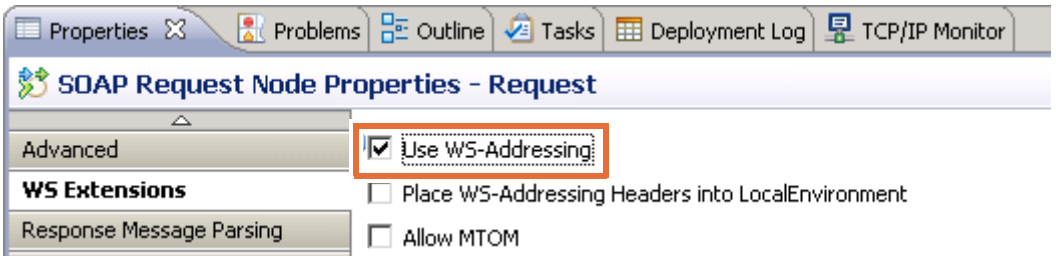
__ 1. Configure the web service consumer flow for WS-Addressing.

__ a. Double-click **SOAPNodesSampleConsumerFlow.msgflow** to open it in the Message Flow editor if it is not already open.

__ b. Double-click the **Invoke_submitPO** subflow to open the subflow in the Message Flow editor.

__ c. Click the SOAP Request node **Request** to display the node **Properties** view.

__ d. On the **WS Extensions** properties tab, click **Use WS-Addressing** to enable this option.

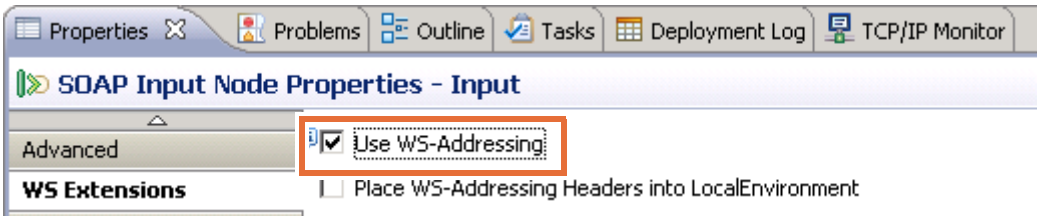


__ e. Save the subflow and then close the Message Flow editor tab.

__ 2. Configure the web service provider flow.

__ a. If the **SOAPNodesSampleProviderFlow** for the message provider is not already open in the Message Flow editor, open it now.

__ b. On the SOAP Input node that is named **Input**, enable the **Use WS-Addressing** property on the **WS Extensions** tab.



__ 3. Save the flow and then close the Message Flow editor tab.

__ 4. Rebuild and redeploy the BAR file.

__ a. If the **SOAPNodesSampleBAR.bar** file in the BAR File editor is not already open, open it now.

__ b. Click the **Build** icon in the BAR File editor toolbar.

__ c. If you receive messages about modified resources, click **Yes** to save them.

__ d. When the **Operation completed successfully** message is displayed, click **OK**.

__ e. Redeploy the BAR file to the integration server that is named **default** on the **TESTNODE_iibadmin** integration node.

__ f. Wait for the deployment operation to complete. Verify that the deployment was successful by reviewing the **Deployment** tab for a success message for the current date and time.

__ 5. Rerun the application test.

In the Integration Toolkit Test Client, click the **Enqueue** and then click **Send Message**.

The application runs again. Another message is displayed in the Message Flow Test Events window, showing that another message was enqueued to the SOAPSAMPLE_IN queue.

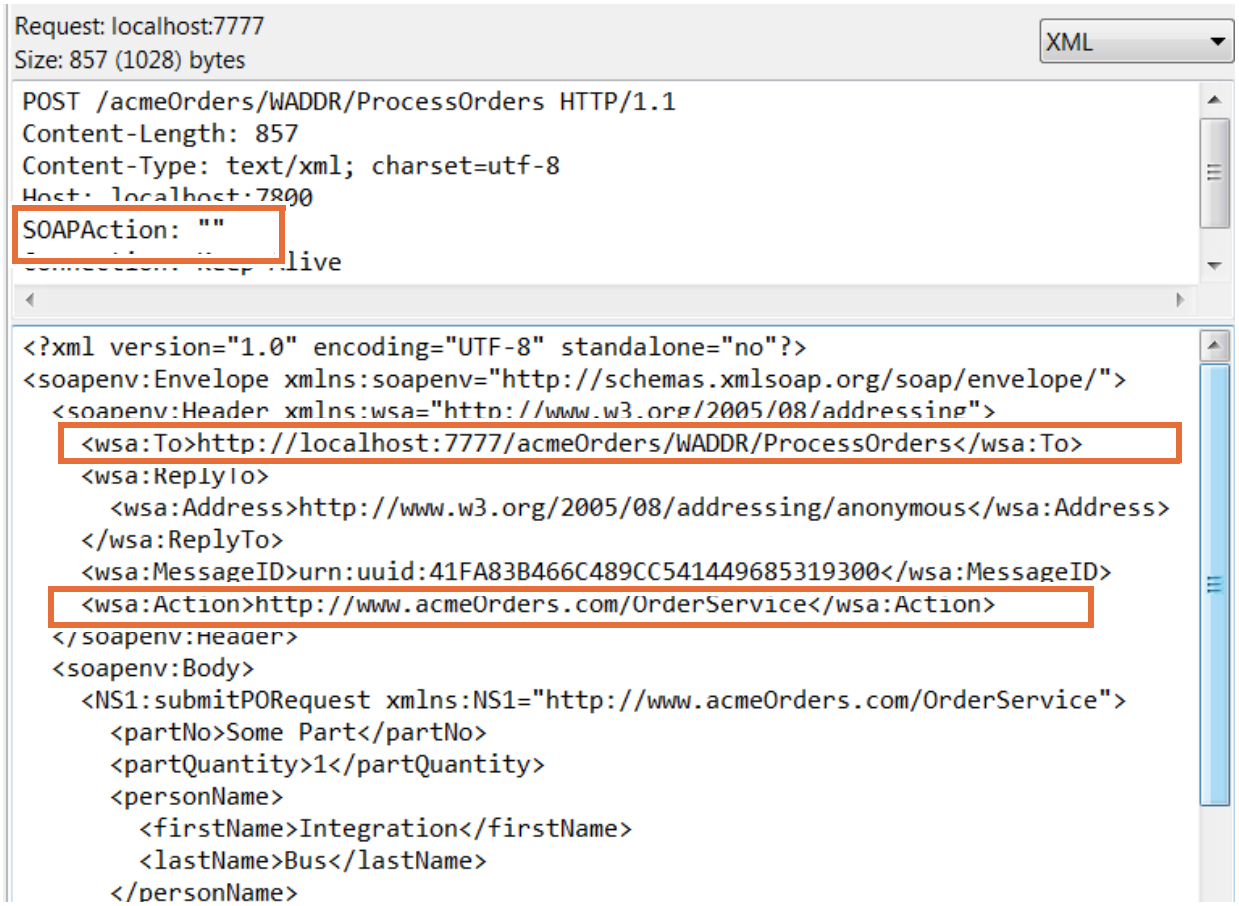
__ 6. On this test, the TCP/IP Monitor also captured the message traffic. Review the TCP/IP Monitor.

__ a. Click the **TCP/IP Monitor** tab.

__ b. Change the display format to XML by clicking the field that currently shows **Byte** and selecting **XML**. Change the display format for both the request message and the response message.

__ c. If necessary, maximize the TCP/IP Monitor window by double-clicking the **TCP/IP Monitor** tab.

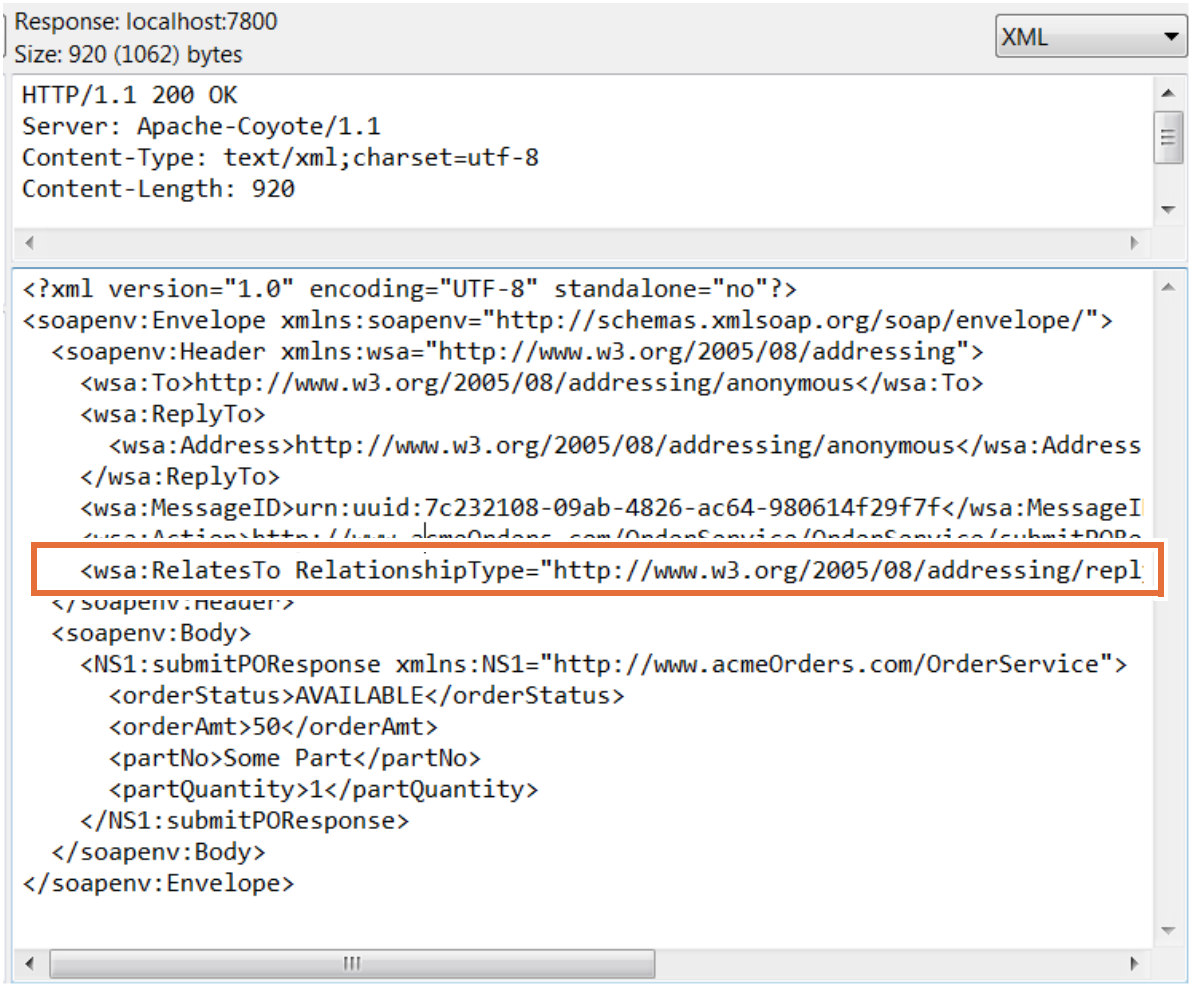
__ d. Review the request message with its header.



This message differs in several ways from the message that was sent before WS-Addressing was enabled:

- The value for **SOAPAction** is now null.
- WS-Addressing fields are listed in the soapenv:Header section of the message. The characters wsa prefix these fields.
- The wsa:Action field was not specified in the WSDL (unlike the **SOAPAction** field, which was specified), so the value was calculated automatically, based on the operation that was started.

__ e. Review the response message with its header.



- As you saw with the request message, the response message contains a number of **wsa** fields because of WS-Addressing.
- Although not fully visible on the screen capture shown here, the value of the **wsa:MessageID** field in the request message was automatically copied into the WS-Addressing **RelatesTo** field in the reply message. Review this correlation between your messages.

Exercise cleanup

__ 1. Stop the TCP/IP Monitor.

__ a. In the **TCP/IP Monitor** view toolbar, click the **View Menu** icon (the downward-pointing triangle in the view toolbar).

__ b. Click **Properties**.

__ c. Click the row that you added, and then click **Stop**. The **Status** column changes from **Started** to **Stop**.

__ 2. Close the TCP/IP Monitor view.

__ 3. Close all open editor windows in the Integration Toolkit. You do not need to save any changes, in case you are prompted.

__ 4. In the **Integration Nodes** view, right-click the integration server that is named **default** on the TESTNODE_iibadmin, and then click **Delete > All flows and resources**. When the **Confirm Deletion** dialog box is displayed, click **OK**.

End of exercise

Exercise review and wrap-up

In the first part of the exercise, you imported a project interchange that contains a web services consumer and provider flow. You then reviewed the message flow so that you have a good understanding of its operation.

In the second part of this exercise, you deployed and tested the web services consumer and provider message flows. You also verified the integration server HTTPConnector port.

In the third part of this exercise, you used the Integration Toolkit TCP/IP Monitor to view message data as it passed through the SOAP/HTTP message transport.

In the fourth part of this exercise, you modified the application to use WS-Addressing in both the provider and the consumer message flows. This modification was to send a reply to a different web service client as specified in the WS-Addressing properties.

Having completed this exercise, you should be able to:

- Import a WSDL file into a message model
- Provide a message flow as a web service that uses SOAP/HTTP
- Start a SOAP/HTTP web service asynchronously
- Test a SOAP/HTTP message flow
- Implement WS-Addressing
- Use the TCP/IP Monitor to view the data as it passes through the SOAP/HTTP message transport

Exercise 4. Creating an integration service

What this exercise is about

In this exercise, you create and implement an integration service.

What you should be able to do

After completing this exercise, you should be able to:

- Create a container for an integration service
- Design the operations, bindings, and messages for a service interface in a graphical editor
- Implement a service operation with a database SQL call
- Deploy and test an integration service by using the IBM Integration Toolkit Flow exerciser
- Export the WSDL document that describes the service by using the IBM Integration web user interface

Introduction

In IBM Integration Bus, an integration service is a specialized application that acts as a container for a web service solution. Integration services contain message flows to implement web service operations. A Web Service Description Language (WSDL) document defines the interface for the web service.

In this exercise, you follow a top-down design and create a web service interface in IBM Integration Toolkit. You specify the operations, message parts, and web service bindings in a standard WSDL document.

In the second part of the exercise, you develop a message flow to implement the web service operation that is defined in the WSDL document. You retrieve information from a data source and construct a web service response message with the mapping node.

In the last part of the exercise, you deploy and test the service with the Integration Toolkit.

Requirements

- IBM Integration Bus V10 and IBM Integration Toolkit.
- The IBM Integration Bus development integration node (TESTNODE_iibadmin).

- IBM DB2.
- The DB2 EMPLOYEE table in the SAMPLE database that is created by running the `DB2CREATE.cmd` file in the `C:\labfiles\Lab04-IntSvcs\resources` directory. This database is already configured on the lab image for this course.
- Lab files in the `C:\labfiles\Lab04-IntSvcs` directory.

Exercise instructions

Exercise setup

This exercise references the DB2 EMPLOYEE table in the SAMPLE database. The database is already set up for you on the lab image but you must create the JDBC configurable service that defines the JDBC connections to the DB2 SAMPLE database for the integration node TESTNODE_iibadmin. You must also create the database definition file that is used by the integration service.

__ 1. Open an IBM Integration Bus Command window.

__ 2. Change directories to the `C:\labfiles\Lab04-IntSvcs\resources` directory. Type:

```
cd C:\labfiles\Lab04-IntSvcs\resources
```

__ 3. Run the `ConfigureJDBC` command to create the JDBC configurable service for the SAMPLE database. Type:

```
ConfigureJDBC TESTNODE_iibadmin SAMPLE
```

__ 4. Run the `mqsicvp` command to verify access to the database from the integration node. Type:

```
mqsicvp TESTNODE_iibadmin -n SAMPLE
```

__ 5. In the Integration Toolkit, discover the database definition for the DB2 database SAMPLE, in the ADMINISTRATOR schema, and store it in the DB_DATA_PROJECT.

__ a. Click **File > New > Database Definition**. The **New Database Definition File** window opens.

__ b. To the right of the **Data design project** field, click **New** to create a data design project.

__ c. For the **Project name**, type `DB_DATA_PROJECT` and then click **Finish**.

__ d. On the **Create a data design project** page, click **Next**.

__ e. In the **Select Connection** page, click **New**.

__ f. In the **Properties** pane, for **Database**, type: `SAMPLE` (if it is not already entered)

__ g. For **User name**, type: `Administrator`

__ h. For **Password**, type: `web1sphere`

__ i. Click **Save password**.

__ j. Click **Test Connection** to ensure that the JDBC connection is tested.

__ k. A “Connection succeeded” message should be displayed. Click **OK**.

__ l. Click **Finish**. The **New Database Definition File** window displays the connection information for the SAMPLE database.

__ m. Click **SAMPLE1**, and then click **Next**.

__ n. Select **ADMINISTRATOR** and then click **Finish**.

Part 1: Create an integration service

In this part of the exercise, you create an integration service to hold the components of the web service.

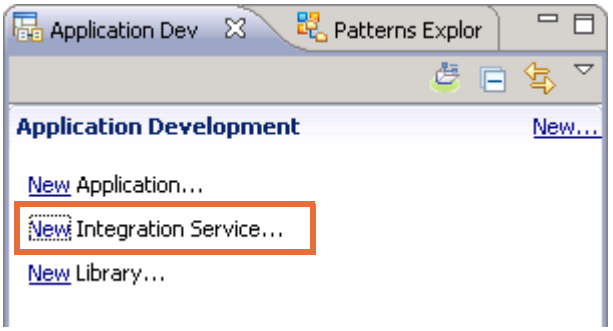
__ 1. If the Integration Toolkit is not already running, start it now.

__ 2. Switch to a new workspace in the Integration Toolkit.

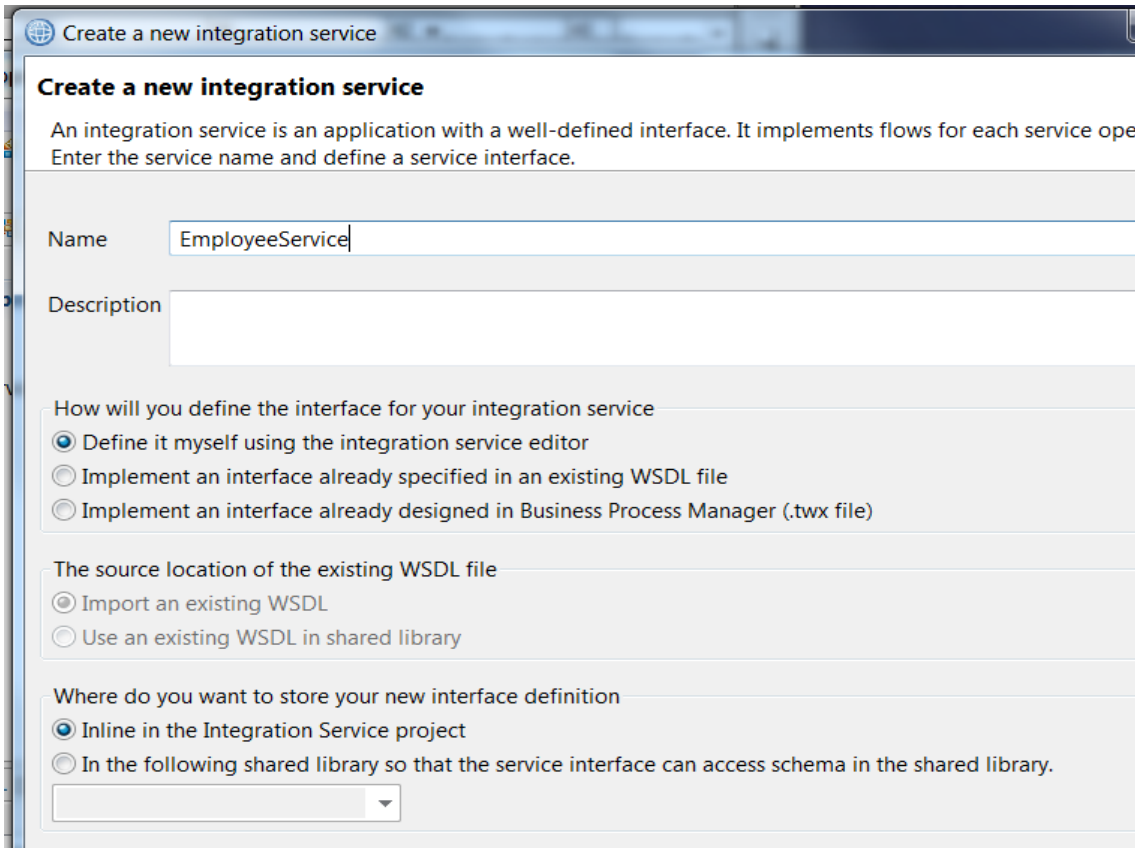
__ 3. Close the **Welcome** view to go to the **Integration Development** perspective.

__ 4. Create an integration service that is named **EmployeeService**.

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



__ b. In the wizard, enter `EmployeeService` as the integration service name.



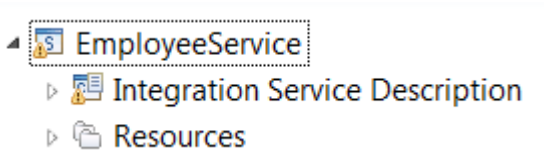
__ c. Accept the default settings for the other options on this page and click **Finish**.

__ 5. Examine the integration service project structure.

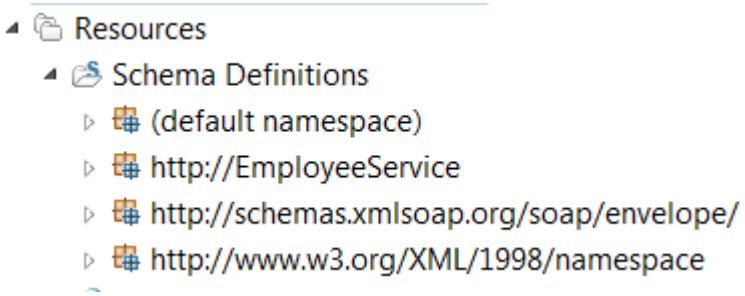
In the **Application Development** view, expand **Employee Services** components.

Information

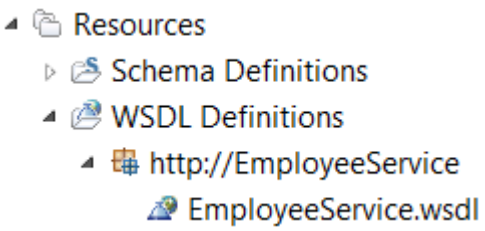
The integration service project has two main parts: the service description, and the resources that implement the service:



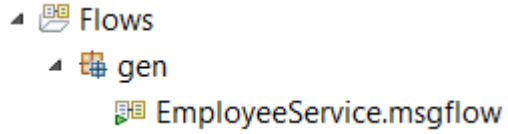
- **Integration Service Description** is a shortcut to the WSDL document that defines the service interface.
- **Resources > Schema Definitions** collects all of the XML schema definitions that the WSDL document uses.



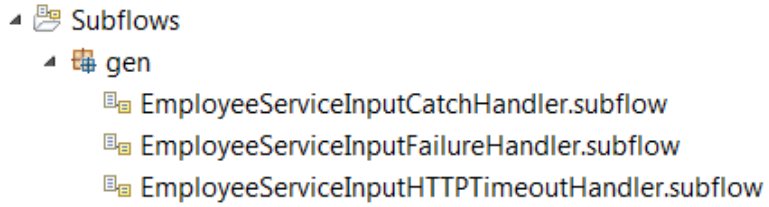
  - XML elements that are not explicitly linked to an XML namespace belong to the **default namespace**.
  - **`http://EmployeeService`** is the namespace for the Employee Service WSDL document. You can change the namespace declaration in the WSDL document.
  - **`http://schemas.xmlsoap.org/soap/envelope`** is the namespace for XML elements that are defined in the SOAP specification V1.1. These XML elements define the body, envelope, fault, and header parts of a web service SOAP message.
  - **`http://www.w3.org/XML/1998/namespace`** defines the namespace declaration in the WSDL document.
- **Resources > WSDL Definitions** stores the EmployeeService web service interface as a WSDL document.



- **Resources > Flows** contains the message flows that implement web service operations. You implement one flow for each operation in the service interface.



- **Resources > Subflows** contain portions of a message flow. By default, the integration service wizard creates three subflows:



  - A Catch error handler
  - A Failure error handler
  - An HTTP timeout error handler

__ 6. In this exercise, you start your design with a top-down approach and design a service interface first.

Import the `Employee.xsd` XML schema document to use the data types in your service interface.

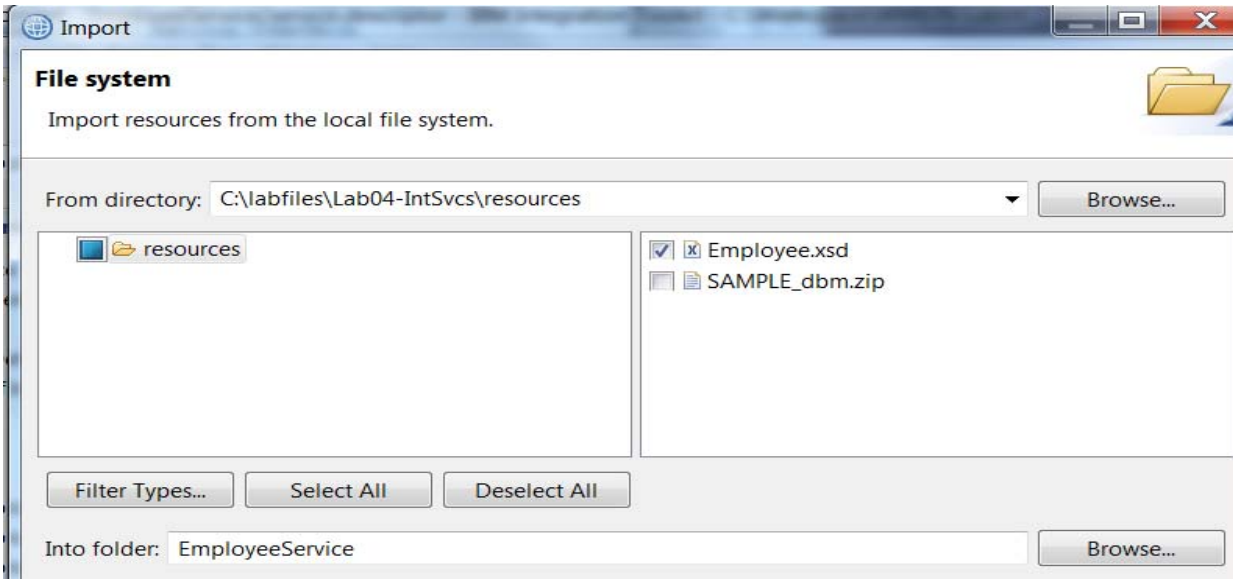
__ a. From the main menu, click **File > Import**.

__ b. In the Import wizard, click **General > File System** and then click **Next**.

__ c. In the **Import** dialog box, click **Browse** for the **From Directory** field, go to the `C:\labfiles\Lab04-IntSvcs\resources` directory, and click **OK**.

__ d. Click **Employee.xsd**.

__ e. Verify that the **Into folder** field is set to **EmployeeService**.



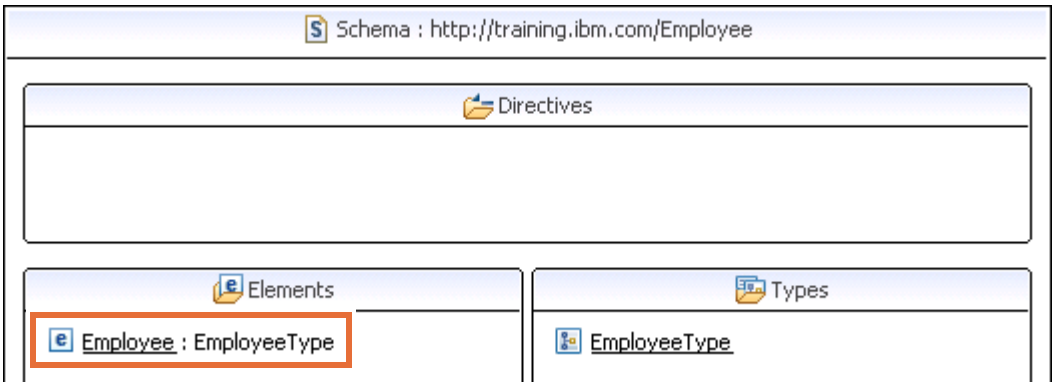
__ f. Click **Finish**.

__ 7. Examine the Employee XML schema that you imported in step 6.

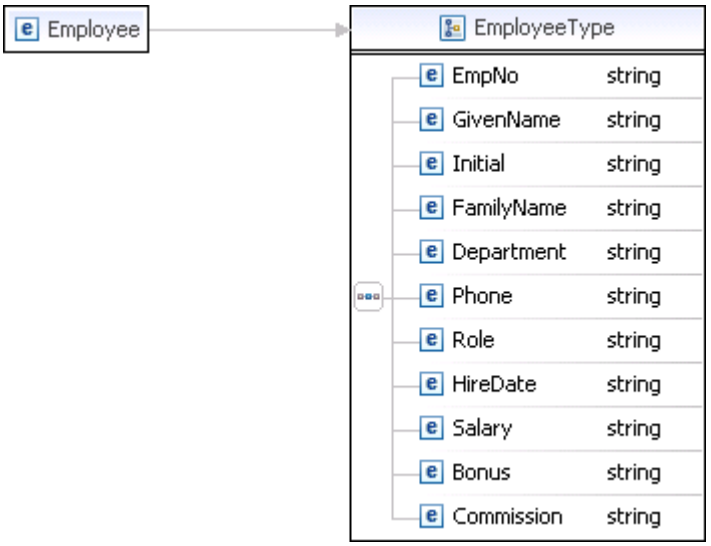
__ a. In the **Application Development** view, expand **EmployeeService > Resources > Schema Definition > http://training.ibm.com/Employee**.

__ b. Double-click the **Employee.xsd** file to open it in the XML schema editor.

__ c. In the XML schema editor, double-click the **Employee** XML element.



__ d. Examine the structure of the Employee XML element.



`EmployeeType` defines the structure of the `<Employee>` XML element. This XML complex type holds a set of properties that represent an employee record.

In this exercise, you design and implement a web service operation that retrieves an employee record from a database. The web service response message returns the record fields as an `<Employee>` element.

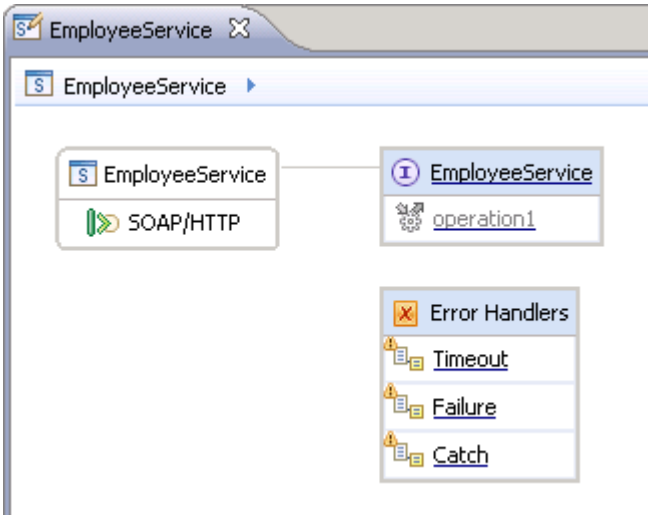
__ e. Close the XML schema editor.

__ 8. Examine the **Integration Service Description**.

__ a. If the **EmployeeService** service description is not already open in the editor view, open it by double-clicking **EmployeeService > Integrated Service Description** in the **Application Development** view.

__ b. The **EmployeeService** integrated service description has two parts: a visual diagram of the service implementation on the **Service** tab, and a service interface on the **Interface** tab.

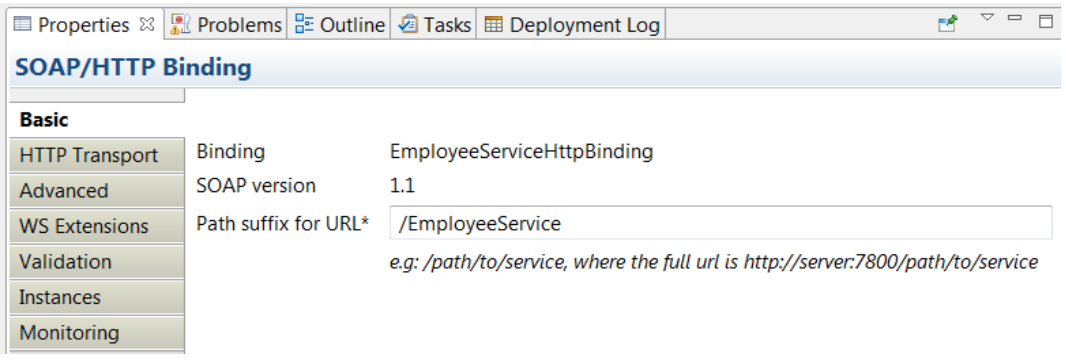
Examine the **EmployeeService** service diagram by clicking the **Service** tab in the editor.



- On the left side of the service implementation, the **EmployeeService** integration service communicates with a web service client with SOAP messages over an HTTP transport (SOAP/HTTP).
- On the right side of the service implementation, the **EmployeeService** service interface defines the web service operations that the service provides. The only operation available is **operation1**.
- **Error Handlers** interrupt the normal message flow on error conditions. The service integration wizard creates three error handlers: an HTTP timeout handler, a failure handler, and a catch handler.

__ c. Click the **SOAP/HTTP** service binding.

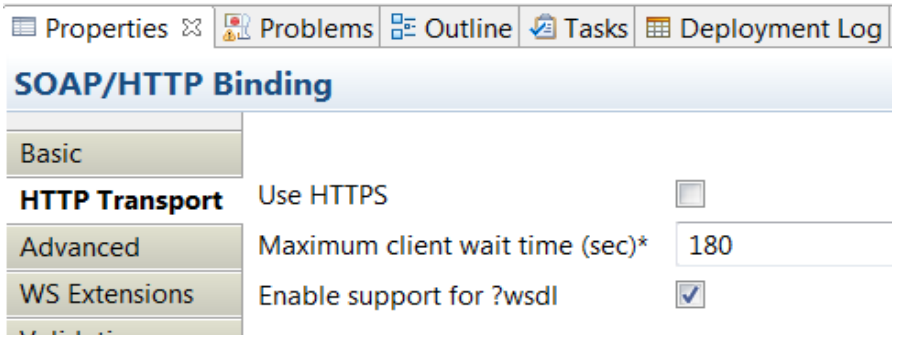
__ d. In the **Properties** view, click **Basic**.



The integration service binding supports SOAP specification version 1.1 as indicated by the **SOAP version** property.

The **Path suffix for URL** sets the web address for the service. For example, the path to the EmployeeService web service is `http://localhost:7800/EmployeeService`.

__ e. Click **HTTP Transport** in the **Properties** view to examine the transport properties.



- The default value for the **Maximum client wait time** is set to 180 seconds. After the wait period, the application runs the **Timeout** error handler.
- The **?wsdl** convention helps web service client developers retrieve the web service interface. For example, to download a copy of the EmployeeService WSDL document at run time, open a web browser and go to: `http://localhost:7800/EmployeeService?wsdl`

__ 9. Create a web service operation that is named `getEmployee` that accepts an employee XML element and returns an employee XML element in the reply message.

Note

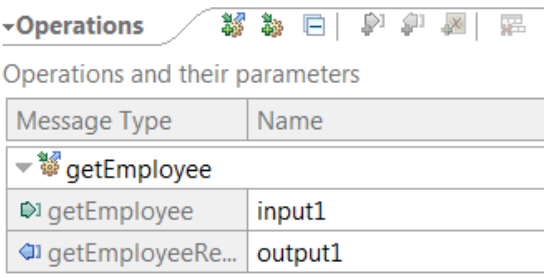
For this exercise, the `getEmployee` web service operation reads the `EmpNo` field only. However, you can modify the `getEmployee` operation to search the EMPLOYEE database table with other fields.

The suggested practice is to make the `getEmployee` interface as generic as possible to allow changes to the service implementation without affecting the service interface.

__ a. In the **EmployeeService** editor, switch to the **Interface** tab.

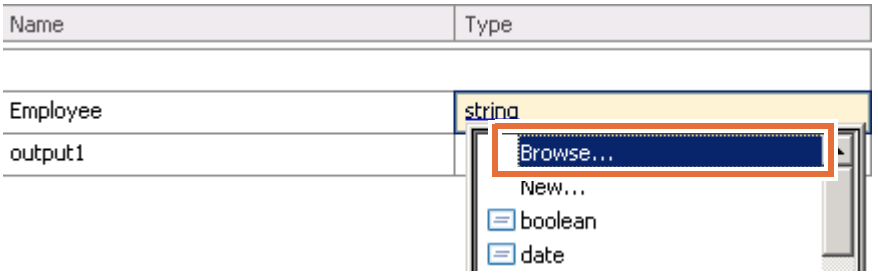
__ b. Change the name of the operation to `getEmployee` in the **Operations** section by double-clicking **operation1** and typing: `getEmployee`

The names of the operation input and output messages change to `getEmployee` and `getEmployeeResponse`.



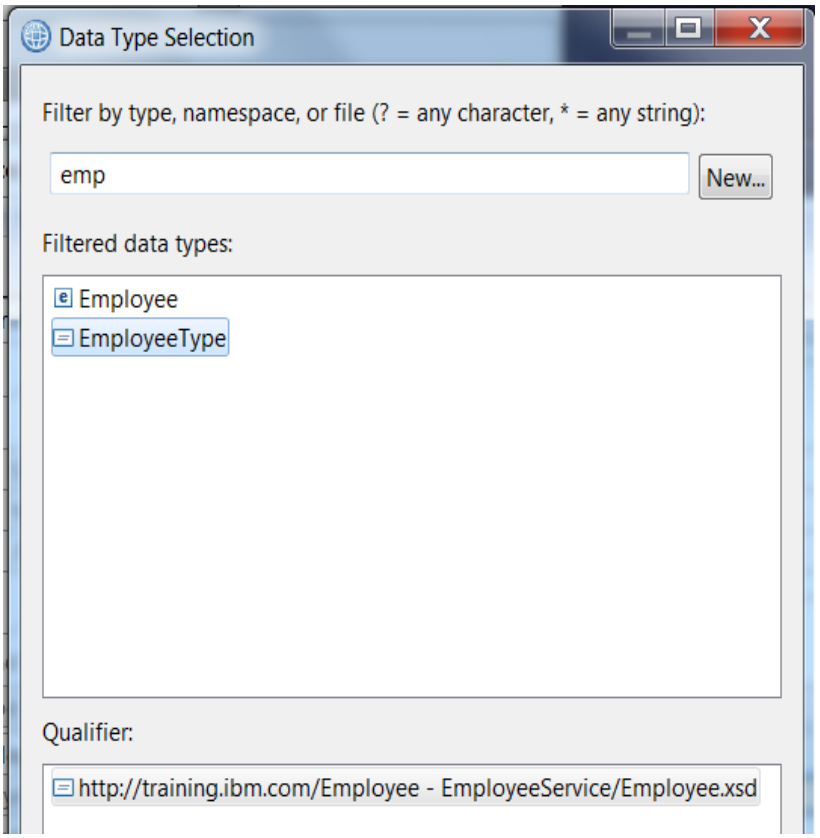
__ c. In the **getEmployee** input field, enter `Employee` for the input message name.

__ d. Select the **string** type in the input field and then click **Browse**.



__ e. In the **Data Type Selection** dialog box, enter `emp` as the filter by type setting.

__ f. Click **EmployeeType** as the data type for the input message and click **OK**.

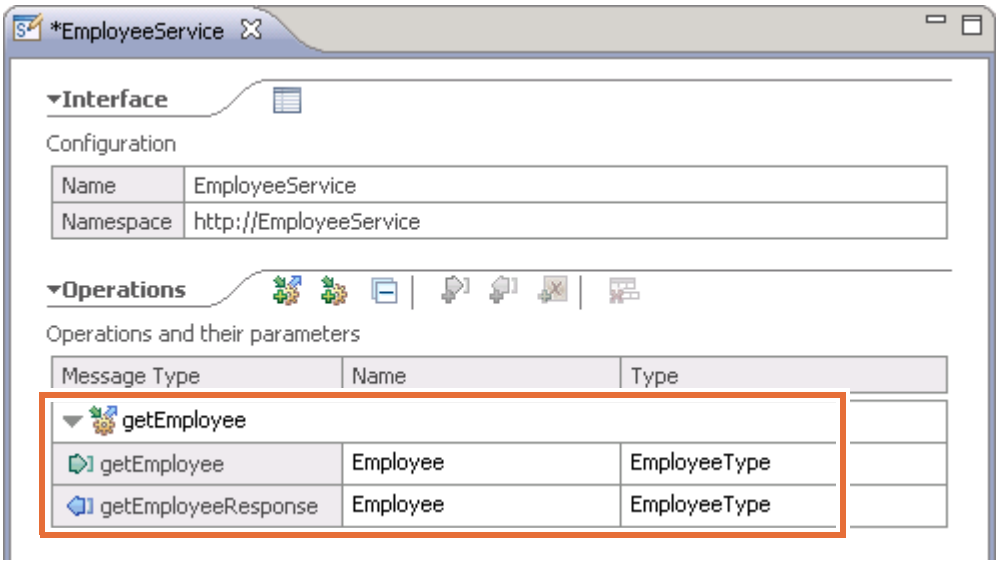


Note

If you do not see **Employee** or **EmployeeType** in the Data Type Selection list, close and then reopen the IBM Integration Toolkit.

__ g. Change the **getEmployeeResponse** message name to **Employee**.

__ h. Following the same procedure as steps d – f, set the **getEmployeeResponse** message type to **EmployeeType**.



__ i. Save the **EmployeeService** interface.

Part 2: Implement the EmployeeService message flow

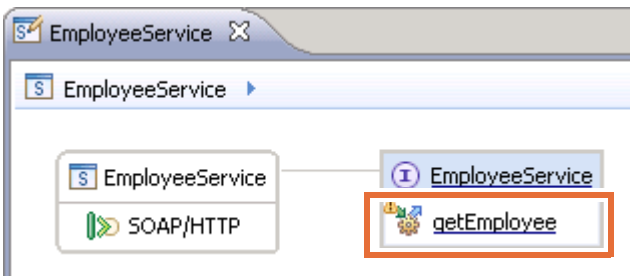
In this part of the exercise, you provide an implementation for the service interface that you defined in Part 1.

You complete the EmployeeService message flow to accept requests for the `getEmployee` web service operation. You use a Mapping node to retrieve an employee record from the DB2 SAMPLE database and write the results in the `getEmployeeResponse` web service output message.

__ 1. Open the **`getEmployee`** web service operation message flow.

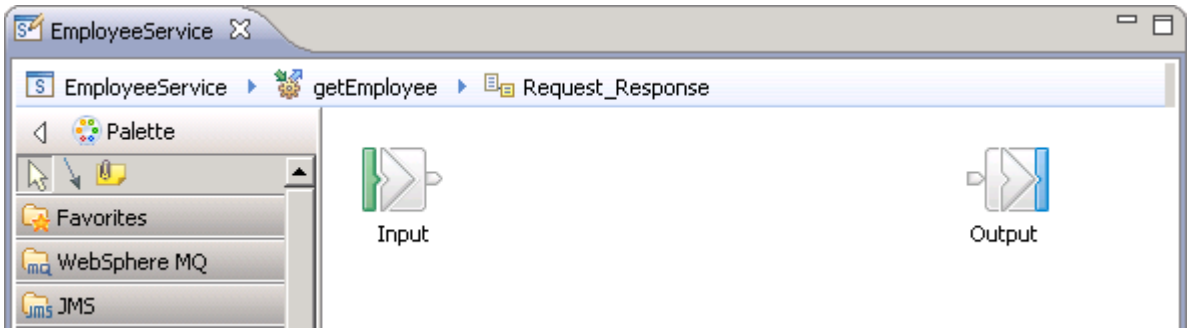
__ a. In the editor view for the EmployeeService, click the **Interface** tab.

__ b. Click the **`getEmployee`** web service operation to view the operation subflow.



Information

The **Request_Response** message flow implements the **getEmployee** service operation.



- The integration service defines a subflow for each service operation. In this case, the **getEmployee Request_Response** subflow represents the **getEmployee** operation.
- The **Input** node accepts the web service input message, **getEmployee**.
- The **Output** node expects the web service output message, **getEmployeeResponse**.

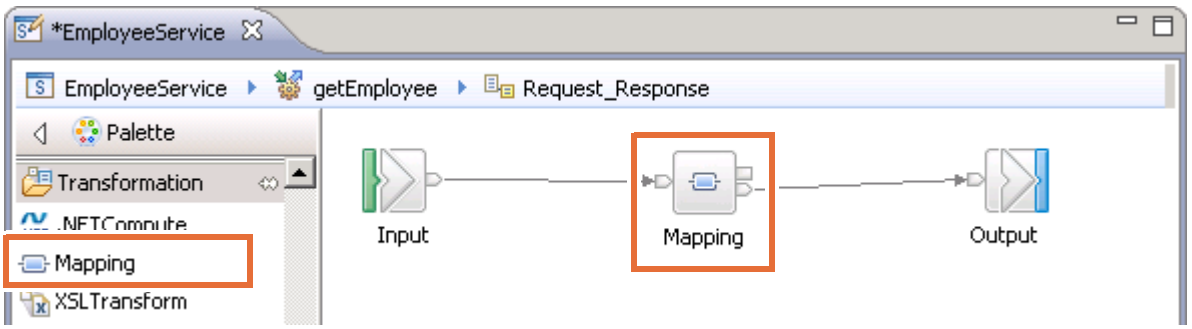
To complete the web service operation, finish the implementation of the message flow.

__ 2. Add a Mapping node between the Input and Output nodes.

__ a. Add a Mapping node (from the **Transformation** drawer) to the message flow canvas.

__ b. Connect the **Input** node **Out** terminal to the **Mapping** node **In** terminal.

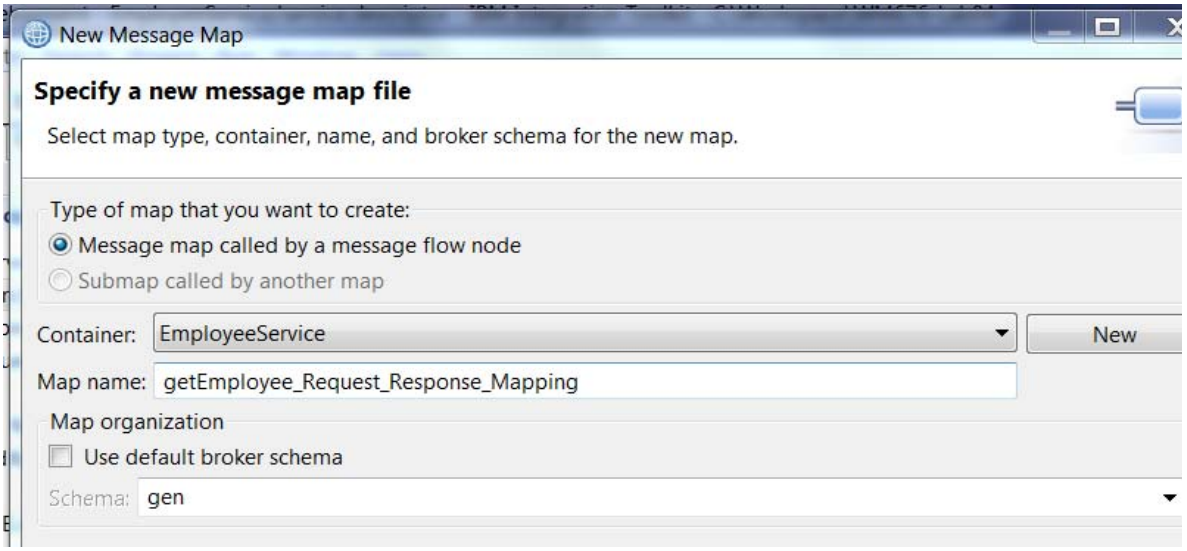
__ c. Connect the **Mapping** node **Out** terminal to the **Output** node **In** terminal.



__ 3. Create a map between the getEmployee WSDL input message and the getEmployeeResponse WSDL output message.

__ a. Double-click the **Mapping** node.

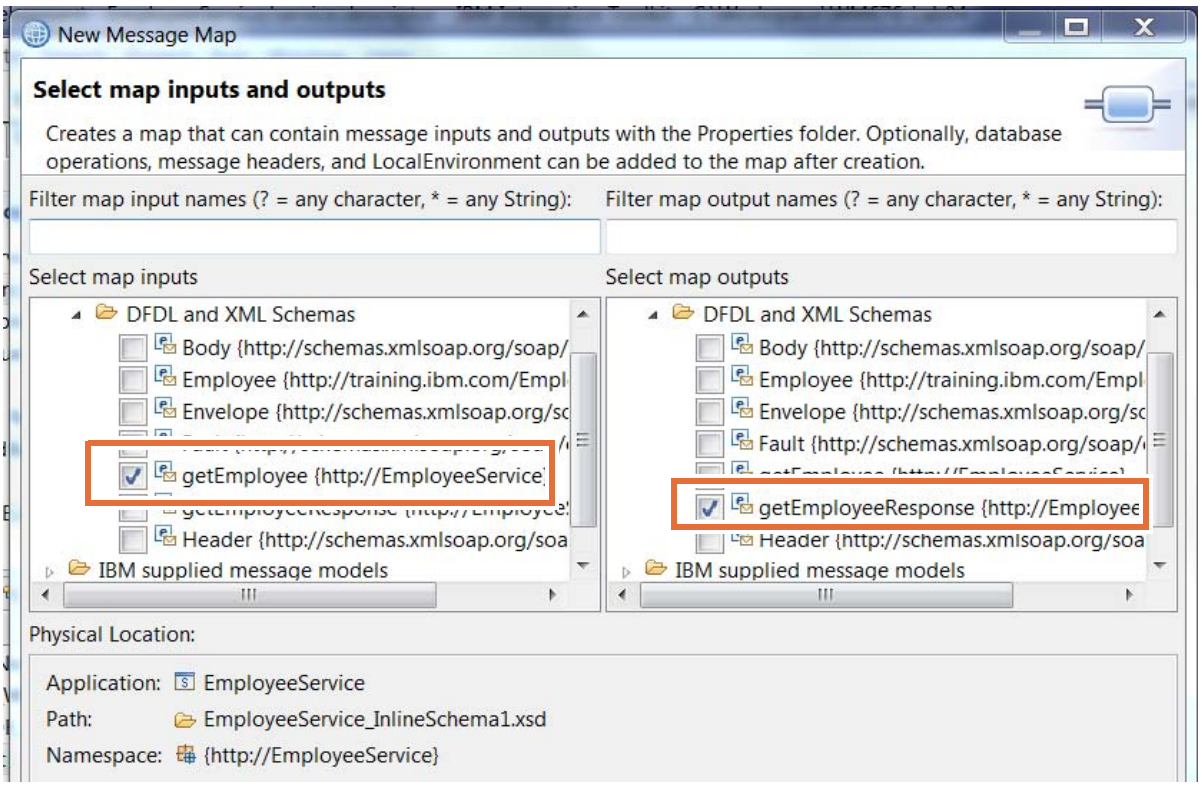
__ b. In the **New Message Map** wizard, leave the message map name to the default values.



__ c. Click **Next**.

__ d. In the **Select map inputs and outputs** page, select the web service input message **getEmployee** under the **EmployeeService > DFDL and XML Schemas** folder for the map input.

__ e. For the map output, select **getEmployeeResponse** under the **EmployeeService > DFDL and XML Schema** folder.



Note

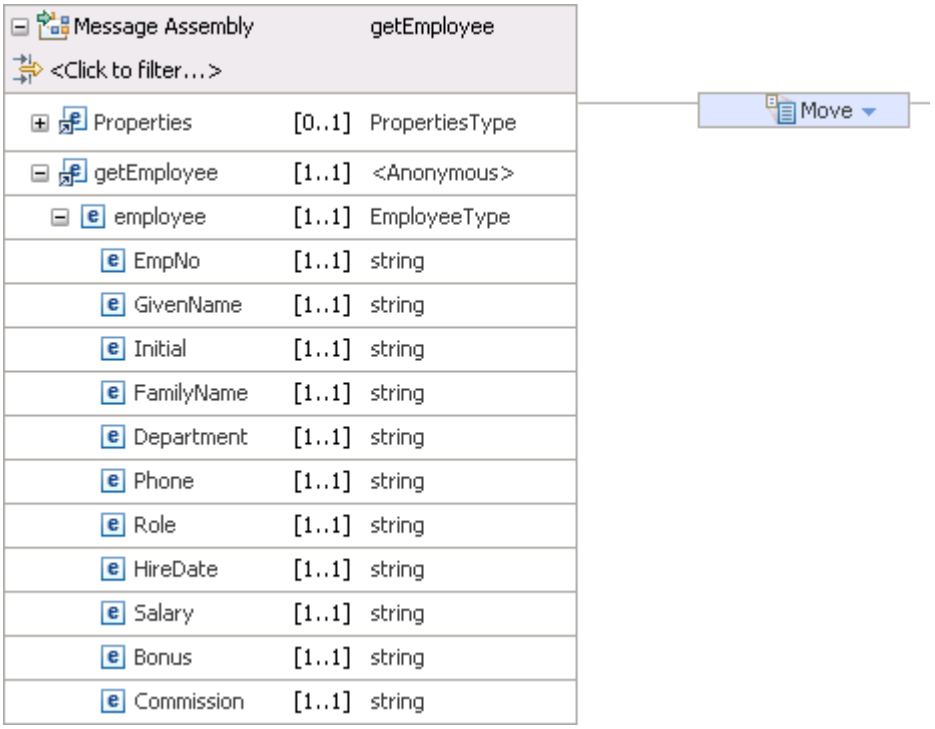
If you select the **Employee** XML element as the input message, you cannot access the web service SOAP headers. By selecting **getEmployee**, the mapping node has access to the entire WSDL input message: the SOAP envelope, header, and body.

__ f. Click **Next**.

__ g. Keep the default output domain of XMLNSC. Click **Finish**.

__ 4. Map the fields from the web service input message to the output message in the EmployeeService message flow.

__ a. In the Mapping editor, expand the **getEmployee** and **getEmployeeResponse** messages.

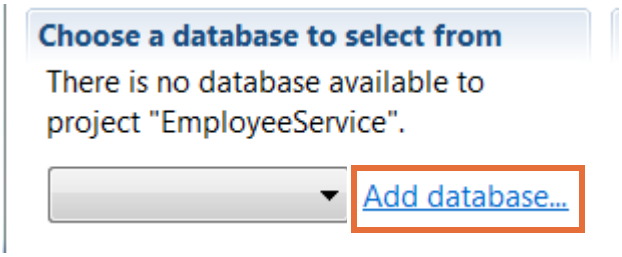


Information

The default mapping node behavior is to copy the entire contents of the WSDL input message into the output message. The **Properties** element in the message assembly stores the SOAP headers. The **getEmployee** and **getEmployeeResponse** elements holds the input message body and output message body.

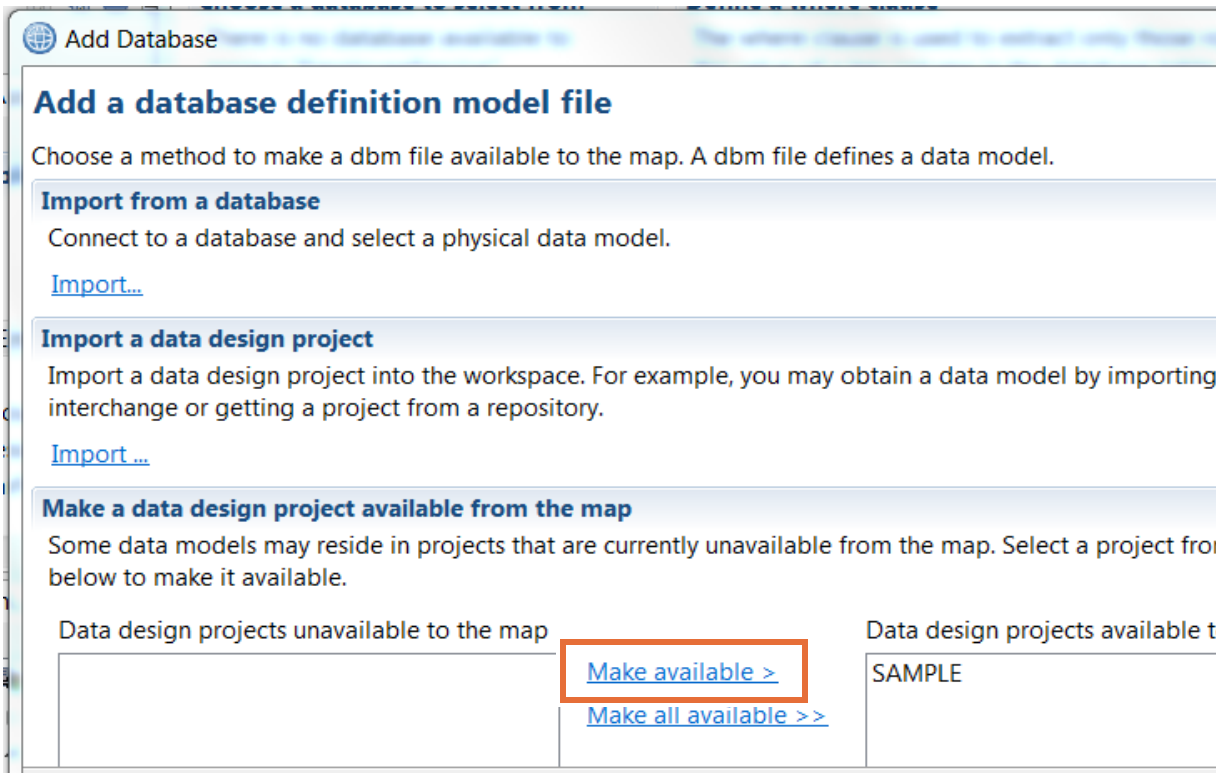
__ b. From the Map editor toolbar, click the **Select rows from a database** icon.

__ c. In the **New Database Select** dialog box, click **Add database** in the **Choose a database to select from** section.

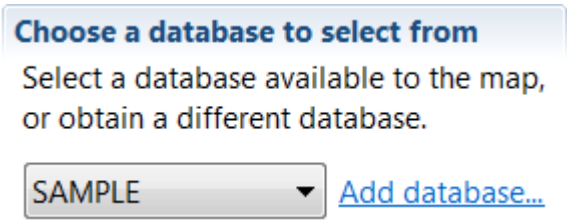


__ d. In the **Add a database definition file**, click the **DB_DATA_PROJECT** database definition file from the **Make a data design project available from the map**.

__ e. Click **Make available**.

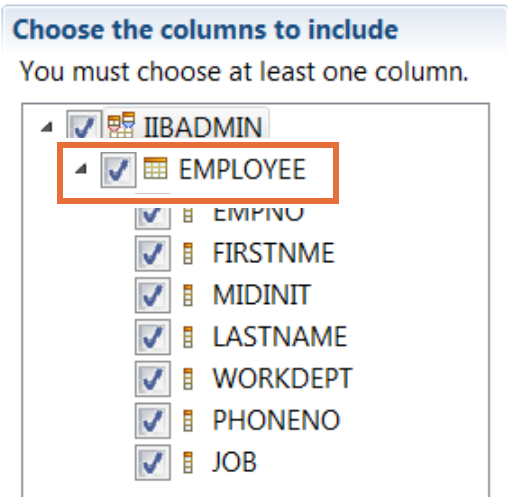


__ f. Click **OK**. Confirm that the wizard selects the SAMPLE data design project.



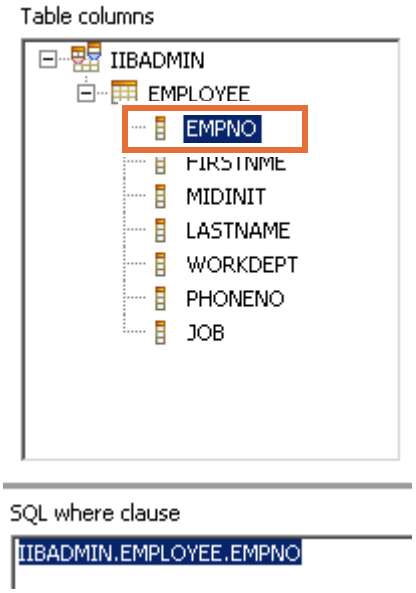
__ 5. Insert the EMPLOYEE table columns from the SAMPLE database to the output message.

__ a. In the **Choose columns to include** section, select the **EMPLOYEE** table.



__ b. In the **SQL where clause** section, delete the **1=1** statement.

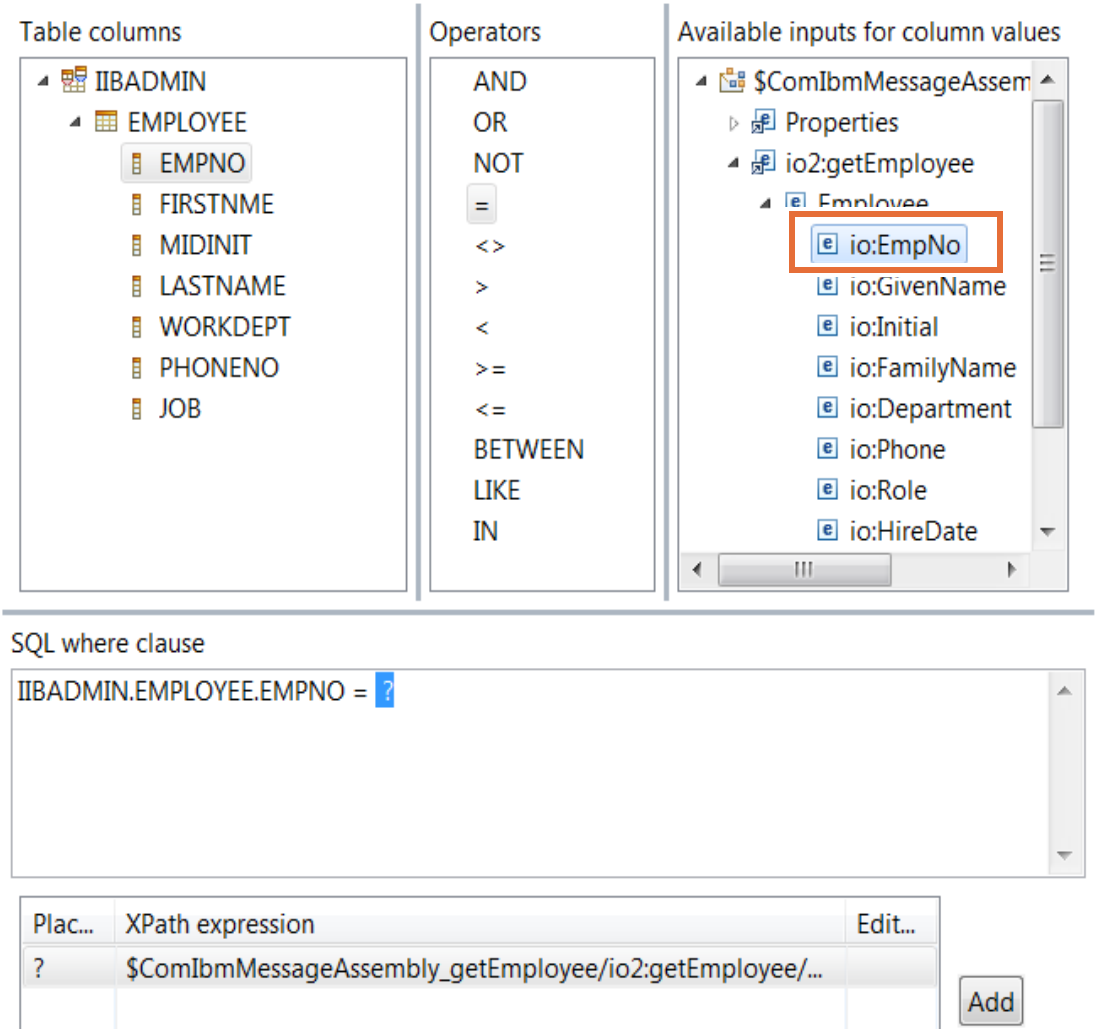
__ c. Double-click **EMPNO** from the table columns section.



The wizard writes `IIBADMIN.EMPLOYEE.EMPNO` into the SQL where clause.

__ d. Double-click **=** (equals operator) from the **Operators** column.

__ e. Double-click the **io:EmpNo** XML element under the **io2:getEmployee** element in the **Available inputs for column values** column.



Information

The New Database Select wizard creates the SQL statement:

```
SELECT IIBADMIN.EMPLOYEE.FIRSTNME, IIBADMIN.EMPLOYEE.MIDINT,
IIBADMIN.EMPLOYEE.LASTNAME, IIBADMIN.EMPLOYEE.WORKDEPT,
IIBADMIN.EMPLOYEE.PHONENO, IIBADMIN.EMPLOYEE.JOB WHERE IIBADMIN.EMPLOYEE.EMPNO
= ?
```

The mapping node replaces the placeholder (?) with the actual employee number specified by the XPath expression:

```
$ComIbmMessageAssembly_getEmployee/io2:getEmployee/Employee/io:EmpNo/
```

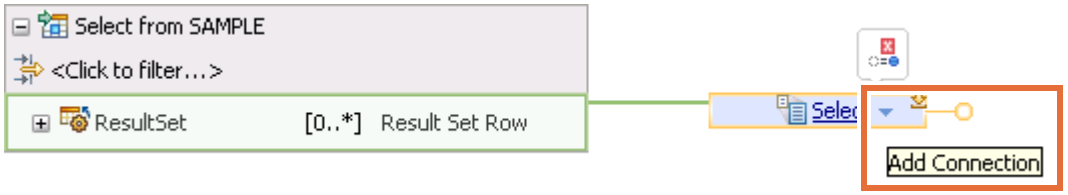
In summary, the database action retrieves an employee record that matches the employee number from the web service input message.

__ f. Click **OK**.

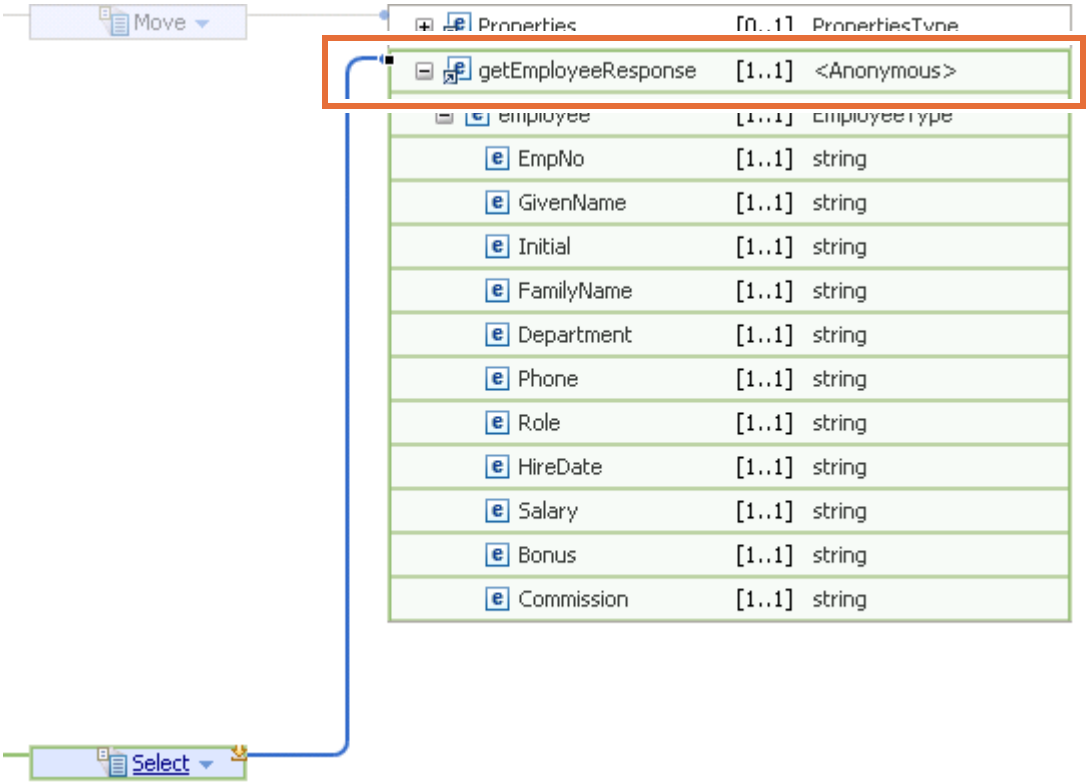
__ 6. Connect the **Select from SAMPLE** result set to the output message **getEmployeeResponse**.

__ a. Scroll down to the bottom of the mapping editor.

__ b. Click the **Add connection** marker to the right of the **Select** action.



__ c. Connect a wire from the Select action to the **Employee** XML element in the output message assembly.



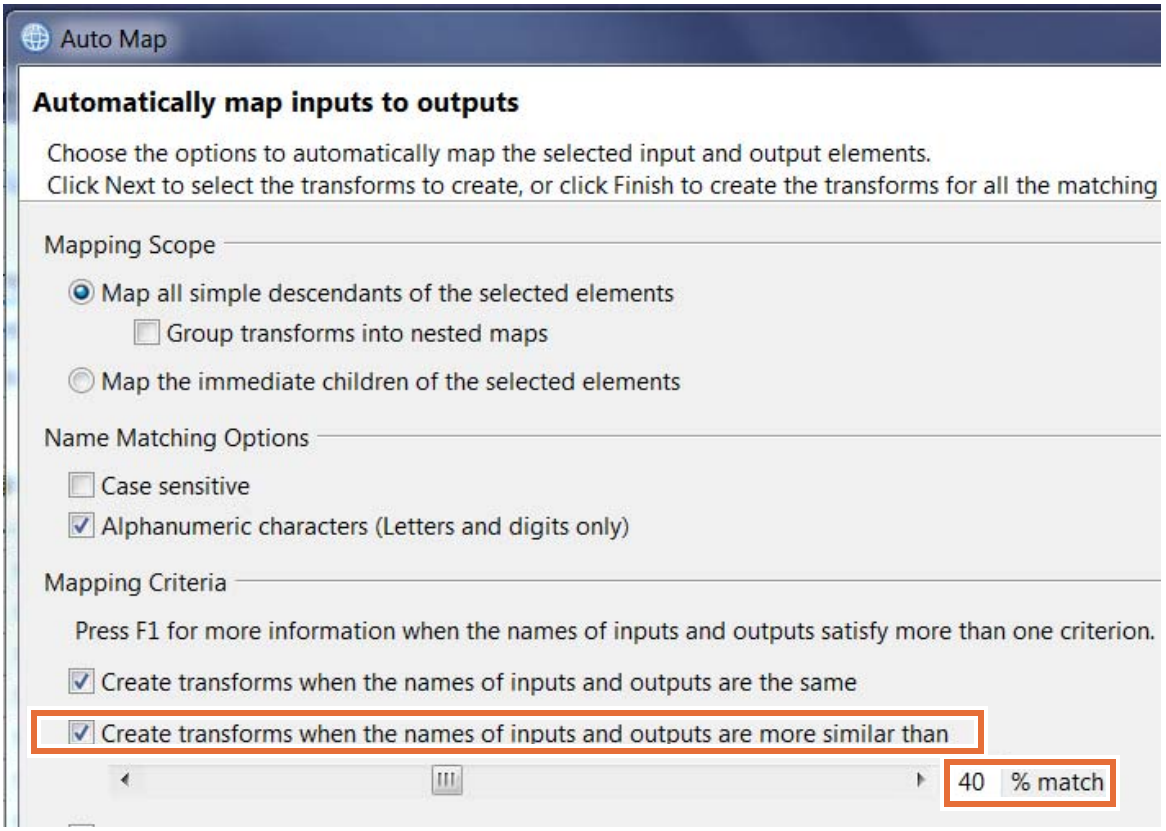
__ 7. Map the **Connect from SAMPLE** result set fields to the elements of **Employee** in the output message.

__ a. Click the **Select** action to open the nested message map.

__ b. Click the **Automap** icon in the toolbar.

__ c. In the AutoMap wizard, select **Create transforms when the names of inputs and outputs are more similar than**.

__ d. Enter `40` as the percentage match factor.



__ e. Click **Finish**.

__ 8. Complete the mapping between the SQL Select result set and the getEmployeeResponse message.

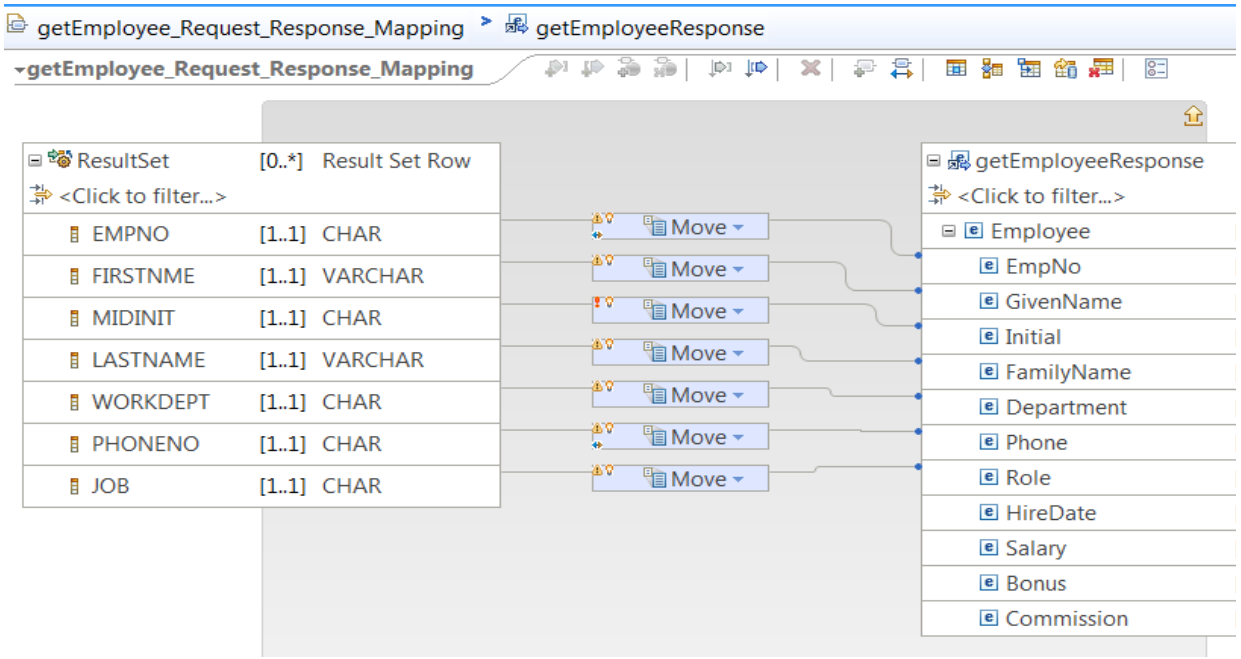
__ a. In the mapping editor, change the wire from **LASTNAME** on the input to **FamilyName** on the output.

__ b. Wire **FIRSTNME** on the input to **GivenName** on the output.

__ c. Wire **WORKDEPT** on the input to **Department** on the output.

__ d. Wire **JOB** on the input to **Role** on the output.

__ e. Confirm that all input elements have mappings to the result set.



__ f. Save and close the mapping editor.

Part 3: Deploy and test the web service

In this part of the exercise, you deploy and test the integration service by using the Integration Toolkit Flow exerciser.

You use the Flow exerciser to send a SOAP request message to the EmployeeService web service endpoint and then view the result of the web service call as an XML message.

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

__ 2. Select the integration server that is named **default** on the integration node that is named **TESTNODE_iibadmin** as the deployment location and then click **Finish**.

__ 3. Confirm that the EmployeeService integration service is deployed and running in the integration server.

__ 4. Retrieve the WSDL document that describes the service endpoint.

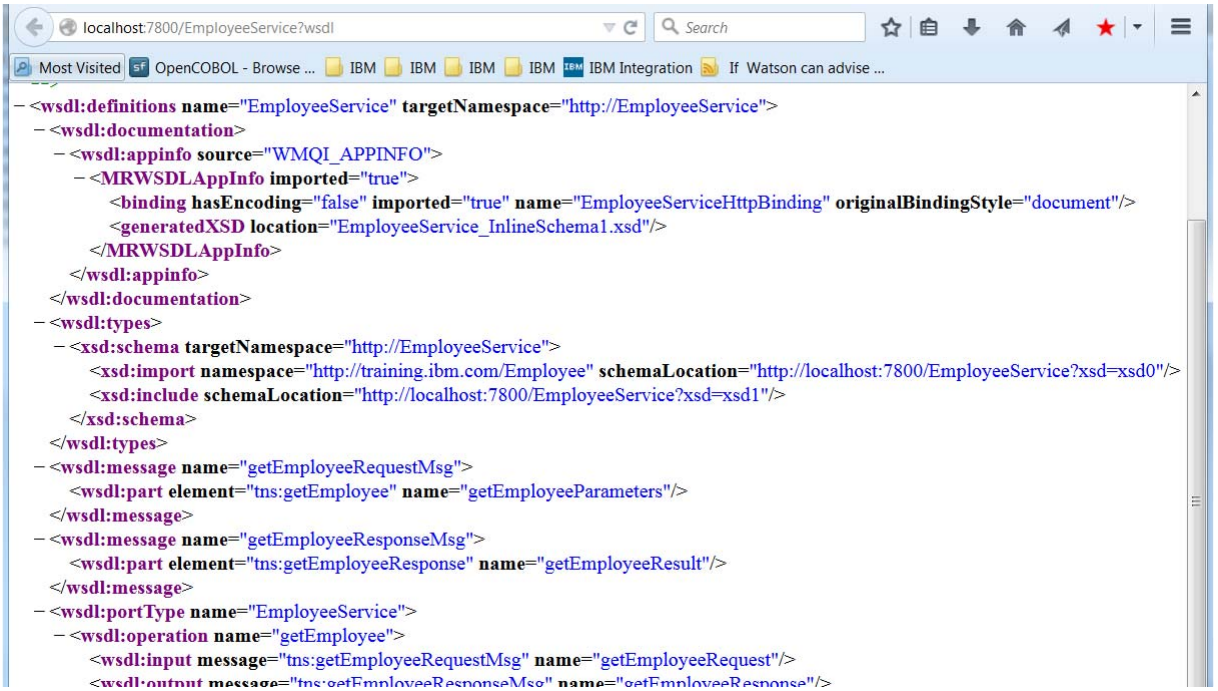
__ a. Open a web browser.

__ b. Enter `http://localhost:7800/EmployeeService?wsdl` in the address bar and then press **Enter**.

Information

You specified the web service address in the SOAP/HTTP binding **Path suffix for URL** property.

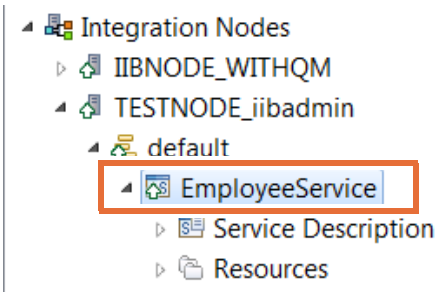
__ c. Confirm that the EmployeeService WSDL document appears in the browser.



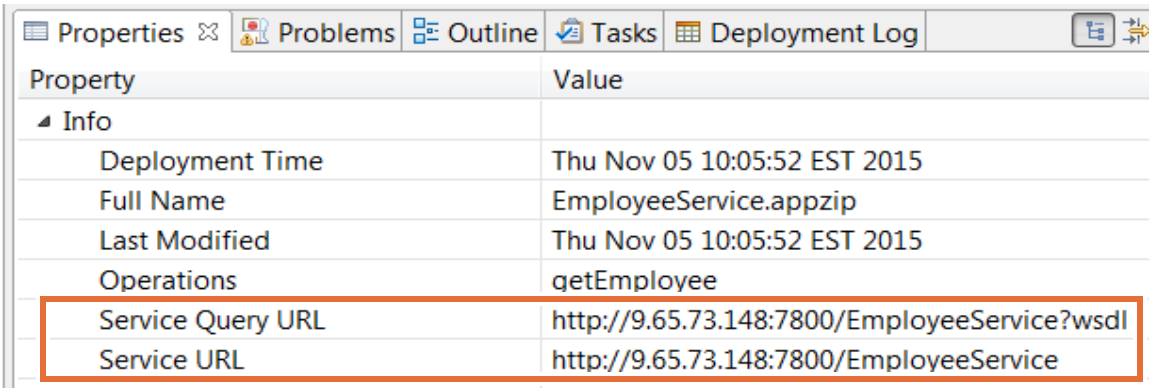
You enabled the WSDL document retrieval feature in the SOAP/HTTP Binding properties by selecting the **Enable support for ?wsdl** property.

__ 5. Examine the EmployeeService web service properties in the Integration Toolkit.

__ a. Select the **EmployeeService** integration service in the **Integration Nodes** view.



__ b. Review the web service endpoint address and the **Service Query URL** in the **Properties** view.



Note

The **Service URL** and **Service Query URL** display the web service endpoint that clients can access over the network. The web service endpoint with `localhost` is valid for applications that are running on the same system as the integration server.

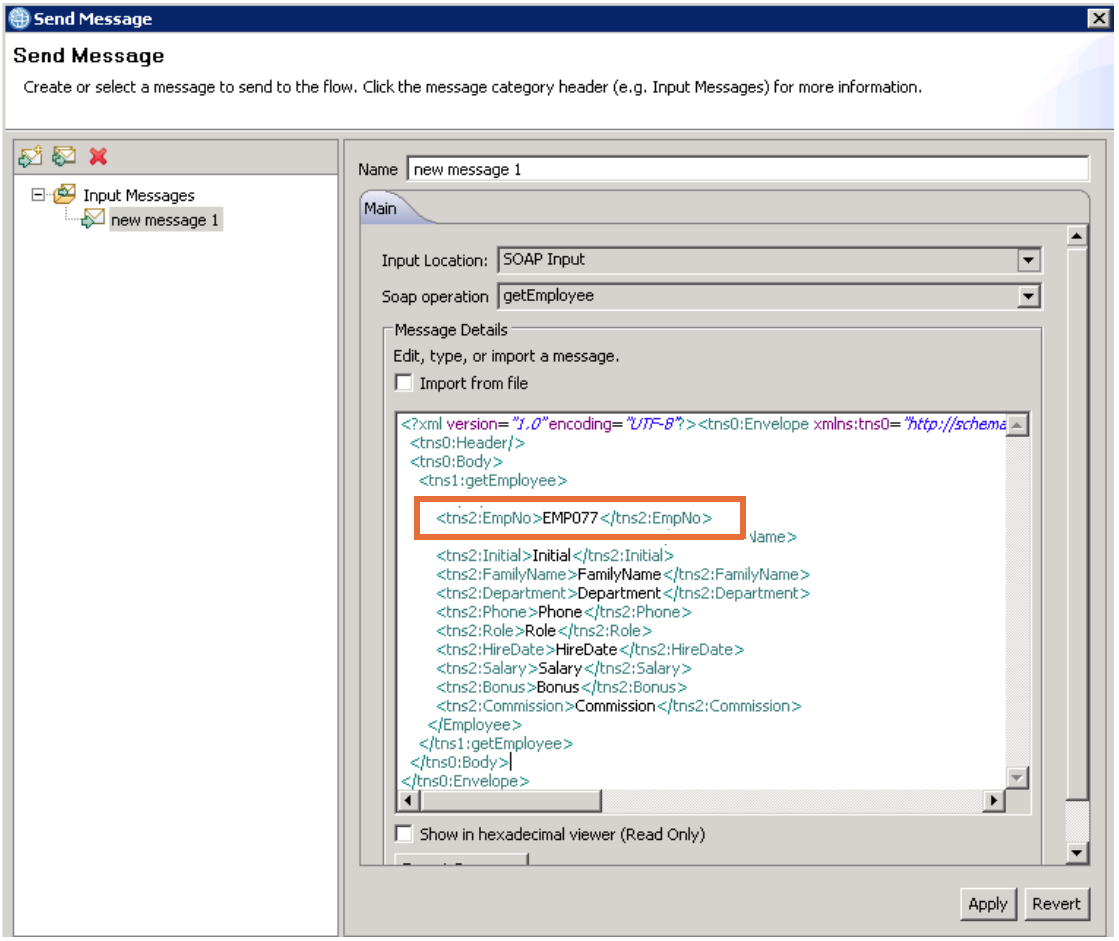
You can also view the web service endpoint address by using the IBM Integration web interface.

The IP address that is shown in your lab environment for the **Service URL** and **Service Query URL** does not exactly match the IP address shown in the example screen capture.

__ 6. In the Message Flow editor, click the **Send a message to the flow** icon.

__ 7. In the **Send Message** window, click the **New Message** icon.

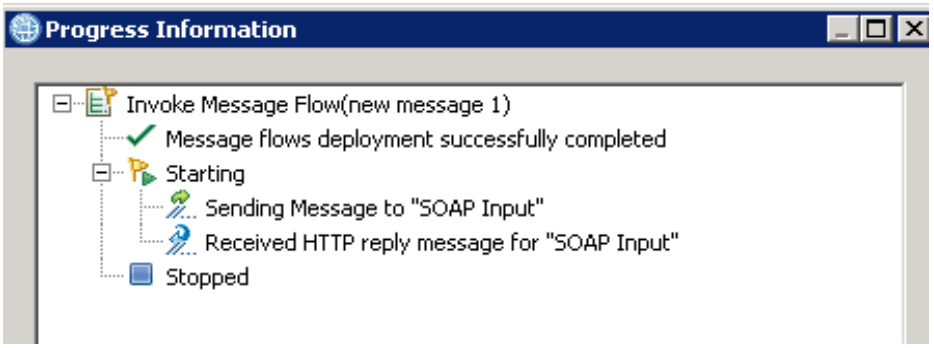
__ 8. Enter `EMP077` as the **EmpNo** value.



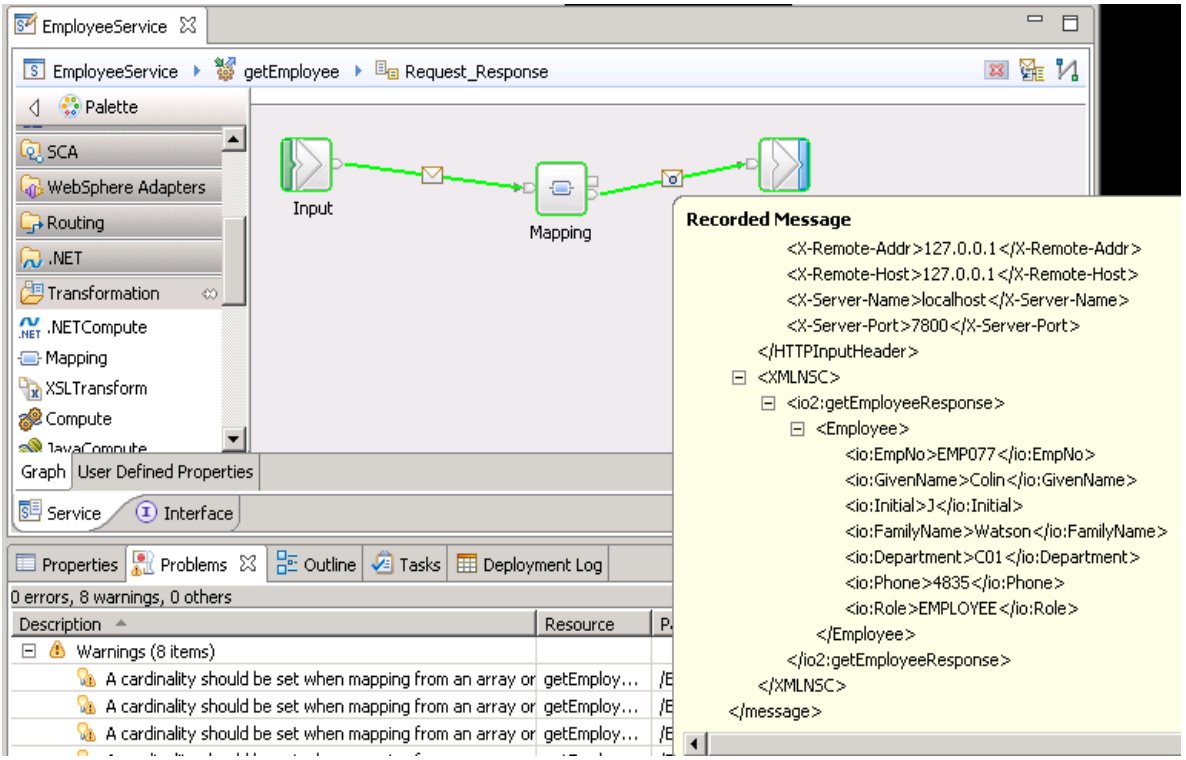
__ 9. Click **Send**.

__ 10. Verify the results of the web service operation.

__ a. In the **Progress Information** view, confirm that the test returns an HTTP reply message.



__ b. Examine the message after the Mapping node processes it to ensure that the database SELECT statement was successful and that it returned the employee details for EMP077.



Information

The **EmployeeService** integration service returns a response to the web service call in a SOAP message. The structure of the SOAP message body matches the WSDL output message, `getEmployeeResponse`. The mapping node in the message flow does an SQL select based on the **`EmpNo`** field, and completes the employee record details.

In the Flow exerciser, HTTPReply nodes can detect that the message that the flow is processing is a recorded message. If one of these nodes receives a recorded message, the node suppresses the reply. Attempting to send a reply would generate an error because no external system is available to receive the reply.

If you want to see the HTTP reply message, test the integration service by using the Integration Toolkit Unit Test Client.

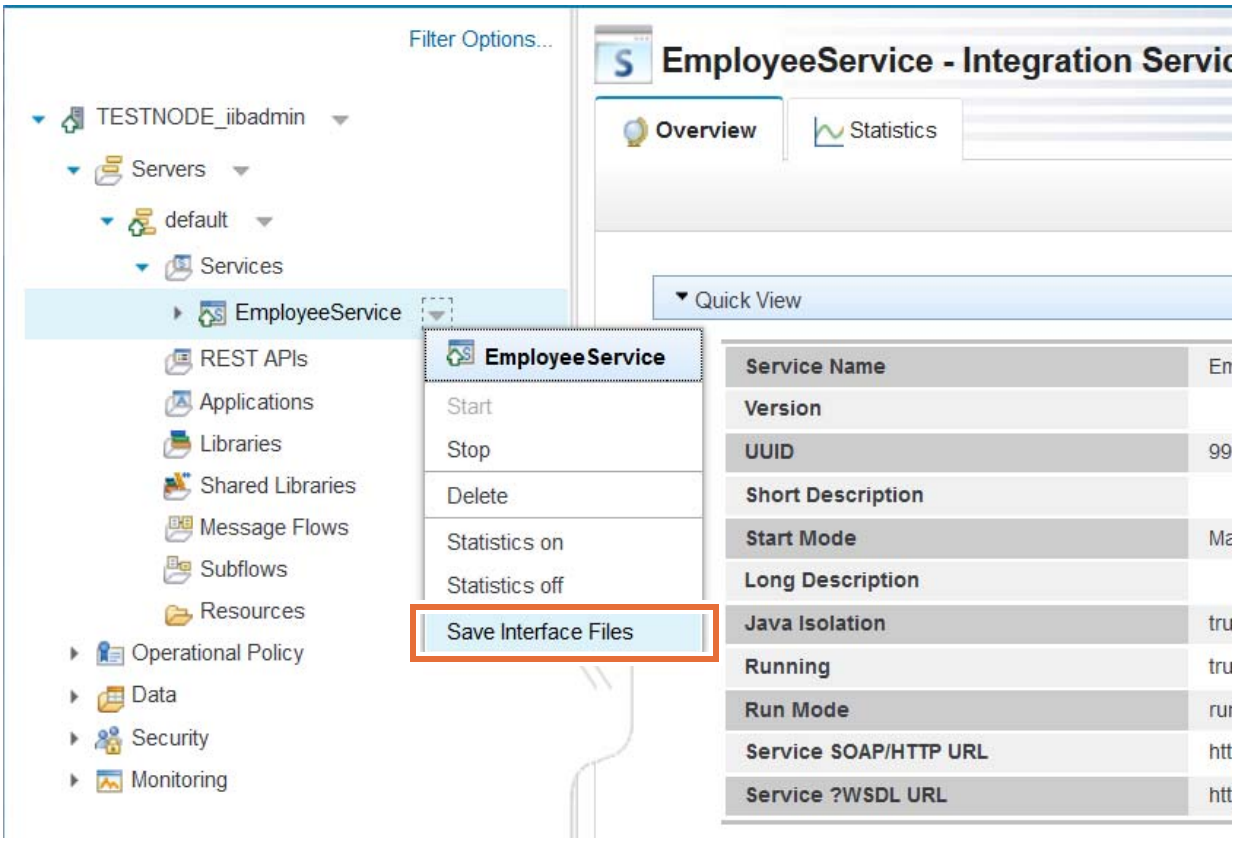
__ 11. Return the message flow to edit mode.

__ 12. Export the WSDL document that describes the EmployeeService integration service.

__ a. Start the web user interface for the integration node by right-clicking the TESTNODE_iibadmin integration node in the **Integration Nodes** view and then clicking **Start Web User Interface**.

__ b. Expand **Servers > default > Services**.

__ c. Click the down arrow to the right of **EmployeeService** and then click **Save Service Interface Files**.



__ d. Click **Save**. The `InterfaceFile.zip` file is downloaded to the `Download` directory.

__ 13. Confirm that the `InterfaceFile.zip` file contains the `EmployeeService.wsdl` file and the `EmployeeService-0.xsd` and `EmployeeService-1.xsd` XML schema definition files.

Exercise cleanup

__ 1. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, in case you are prompted.

__ 2. In the **Integration Nodes** view, right-click the TESTNODE_iibadmin integration node **default** integration server, and then click **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.

__ 3. In the **Integration Nodes** view, right-click the TESTNODE_iibadmin integration node **default** integration server, and then click **Stop Recording**.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you followed a top-down design and created a web service interface in IBM Integration Toolkit. You specified the operations, message parts, and web service bindings in a standard WSDL document.

In the second part of the exercise, you developed a message flow to implement the web service operation that is defined in the WSDL document. You retrieved information from a data source and constructed a web service response message with the mapping node.

In the last part of the exercise, you deployed and tested the service with the Integration Toolkit.

Having completed this exercise, you should be able to:

- Create a container for an integration service
- Design the operations, bindings, and messages for a service interface in a graphical editor
- Implement a service operation with a database SQL call
- Deploy and test an integration service by using the IBM Integration Toolkit flow exerciser
- Export the WSDL document that describes the service by using the IBM Integration web user interface

Exercise 5. Creating IBM MQ and database services

What this exercise is about

In this exercise, you create an IBM MQ request and response service and a database service.

What you should be able to do

After completing this exercise, you should be able to:

- Create an IBM MQ request and response service definition
- Publish the IBM MQ services in an Integration Registry
- Import previously stored IBM MQ services into a developers workspace
- Create a database service
- Extend an existing database service by adding more operations

Introduction

IBM Integration Bus provides a service discovery feature for IBM MQ and database resources. Discovered services are used in the Integration Toolkit to create IBM MQ and database artifacts. These artifacts are then used to create applications.

IBM MQ services can be optionally stored in an Integration Registry component that runs on an integration node. If IBM MQ services are in the Integration Registry, other Integration Bus developers can import them.

In this lab, you create a request and response IBM MQ service that is used to develop an application. You publish discovered IBM MQ services in the Integration Registry so that other developers might develop applications by using the discovered services.

You also create a database service to manage and retrieve data in a database.

Requirements

- IBM Integration Bus V10 and the development integration node TESTNODE_iibadmin
- IBM MQ V8 and a queue manager that is named IIBQM

- DB2 Express 10.1 and the SAMPLE database tables that are created by running the `DB2CREATE.cmd` in the `C:\Labs\Lab05-SvcDisc\install\DBSetup` directory
- JDBC configurable service for the SAMPLE database on TESTNODE_iibadmin
- Lab files in the `C:\Labs\Lab05-SvcDisc` directory.
- The following IBM MQ local queues (created in the exercise by running an MQSC script): SERVICE.DISCOVERY.DB.READ.IN, SERVICE.DISCOVERY.DB.READ.OUT, SERVICE.DISCOVERY.DB.ADD.IN, SERVICE.DISCOVERY.DB.ADD.OUT, SERVICE.TEST, SERVICE.TEST.RETRIEVE, SERVICE.DISCOVERY.DB.DELETE.IN, SERVICE.DISCOVERY.DB.DELETE.OUT

Exercise instructions

Exercise setup

Create the queues that you add to an existing IBM MQ service to the Integration Registry and import a service definition that is used later in the exercise.

__ 1. This exercise uses an IBM MQ queue manager that named **IIBQM**.

Verify that you have a queue manager that is named IIBQM and that it is running.

If you do not have a queue manager that is named **IIBQM**, create one now by using the IBM MQ Explorer.

__ 2. Define the application queues that are required for this exercise.

__ a. Start the IBM Integration Console.

__ b. Type the following command:

```
runmqsc IIBQM < C:\labfiles\Lab05-SvcDisc\install\MQSetup\Q_Defs.mqsc
```

__ c. In IBM MQ Explorer, verify that the following local queues were created successfully:

- SERVICE.DISCOVERY.DB.READ.IN
- SERVICE.DISCOVERY.DB.READ.OUT
- SERVICE.DISCOVERY.DB.ADD.IN
- SERVICE.DISCOVERY.DB.ADD.OUT
- SERVICE.TEST
- SERVICE.TEST.RETRIEVE
- SERVICE.DISCOVERY.DB.DELETE.IN
- SERVICE.DISCOVERY.DB.DELETE.OUT

__ 3. If it is not already open, start the IBM Integration Toolkit.

__ 4. Switch to a new workspace.

__ 5. Ensure that the integration node TESTNODE_iibadmin is running. Also, ensure that no flows or resources are deployed to the integration server that is named **default**.

This lab requires an existing IBM MQ service that is stored in the Integration Registry for the integration node that is named TESTNODE_iibadmin. The required IBM MQ service is available in a Project Interchange file in `C:\labfiles\Lab05-SvcDisc\install\PrimeServiceRegistry.zip`.

Import this Project Interchange file into the IBM Integration Toolkit. A library that is named `PrimeServiceRegistry` is created.

__ a. Click **File > Import**.

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

__ c. Browse to the `C:\labfiles\Lab05-SvcDisc\install` directory.

__ d. Click `PrimeServiceRegistry.zip`.

__ e. Click the **PrimeServiceRegistry** project and then click **Finish**.

__ 6. Publish the service definition to the Integration Registry for TESTNODE_iibadmin.

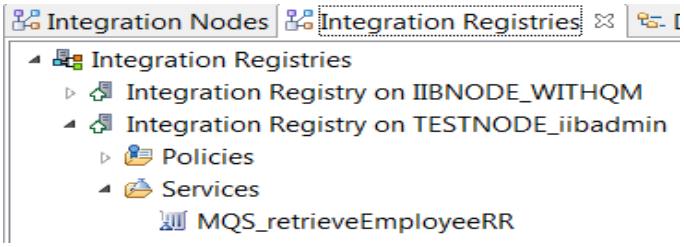
__ a. In the Integration Toolkit **Application Development** view, expand the **PrimeRegistryService** library until you see the IBM MQ Service definition `MQS_retrieveEmployeeRR.service` (under the **Services** folder).

__ b. Right-click the `Service MQS_retrieveEmployeeRR.service` service definition and then click **Publish to Registry**.

__ c. Select the Integration Registry for the TESTNODE_iibadmin integration node and then click **Finish**.

__ d. After a few moments, the Integration Registry view opens.

Verify that the **Integration Registries** view contains a service that is named `MQS_retrieveEmployeeRR` in the **Services** folder in the Integration Registry for TESTNODE_iibadmin.



Note

If you do not see the service definition in the Integration Registry, right-click **Integration Registry on TESTNODE_iibadmin** and then click **Refresh** to refresh the view.

__ e. After you confirm that the Integration Registry for TESTNODE_iibadmin contains a service, delete the **PrimeServiceRegistry** library from the **Application Development** view.

__ 7. Create a shared library that is named `ServiceDefinitions` to hold the IBM MQ and DB2 database service definitions that are created in this exercise.

__ a. From the **Application Development** view, click **New Library**.

__ b. For the library name, enter: `ServiceDefinitions`

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

__ d. Click **Finish**.

__ 8. An XML schema that is named `Employee.xsd` defines the message types that the IBM MQ service definitions use later in this exercise.

Import the `Employee.xsd` file into the **ServiceDefinitions** library.

__ a. Open Windows Explorer and go to the `C:\labfiles\Lab05-SvcDisc\resources` directory.

__ b. Drag the `Employee.xsd` file from Windows Explorer and drop it onto the **ServiceDefinitions** library in the Integration Toolkit **Application Development** view.

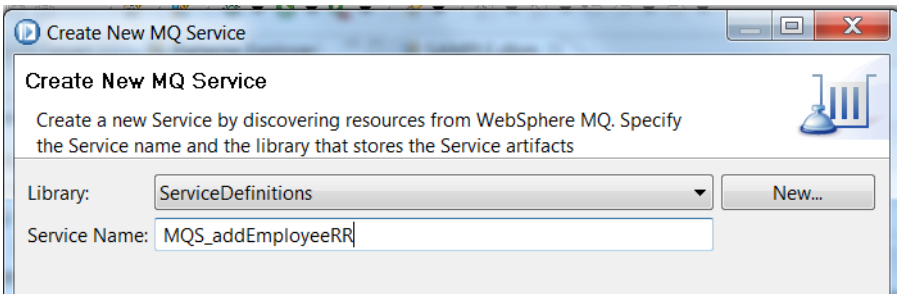
The `Employee.xsd` schema should now be listed under the **Service Definitions** folder of the **Service Definitions** library.

Part 1: Create an IBM MQ request/response service definition

In this part of the lab, you create IBM MQ Service definitions for queues that are used in a message flow later in this exercise. You also create and IBM MQ connection policy and store the IBM MQ service in the Integration Registry.

__ 1. In the **Application Development** view, click **New > MQ Service**.

__ 2. In the **Create New MQ Service** window, type `MQS_addEmployeeRR` for the **Service Name** and then click **Finish**.



The IBM MQ Service editor opens on the **Description** page.

__ 3. Click **Connect** in the **Checklist** pane to go to the **Connect** page.

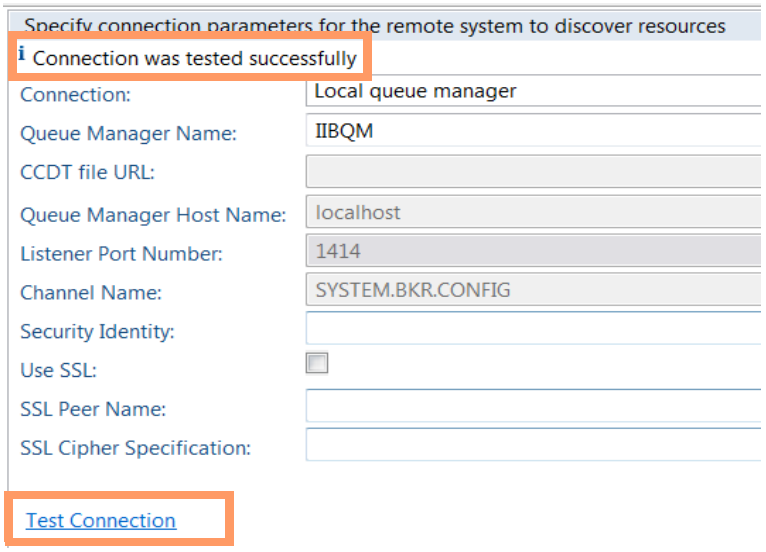
Note

You can expand the IBM MQ Service editor window by double-clicking the **MQS_addEmployeeRR.service** tab.

__ 4. On the **Connect** page, test the connection to the queue manager from the Integration Toolkit.

__ a. For the **Queue Manager Name,** type: `IIBQM`

__ b. Click **Test Connection**. The message `Connection was tested successfully` should appear on the **Connect** page.



__ 5. The **Select Resources** page lists the queues on queue manager IIBQM. On this page, define a request-response operation.

__ a. Click **Select Resources** in the **Checklist** pane.

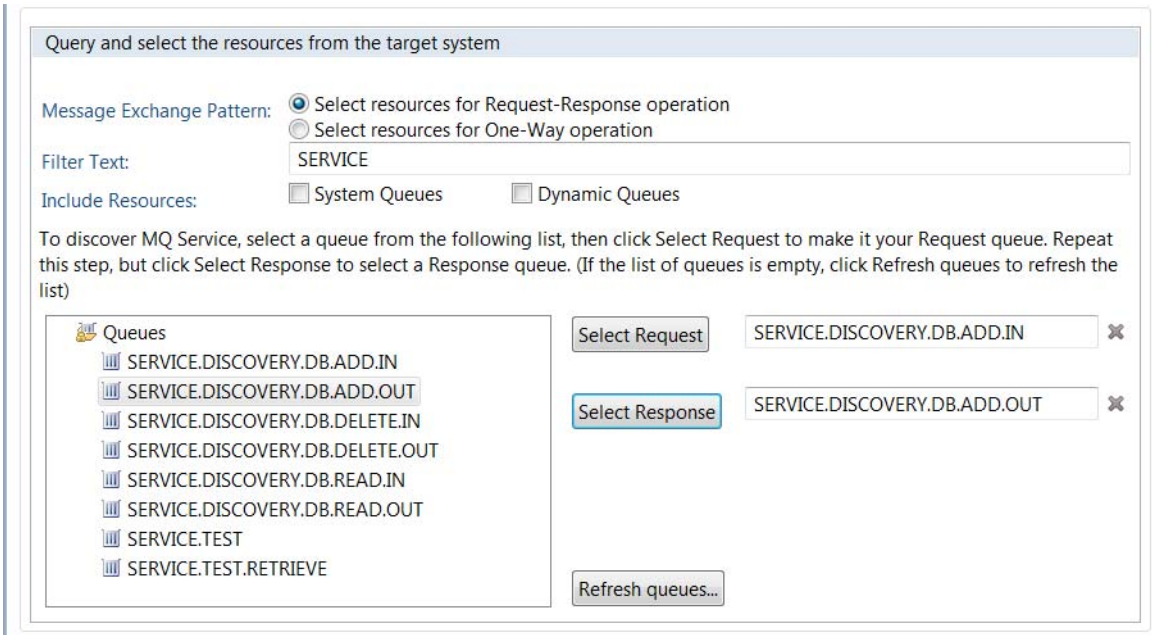
__ b. Click the **Select resources for Request-Response operation** option.

__ c. To restrict the number of queues in the view to only those queues that are used for this exercise, type `SERVICE` in the **Filter Text** field.

The queues with `SERVICE` in the first part of the queue name are listed under the **Queues**.

__ d. Click the queue that is named `SERVICE.DISCOVERY.DB.ADD.IN` in the list of queues and then click **Select Request**. The queue name is copied to the **Select Request** field.

__ e. Click the queue that is named `SERVICE.DISCOVERY.DB.ADD.OUT` in the list of queues and then click **Select Response**. The queue name is copied to the **Select Response** field.



__ 6. The **MQ Configuration** page defines the message headers.

__ a. In the **Checklist** pane, click **MQ Configuration > Request Message Headers**.

__ b. Verify that the **Message type** is set to `Request`.

__ c. In the **Checklist** pane, click **MQ Configuration > Response Message Headers**.

__ d. Verify that the **Message type** is set to `Reply` for the **Response Message Headers**.

__ 7. On the **Service Interface** page, update the input and output operations to reference the **EmployeeType** from the `Employee.xsd` file that you imported in **Exercise setup**.

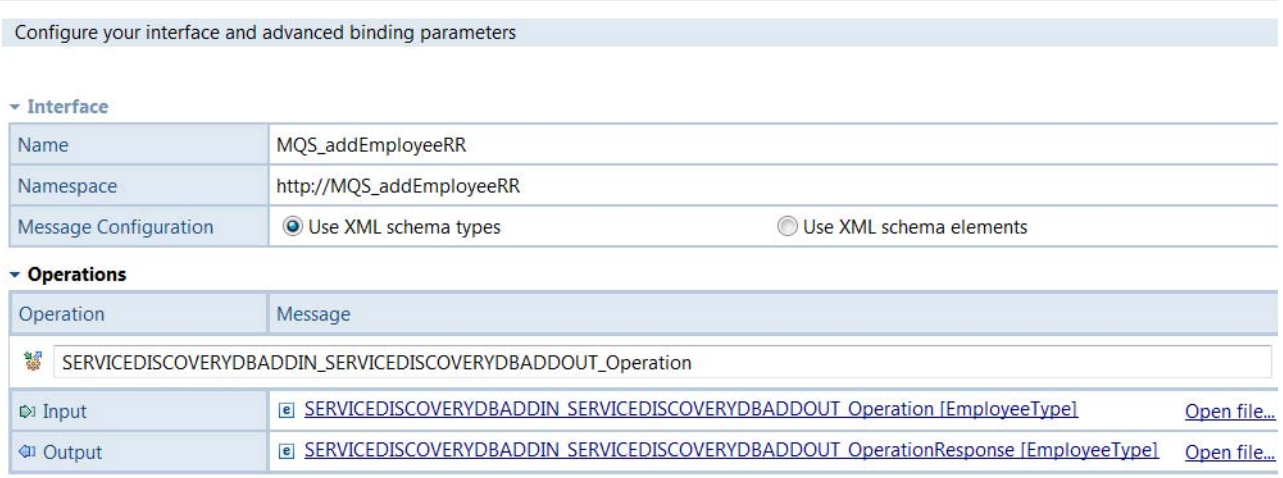
__ a. In the **Checklist**, click **Service Interface**.

__ b. Under the **Operations** section, click the **Click to select message** link for the **Input**.

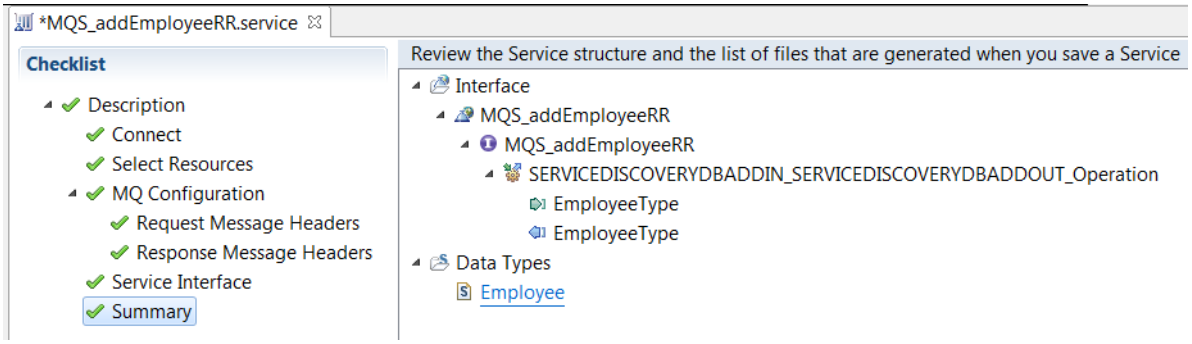
__ c. Click **EmployeeType** from the list of **Filtered data types** and then click **OK**.

__ d. Click the **Click to select message** link for the **Output**.

__ e. Select **EmployeeType** from the list of **Filtered data types** and then click **OK**.



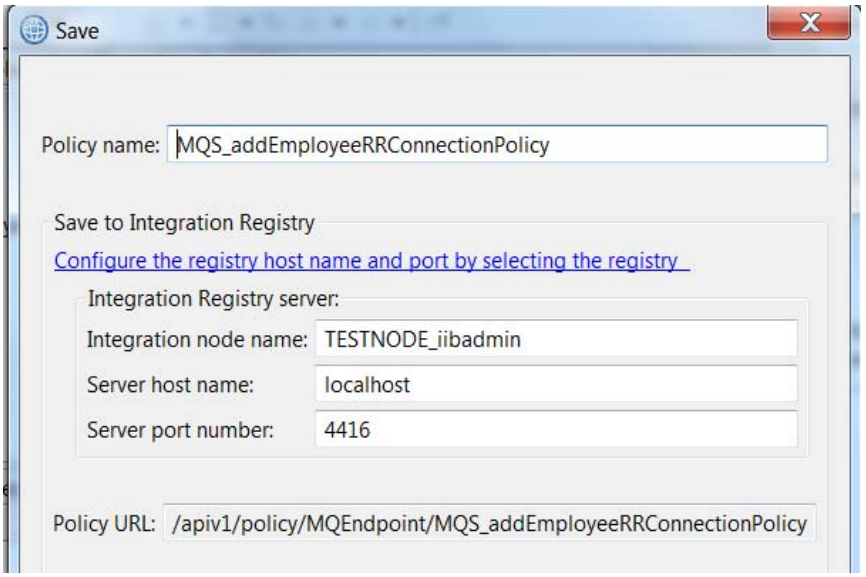
__ 8. On the **Summary** page, review the service structure. In the Checklist pane, click **Summary**.



__ 9. Click **File > Save** to save the service definition.

__ 10. Click **Yes** to the prompt to create an IBM MQ connection policy.

__ 11. Verify that the target integration node for the connection policy is TESTNODE_iibadmin and then click **OK** on the **Save** window.

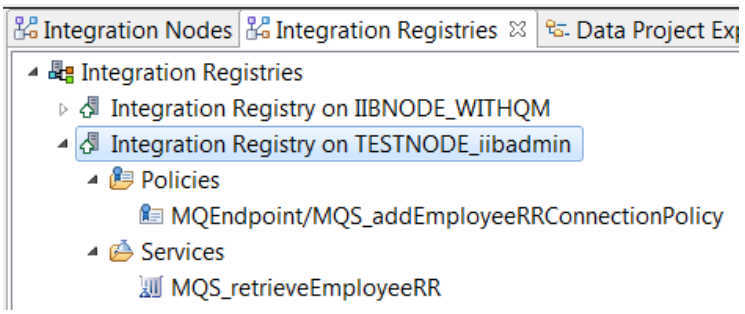


__ 12. Verify that the new IBM MQ Connection policy is stored in the TESTNODE_iibadmin Integration Registry.

__ a. In the **Integration Registries** view, right-click **Integration Registry on TESTNODE_iibadmin** and then click **Refresh** to refresh the view.

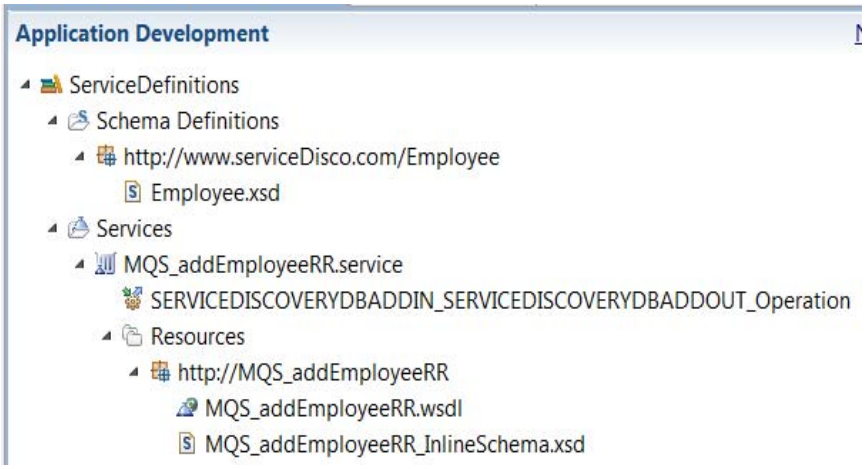
__ b. Verify that the **Policies** folder contains an MQEndpoint policy that is named: `MQS_addEmployeeRRConnectionPolicy`

It might be necessary to refresh the view by right-clicking **Integration Registry on TESTNODE_iibadmin** and then clicking **Refresh**.



__ 13. The **Application Development** view should now contain the `MQS_addEmployeeRR.service` under the **Services** folder in the **ServiceDefinitions** library.

Close the **MQS_addEmployeeRR.service** tab.

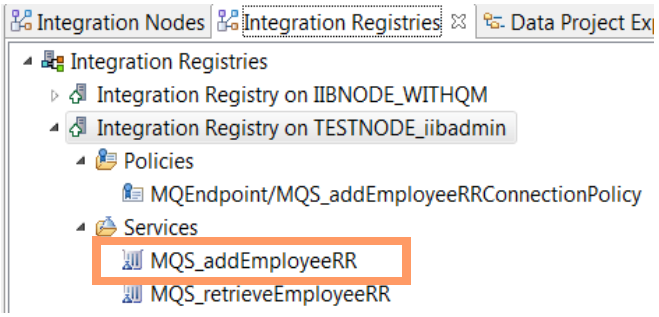


__ 14. Publish the IBM MQ service in the Integration Registry for TESTNODE_iibadmin.

__ a. In the **Application Development** view, right-click the **MQS_addEmployeeRR.service Services** entry and then click **Publish to Registry**.

__ b. Click **TESTNODE_iibadmin** and then click **Finish**.

After a few moments, the **MQS_addEmployeeRR** service should be listed under the **Services** folder in the Integration Registry on TESTNODE_iibadmin.



Part 2: Discover the database service

In this part of the exercise, you create a database service that is based on the DB2 SAMPLE database. If you are using the lab image, the database is created for you.

First, you ensure that a JDBC Provider is configured for the SAMPLE database and then you create the database definition file that defines the database connection. Then, you create a database service that contains the add (INSERT) and retrieve (SELECT) operations on an EMPLOYEE table in the SAMPLE database.

__ 1. Ensure that the integration node TESTNODE_iibadmin contains a JDBC configurable service for the DB2 SAMPLE database.

__ a. Start the IBM Integration web user interface for TESTNODE_iibadmin.

__ b. Expand **Operational Policy > Configurable Services > JDBC Providers**.

__ c. Verify that JDBC Providers contains a provider that is named **SAMPLE**.

If the SAMPLE JDBC Provider does not appear in the list, open an IBM Integration console and run the **`ConfigureJDBC`** command to create the JDBC configurable service for the SAMPLE database.

In an IBM Integration Console, change directories to the `C:\labfiles\Lab05-DiscSvcs\install\DBSetup` directory.

To create the JDBC Provider for the SAMPLE database, type:

```
ConfigureJDBC TESTNODE_iibadmin SAMPLE
```

__ 2. In the Integration Toolkit, create the database definition file for the DB2 database SAMPLE, in the ADMINISTRATOR schema, and store it in the DB_DATA_PROJECT design 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` and then click **Finish**.

__ d. On the **Create a data design project** page, click **Next**.

__ e. In the **Select Connection** page, click **New**.

__ f. In the **Properties** pane, for **Database**, type: `SAMPLE` (if it is not already entered)

__ g. For **User name**, type: `Administrator`

__ h. For **Password**, type: `web1sphere`

__ i. Click **Save password**.

__ j. Click **Test Connection** to ensure that the JDBC connection is tested.

__ k. A “Connection succeeded” message should be displayed. Click **OK**.

__ l. Click **Finish**. The **New Database Definition File** window displays the connection information for the SAMPLE database.

__ m. Click **SAMPLE1**, and then click **Next**.

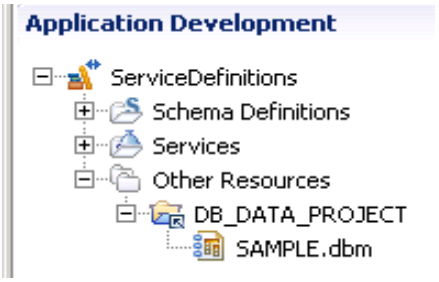
__ n. Select **ADMINISTRATOR** and then click **Finish**.

__ 3. Add the database definition to the **ServiceDefinitions** library.

__ a. Right-click the **ServiceDefinitions** library in the **Application Development** view and then click **Manage included projects**.

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

The **SAMPLE** project should now be listed within the **Other Resources** folder under the **ServiceDefinitions** library.



__ 4. Create a database service that is named `DBS_employee`.

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

__ b. Name the new service `DBS_employee` and then click **Finish**.

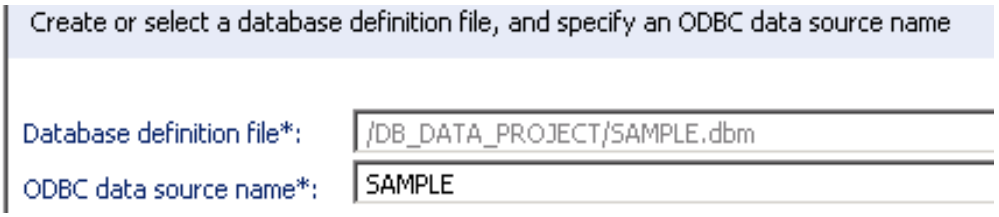
The Database Service editor opens.

__ c. In the **Checklist** pane, click **Database Definition**.

__ d. On the **Database Definition** page, click **Select** to select a database definition file.

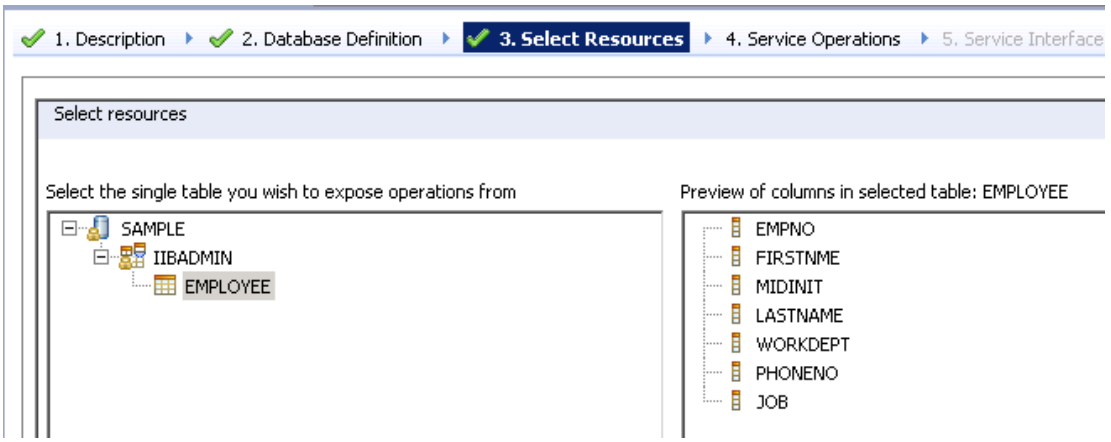
__ e. Click `SAMPLE.dbm` and then click **OK**.

The details of the SAMPLE database are copied into the **Database definition file** and **ODBC data source name** fields.



__ f. In the **Checklist**, click **Select Resources**.

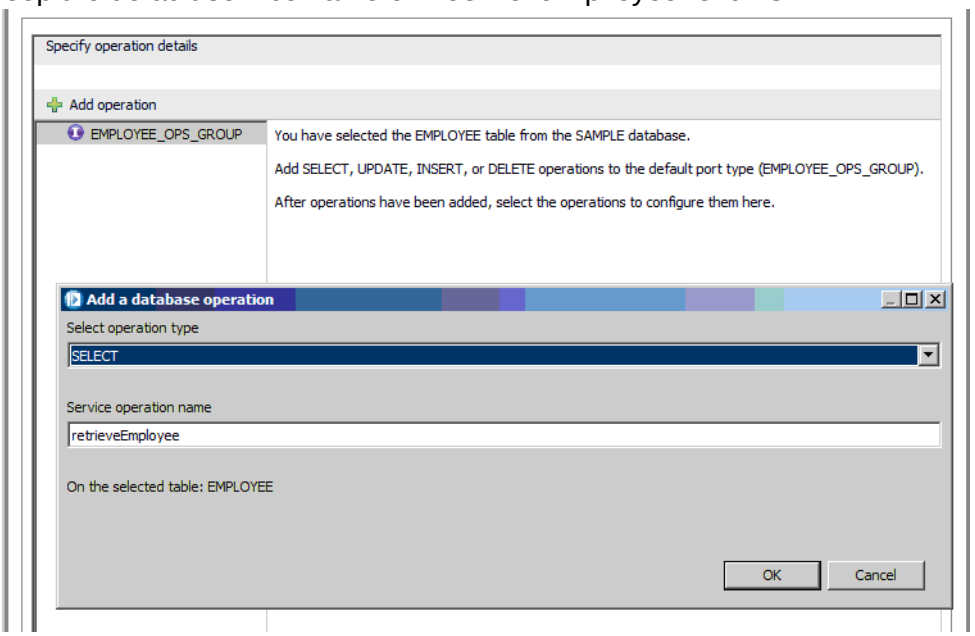
__ g. On the **Select Resources** page, click EMPLOYEE to select the EMPLOYEE table. The **Preview of columns** pane is updated with columns of the selected table.



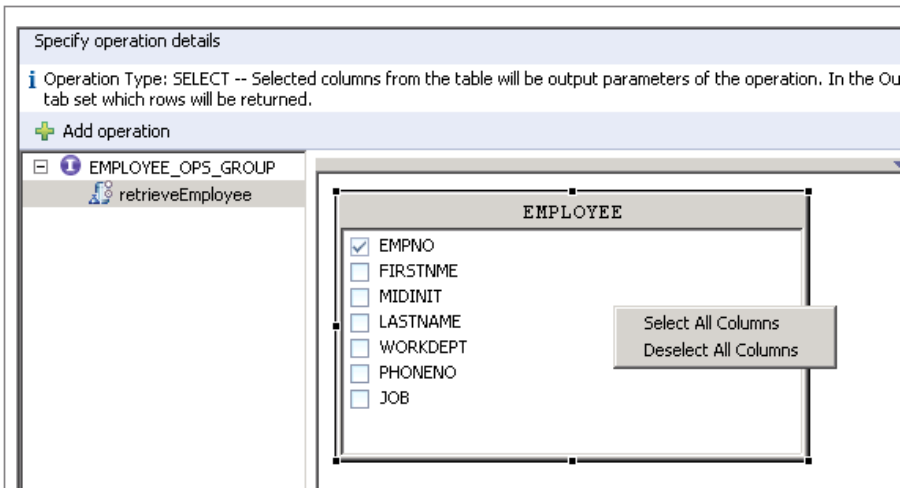
__ h. In the **Checklist**, click **Service Operation.**

__ i. On **Service Operations** page, click **Add operation**.

__ j. On the **Add a database operation** window, choose **SELECT** for the operation type and keep the default service name of `retrieveEmployee`. Click **OK**.



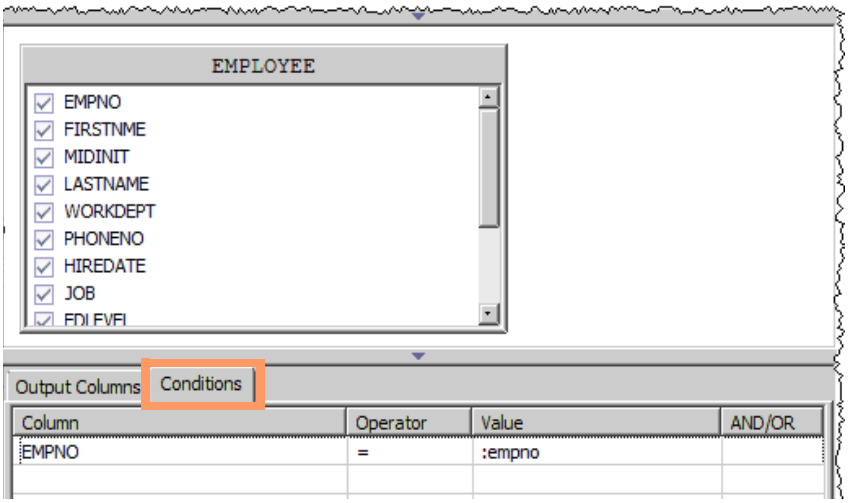
__ k. A list of column names that can be selected as output parameters of the **retrieveEmployee** operation are displayed. EMPNO is selected by default. Right-click the EMPLOYEE table and click **Select All Columns**.



__ l. Click the **Conditions** tab.

__ m. Click an empty cell under **Column** and then select EMPLOYEE.EMPNO from the EMPLOYEE table.

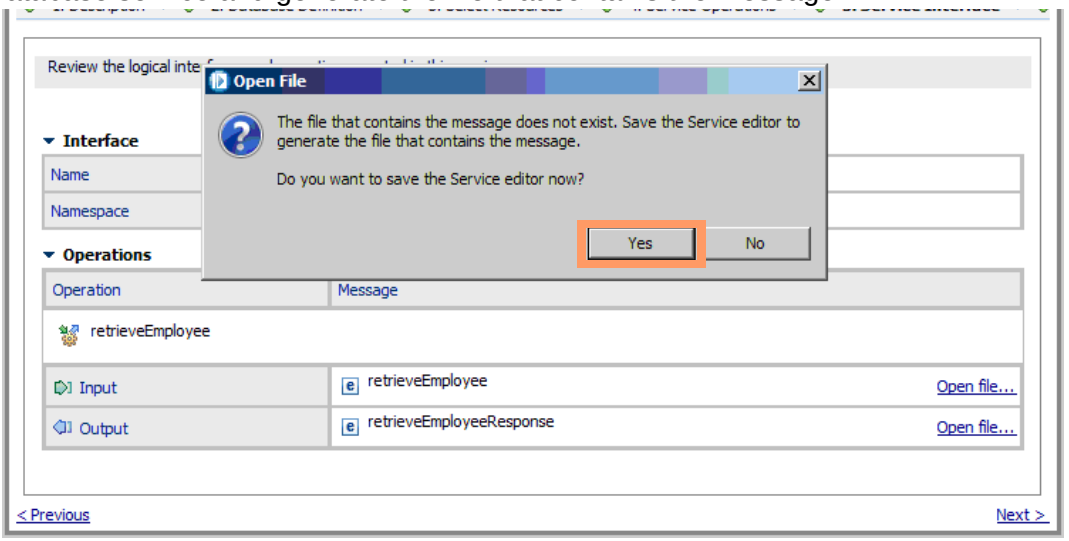
The **Operator** column is set to = and the **Value** column is set to `empno` (the input parameter) automatically.



__ n. In the **Checklist**, click **Service Interface**.

__ o. On the **Service Interface** page, click the **Open file** link on the **retrieveEmployee** Input row.

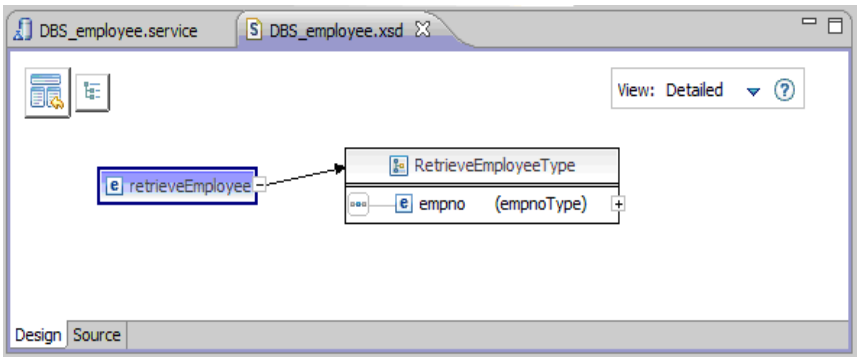
__ 5. A message is displayed indicating that the file does not exist. Click **Yes** to save the database service and generate the file that contains the message.



__ a. Click **File > Save** to save the service.

__ b. On the **Service Interface** page, click the **Open file** link on the **retrieveEmployee** Input row.

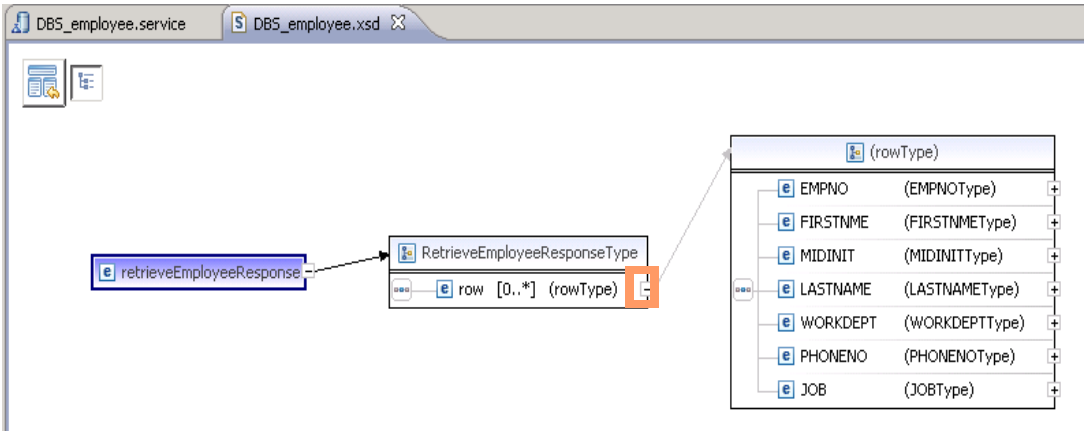
__ c. After it is generated, the **retrieveEmployee** input XML schema file (`DBS_employee.xsd`) is opened in the XML Schema editor.



__ d. Close the `DBS_employee.xsd` file.

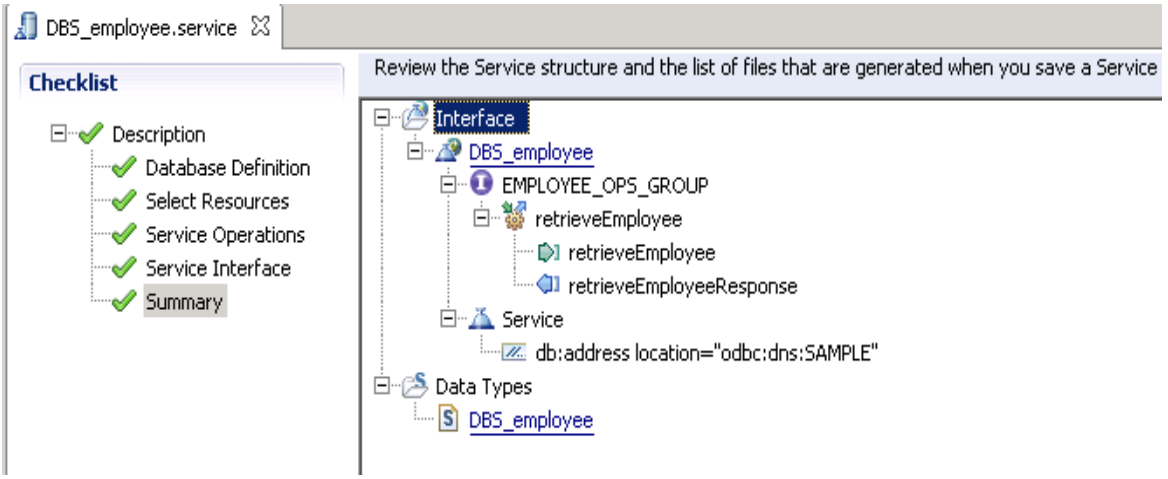
__ e. On the **Service Interface** page of the Database Service editor, click the **Open file** link for the **retrieveEmployeeResponse** output.

The generated XML schema file for the output is opened on the **Design** tab. The columns that you selected for output are included; click the **+** sign to display the list.



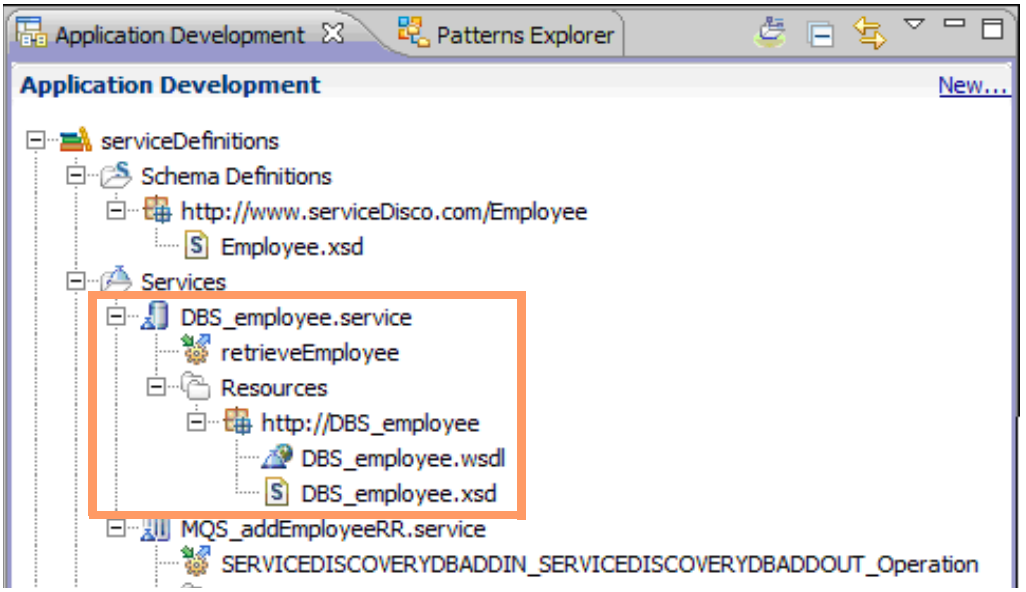
__ f. Close the `DBS_employee.xsd` file.

__ g. In the **Checklist**, click **Summary** to go to the **Summary** page.



__ h. You saved the Database Service file to generate the `DBS_employee.xsd` file, so the **Save** option is not available. Close the Database Service editor for `DBS_employee.service`.

__ 6. Now that you created the database service, the **Application Development** view is updated to include the database service under the **Services** folder in the **ServiceDefinitions** library:



Part 3: Add an operation to the database service

When you discovered the database service, you added the SELECT operation. This operation is used to retrieve data from the database by the application.

In this part of the exercise, you edit the existing discovered database service and add the INSERT operation so that you can add data to the database by using the database service.

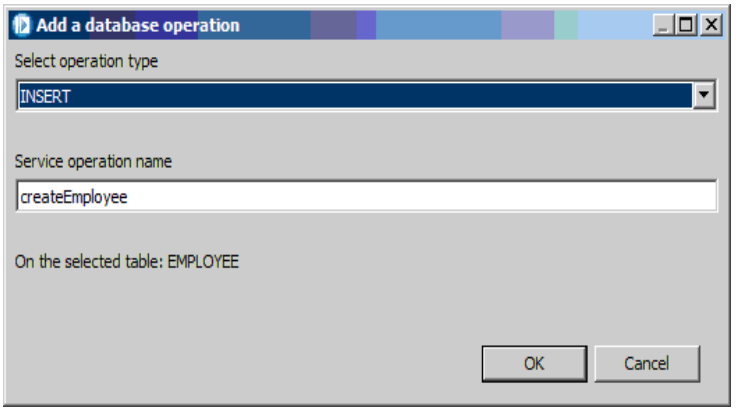
__ 1. Open the Database Service editor by double-clicking **DBS_employee.service** in the **ServiceDefinitions** library.

__ 2. In the Database Service editor, click **Service Operations** in the **Checklist** pane.

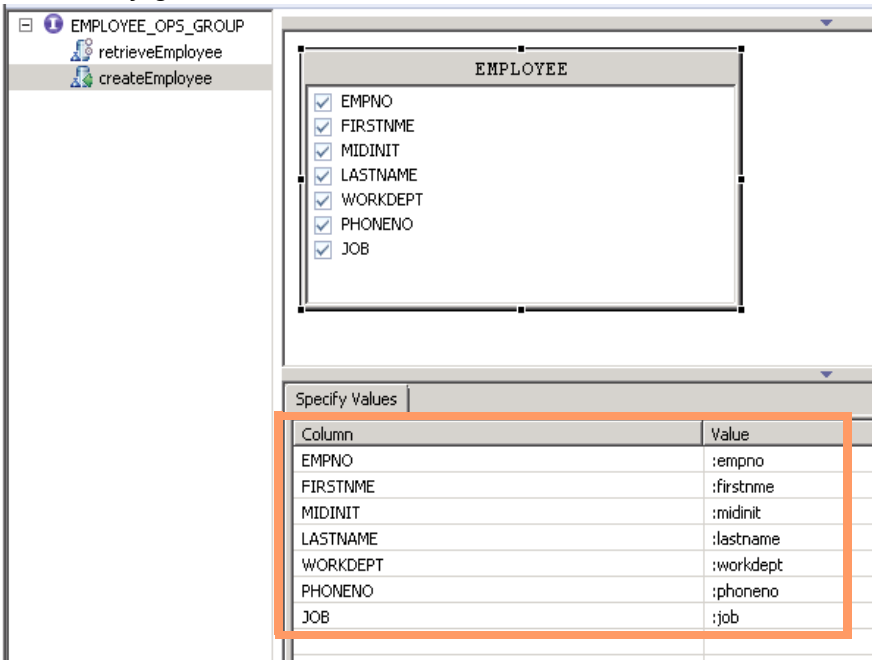
The operation list contains the **retrieveEmployee** operation that you created in Part 2 of this exercise.

__ 3. Click **Add operation**.

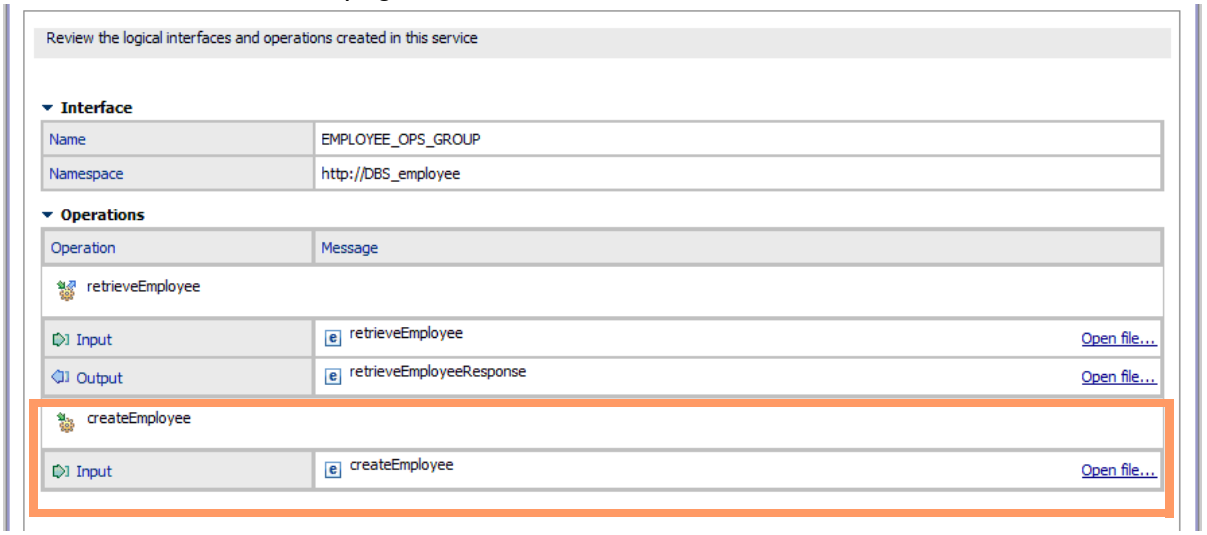
__ 4. Select INSERT from the **Select operation type** list. Accept the **Service operation name** of `createEmployee` and click **OK**.



__ 5. Right-click the EMPLOYEE table and click **Select All Columns**. Values for all the columns are automatically generated.

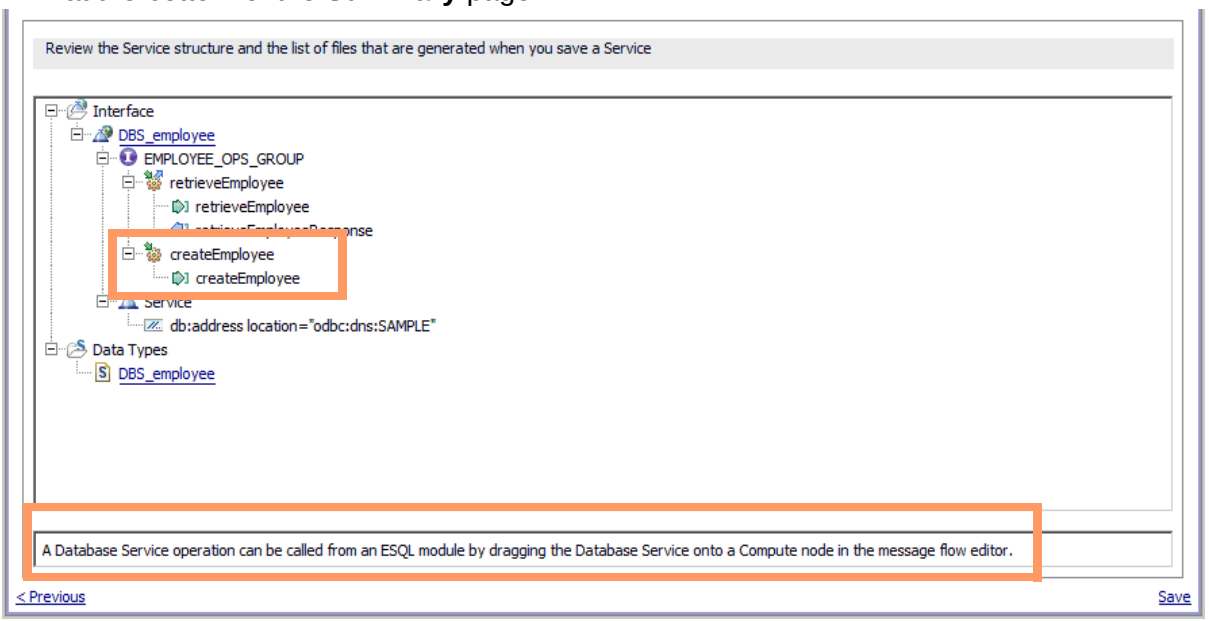


__ 6. In the **Checklist** pane, click **Service Interface**. The **createEmployee** operation is added to the **Service Interface** page.



__ 7. In the **Checklist** pane, click **Summary**.

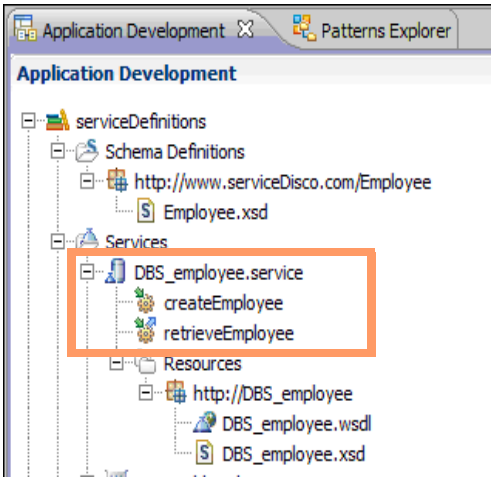
On the **Summary** page, you should see that the **createEmployee** operation is added to the EMPLOYEE_OPS_GROUP. An explanation on how to use the database service is provided at the bottom of the **Summary** page.



__ 8. Click **File > Save** to save the **createEmployee** operation in the database service.

__ 9. Close the Database Service editor.

__ 10. Verify that the **createEmployee** operation is listed under the **DBS_employee.service** in the **Application Development** view.



__ 11. Close all open editors.

End of exercise

Exercise review and wrap-up

In Part 1 of the exercise, you created an IBM MQ request and response service definition. You published the IBM MQ request and response service definition to the Integration Registry.

In Part 2 of the exercise, you created a database service definition that contained add (INSERT) and retrieve (SELECT) operations on an EMPLOYEE table in the SAMPLE database.

In Part 3 of the exercise, you added an operation to a database service.

Exercise 6. Creating a decision service

What this exercise is about

In this exercise, you use the IBM Integration Toolkit to create a decision service, which is a collection of rules that are used to process a message. You then test the decision service by using it in a message flow.

What you should be able to do

After completing this exercise, you should be able to:

- Create a decision service
- Write business rules in the IBM Integration Toolkit Rule Designer
- Test the decision service

Introduction

You can use the IBM Integration Bus Toolkit to write business rules by using natural language, so that business users, such as a business analyst, can read them. In the Integration Toolkit, you create a decision service, which is a collection of rules that are used to process a message.

A Decision Service node runs those business rules to provide operations like routing, validation, and transformation.

The decision service that you create in this exercise references the `BookOrder.xml` file.

The structure of the XML file is shown here.

```
<bookOrder xsi:noNamespaceSchemaLocation="book_order.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<orderDate>2015-12-31T12:00:00</orderDate>
<orderList>
   <item>
      <author>author</author>
      <title>title</title>
      <price>0.0</price>
      <quantity>0</quantity>
   </item>
</orderList>
<orderSummary orderNumber="">
   <orderTotal>0.0</orderTotal>
   <promotionCode>promotionCode</promotionCode>
   <loyaltyCard>loyaltyCard</loyaltyCard>
   <postAndPackaging>0.0</postAndPackaging>
   <discounts>discounts</discounts>
</orderSummary>
</bookOrder>
```

The `BookOrder.xsd` XML schema file describes the `BookOrder.xml` file. In the first part of this exercise, you create a library and then import the `BookOrder.xsd` file into the library. Next, you create a decision service that references the `BookOrder.xsd` schema.

The business rule that you create in this exercise sets the <postAndPackaging> value to zero if the <orderTotal> value is greater than or equal to 15. If the <orderTotal> value is less than 15, then the <postAndPackaging> value is set to 2.50.

```
if the order total of the order summary of 'the book order' is at least 15
then
set the post and packaging of the order summary of 'the book order' to 0;
else
set the post and packaging or the order summary of 'the book order' to 2.50;
```

In the second part of this exercise, you create a message flow application that implements the decision service.

In the third part of this exercise, you use the Integration Toolkit Flow exerciser to test the decision service.

Requirements

- IBM Integration Toolkit V10
- IBM MQ V8 and basic experience with IBM MQ Explorer
- Lab files in the `C:\labfiles\Lab06-DecSvc` directory

Exercise instructions

Exercise setup

__ 1. The message flow that you create in this exercise contains an MQ Input and MQ Output node.

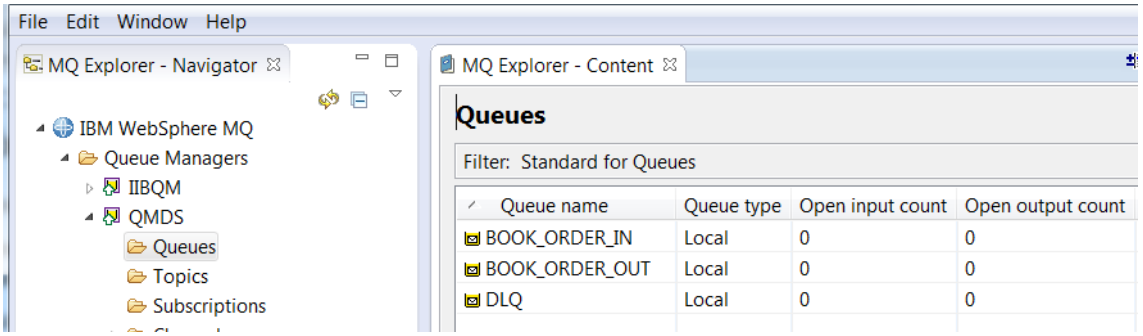
Using IBM MQ Explorer, create a queue manager that is named `QMDS` with a dead-letter queue that is named `DLQ`. This queue manager hosts the application queues for this exercise.

__ 2. Run a script in the IBM Integration Console to create the dead-letter queue (DLQ) and application queues for this exercise (BOOK_ORDER_IN, BOOK_ORDER_OUT).

In the IBM Integration Console, type:

```
runmqsc QMDS < C:\labfiles\Lab06-DecSvc\resources\Q_defs.mqsc
```

__ 3. Using IBM MQ Explorer, verify that the dead-letter queue and the application queues were created successfully.



Part 1: Create the decision service

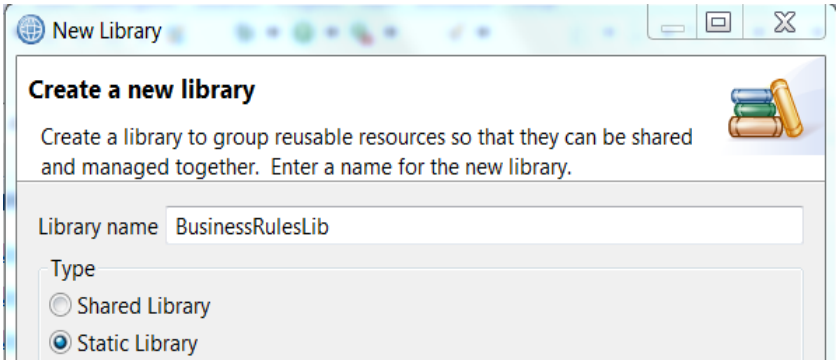
In this part of the exercise, you import the `BookOrder.xsd` XML schema. You then create a decision that references the schema and define the business rule.

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

__ 2. Switch to a new workspace.

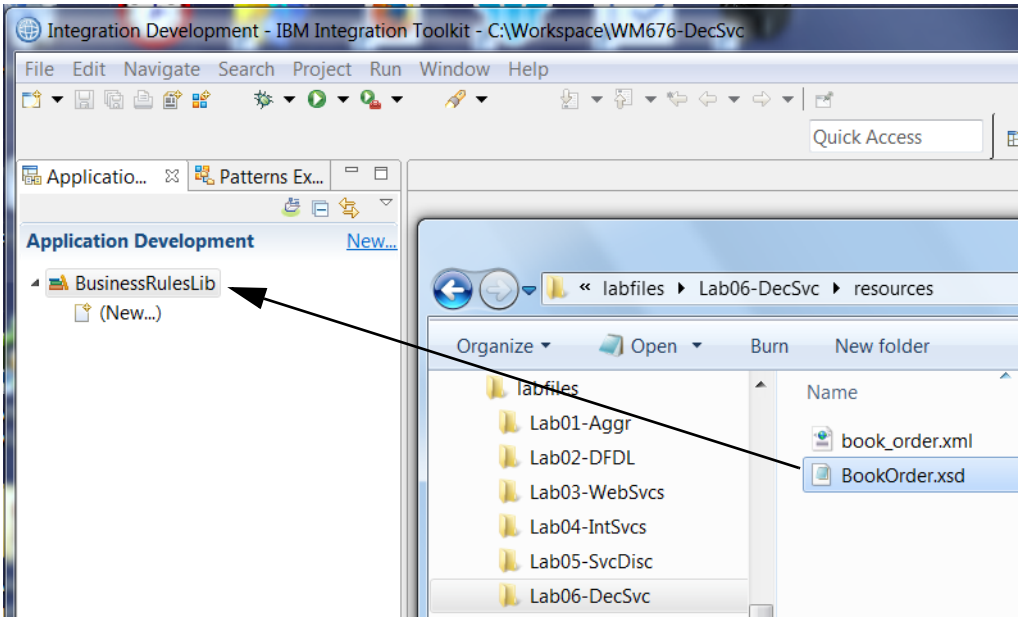
__ 3. You can create a decision service in a static or shared library. For this exercise, you create the decision service in a static library.

Create a static library that is named BusinessRulesLib.

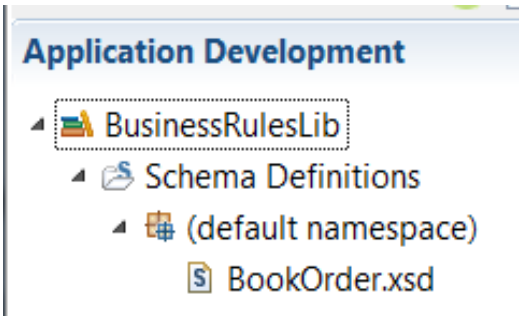


__ 4. Import the BookOrder.xsd XML schema into the **BusinessRulesLib** library.

__ a. From Windows Explorer, drag the BookOrder.xsd schema from the C:\labfiles\Lab06-DecSvc\resources directory onto the **BusinessRulesLib** library in the **Application Development** view in the Integration Toolkit.

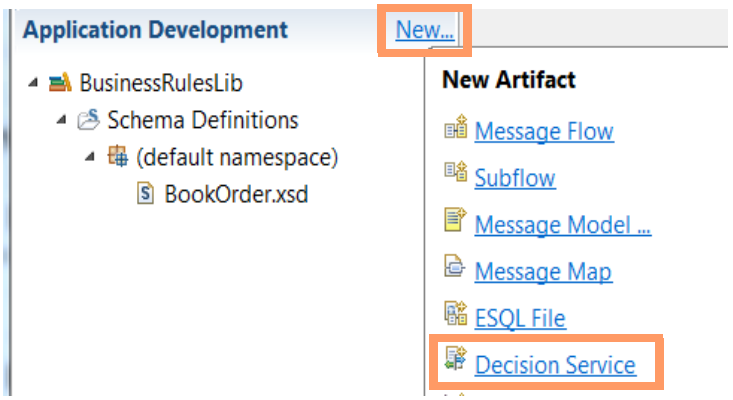


__ b. Verify that the schema was imported successfully. The schema definition should appear in the **Application Development** view.



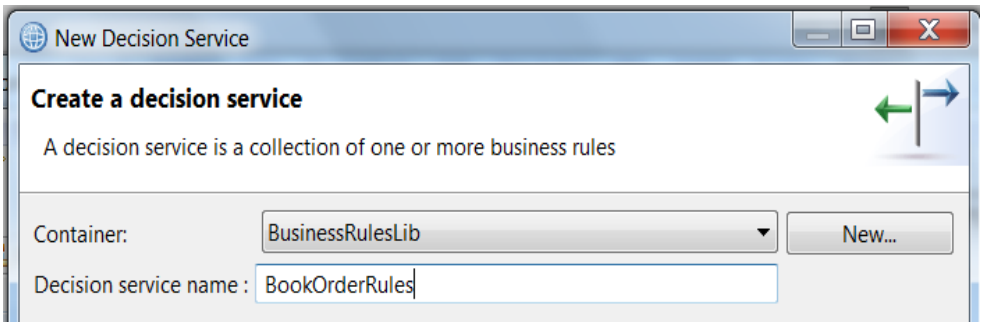
__ 5. Create a decision service in the **BusinessRulesLib** library that is named: `BookOrderRules`

__ a. Click **New > Decision Service**.



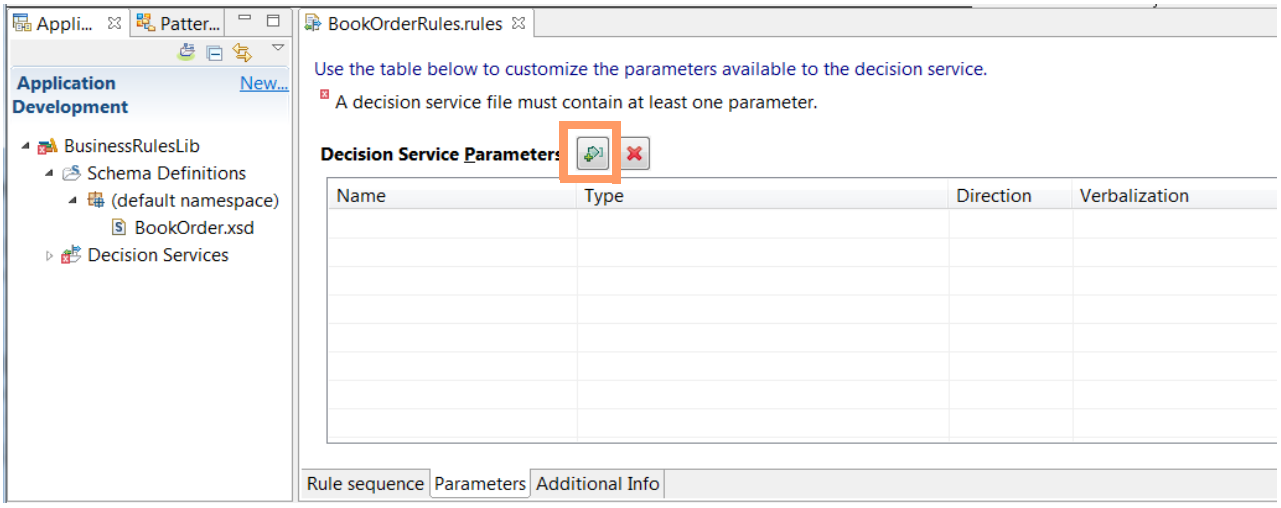
__ b. For the **Container**, select the **BusinessRulesLib** library.

__ c. For the **Decision service name**, type: `BookOrderRules`



__ d. Click **Finish**.

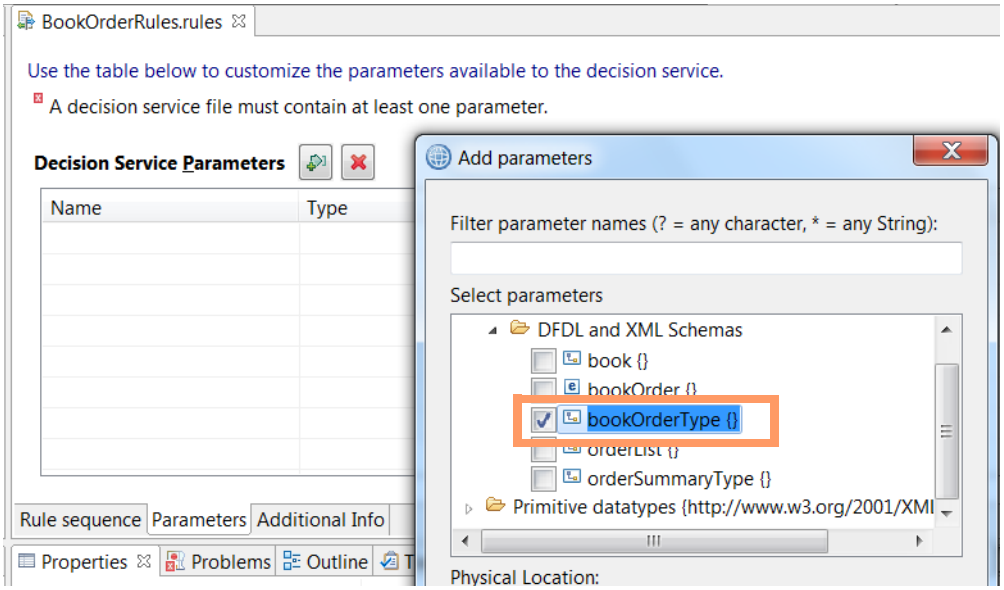
__ e. The Rules editor opens on the **Parameters** tab.



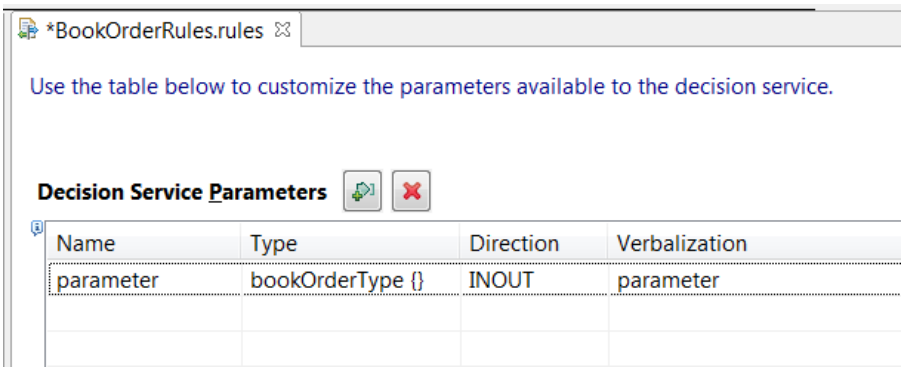
__ 6. Add the business rule parameters in the Rules editor **Parameters** tab.

__ a. Click the **Add parameters** icon.

__ b. On the **Add parameters** window, expand **BusinessRulesLib > DFDL and XML schemas**, and then click **bookOrderType**.



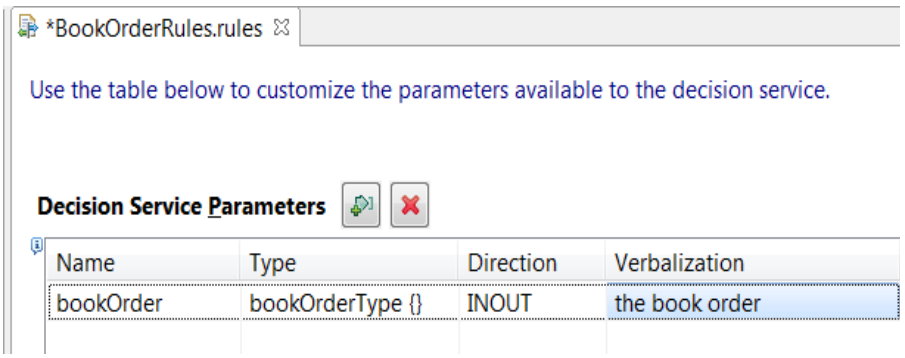
__ c. Click **OK**. The parameter is added to the **Decision Service Parameters** table.



__ 7. The **bookOrderType** parameter is added to the table but it does not have meaningful or descriptive names.

__ a. Change the **Name** value from `parameter` to `bookOrder` by clicking the field and then typing the new value.

__ b. Change the **Verbalization** value from `parameter` to: `the book order`

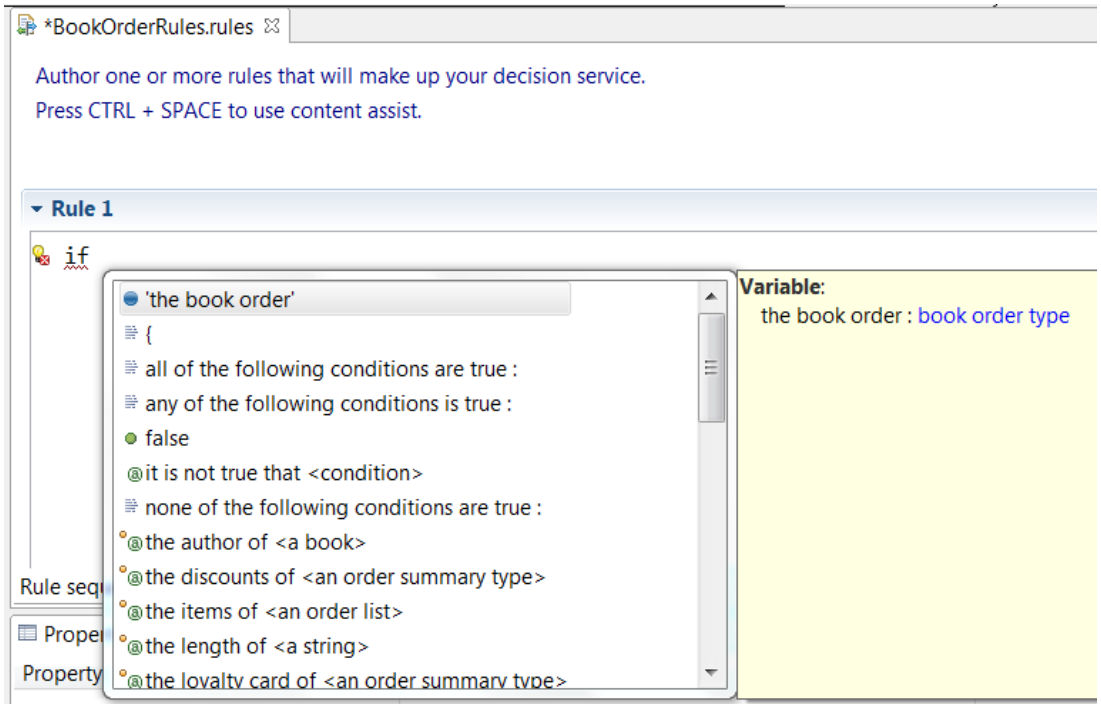


__ 8. Create a business rule. The business rule that you create in this step is shown here.

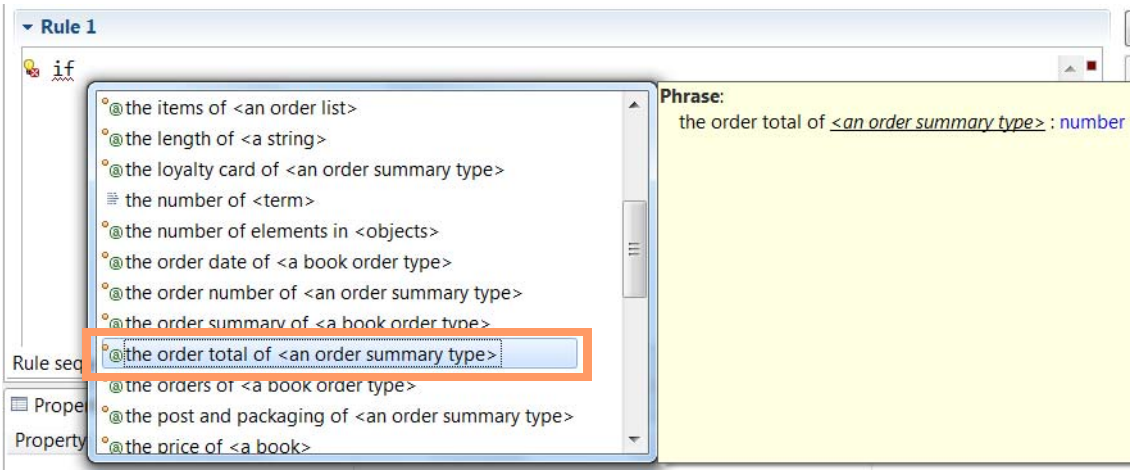
```
if the order total of the order summary of 'the book order' is at least 15
then
set the post and packaging of the order summary of 'the book order' to 0;
else
set the post and packaging or the order summary of 'the book order' to 2.50;
```

__ a. Switch to the Rule editor **Rule sequence** tab.

__ b. In the **Rule 1** pane, type the word `if` followed by a space. The content-assist window opens.

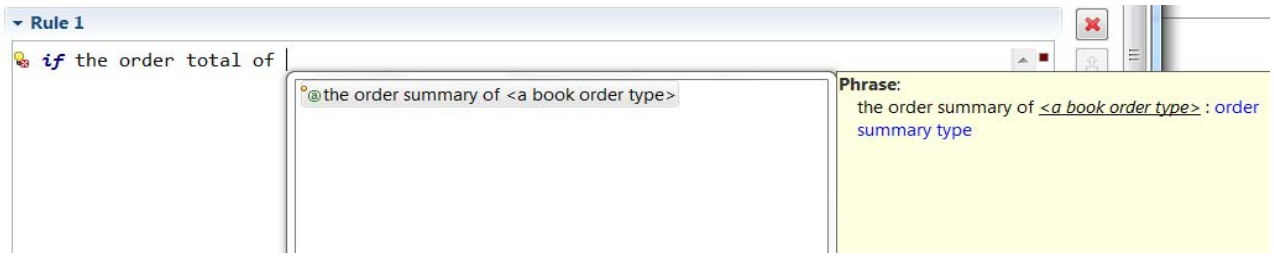


__ c. Scroll down the list and click the line: `the order total of <an order summary type>`

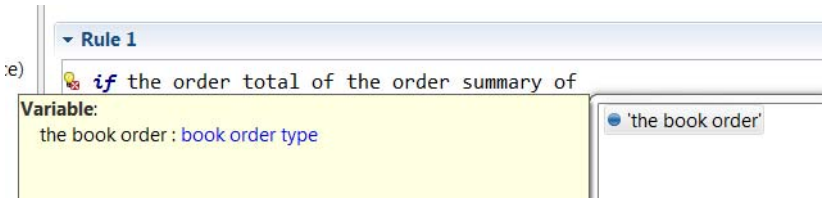


__ d. Press Enter.

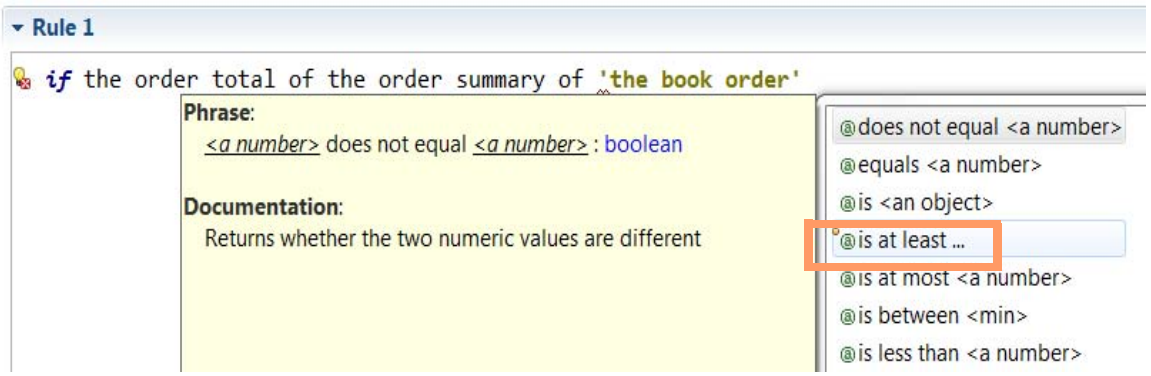
__ e. The next available option is shown. In this case, the only option is `the order summary of <a book order type>`. Press Enter to select this line.



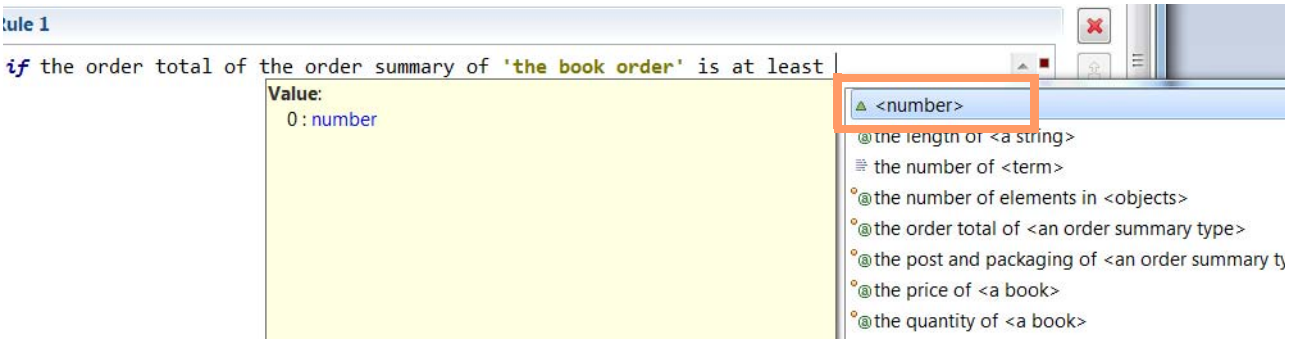
__ f. Press Enter again to select `the book order`.



__ g. Select the choice **is at least** and then press Enter.



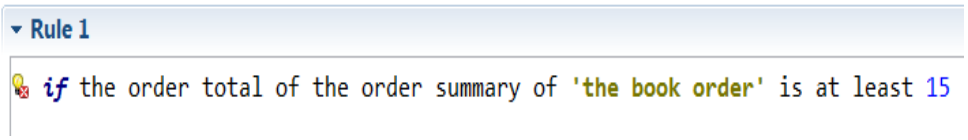
__ h. You now have the option of providing a number, or one of several variables. Click **<number>** and then press Enter.



__ i. The default value for <number> is `0`. Change this value to 15.

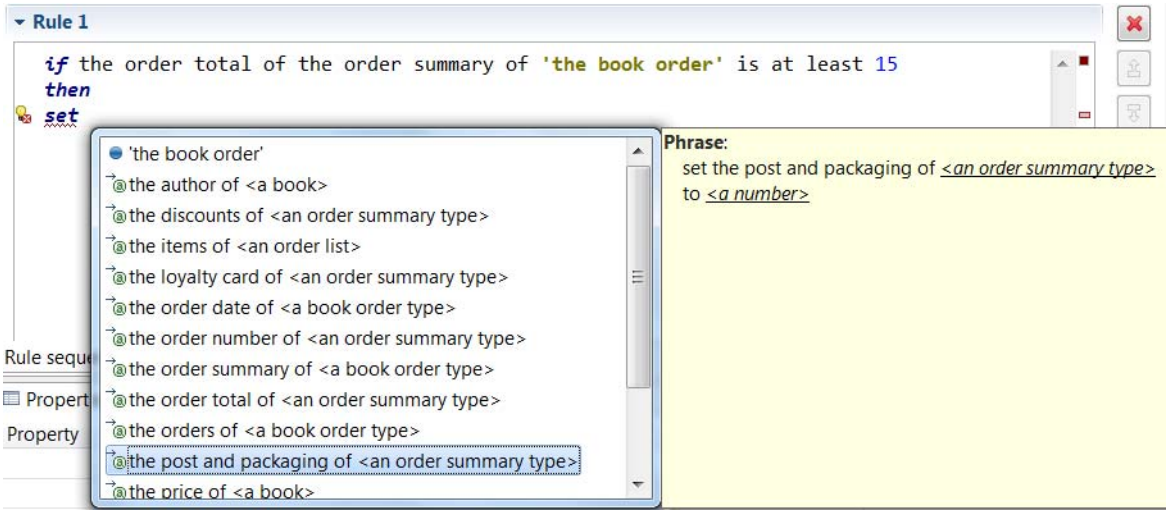
Your rule should be:

```
if the order total of the order summary of 'the book order' is at least 15
```



__ j. Type the word `then` followed by the word `set`.

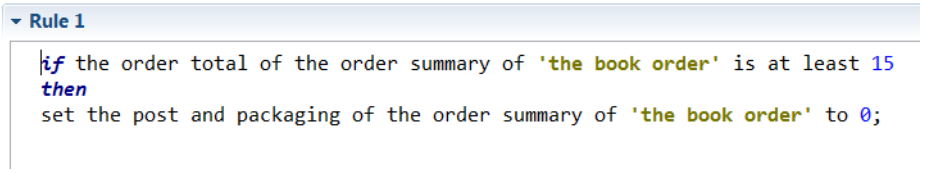
__ k. Similar to the procedure that you followed for the `if` statement, click `the post and packaging of <an order summary type>` and then press Enter.



__ l. Press Enter several times in succession to create the rule.

__ m. Finally, set the actual value to 0, followed by a semicolon.

```
set the post and packaging of the order summary of 'the book order' to 0;
```



__ n. Type: `else`

__ o. Following the same procedure that you followed for creating the `then` clause, create the `else` clause. Be sure to end the line with a semicolon.

The completed rule should be:

```
set the post and packaging or the order summary of 'the book order' to 2.50;
```

```
Rule 1
if the order total of the order summary of 'the book order' is at least 15
then
set the post and packaging of the order summary of 'the book order' to 0;
else
set the post and packaging of the order summary of 'the book order' to 2.50;
```

__ 9. Save the BookOrderRules.rules file.

Part 2: Create an integration application that uses the decision service

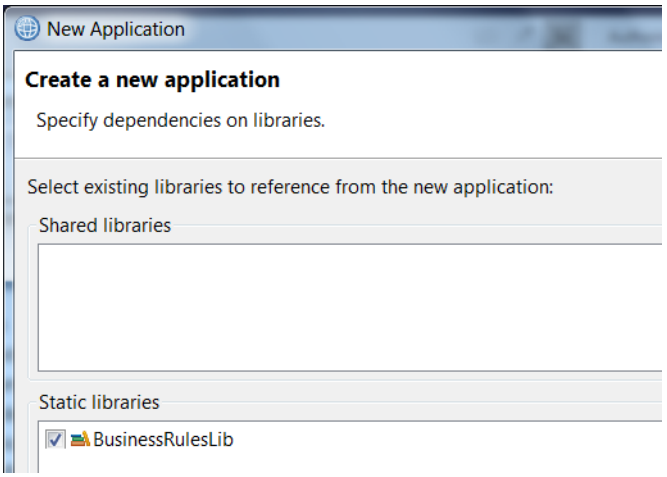
In this part of the exercise, you create an integration application. The integration application contains a message flow that references the decision service.

__ 1. Create an application that is named BusinessRulesApp that references the library that contains the decision service and the **BookOrders** schema.

__ a. Click **File > New > Application**.

__ b. For the **Application name**, type BusinessRulesApp and then click **Next**.

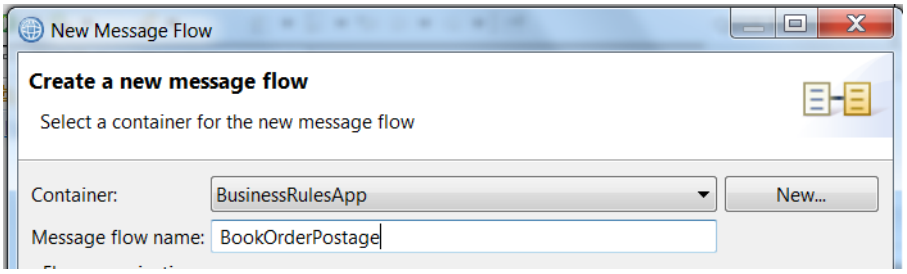
__ c. Select **BusinessRulesLib** as a dependent library and then click **Finish**.



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

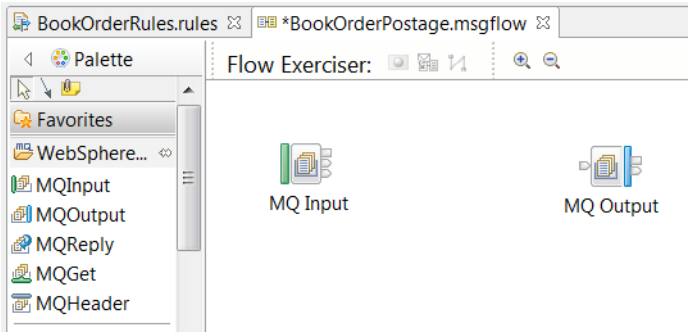
__ a. Under the **BusinessRulesApp** application in the **Application Development** view, click **New > Message Flow**.

__ b. For the **Message flow name**, type: BookOrderPostage



__ c. Click **Finish**.

__ 3. Add one MQ Input node and one MQ Output node to the message flow.



__ 4. Configure the MQ Input node properties.

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

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

__ c. For the **Message domain** on the **Input Message Parsing** tab, select **XMLNSC**.

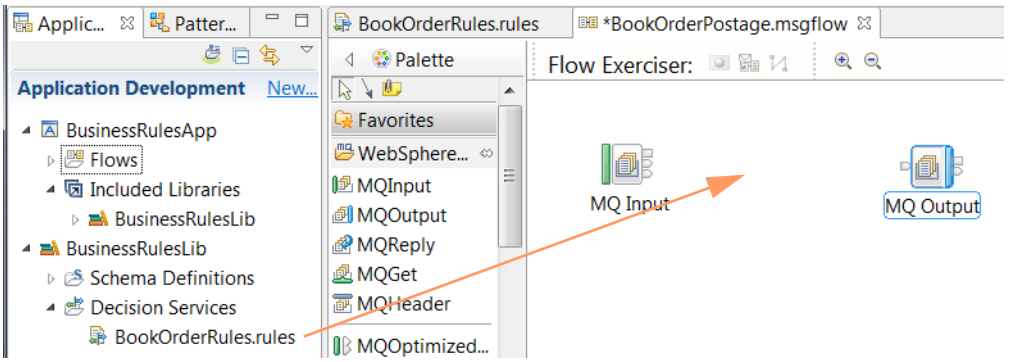
__ 5. Configure the MQ Output node queue properties

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

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

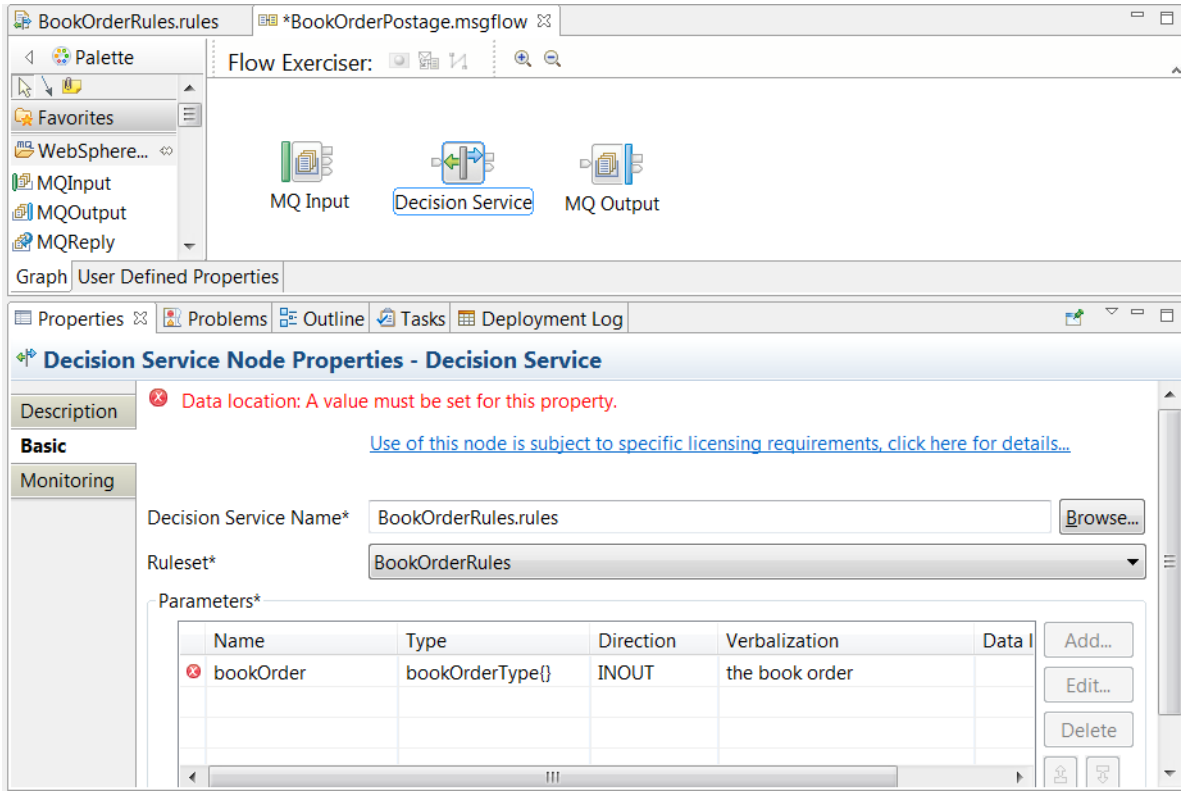
__ 6. Add the decision service to the flow.

__ a. Drag the **BookOrderRules.rules** file from the **Application Development** view onto the Message Flow editor canvas.

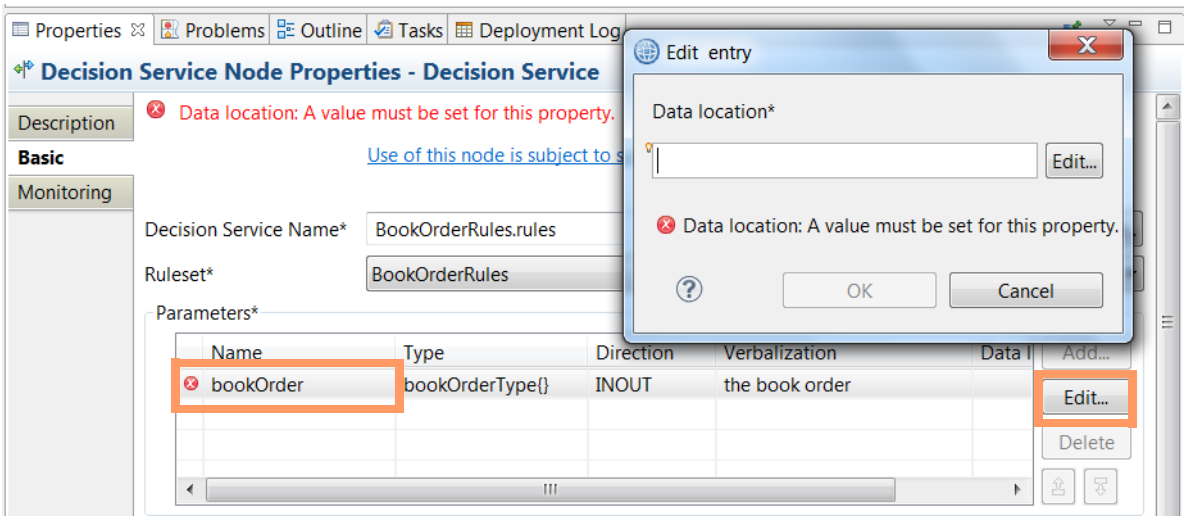


The Decision Service node is added to the Message Flow editor canvas.

__ b. An error message on the **Properties** view for the Decision Service node indicates that a data location value must be set for the ruleset that you defined in Part 1 of this exercise. The data location identifies the location of the data in the input message tree.

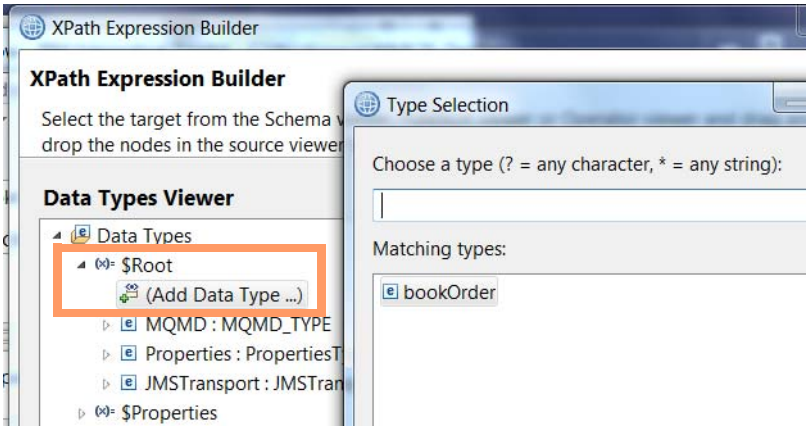


__ c. In the Decision Service node **Properties Parameters** table, click the **bookOrder** parameter and then click **Edit**.



__ d. In the **Edit entry** window, click **Edit**.

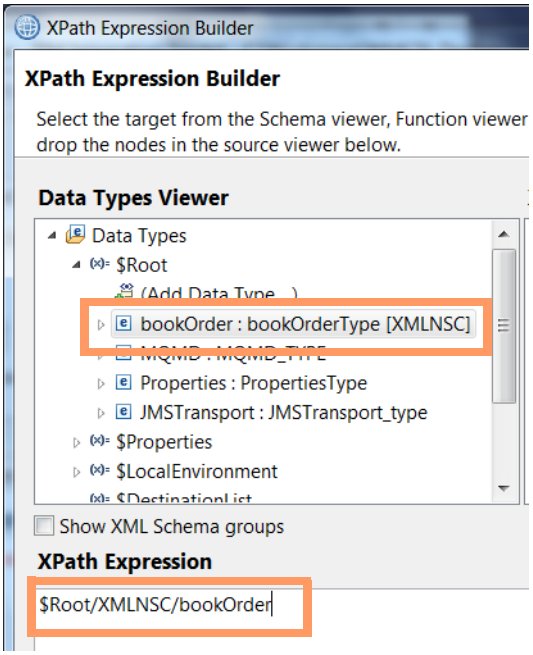
__ e. In the **XPath Expression Builder**, expand **$Root** and then click **(Add Data Type)**.



__ f. In the **Type Selection** window, click **bookOrder** and then click **OK**.

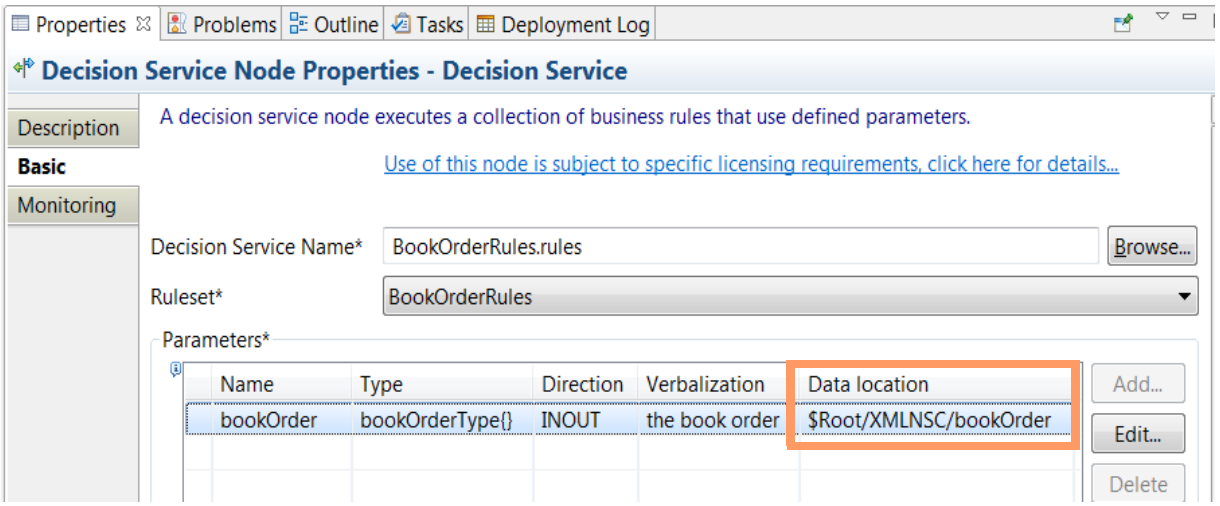
In the **Data Types Viewer** pane of the **XPath Expression Builder**, you now see the **bookOrder** data type.

__ g. Double-click the **bookOrder** data type to create the required XPath expression of `$Root/XMLNSC/bookOrder`.



__ h. Click **Finish** and then click **OK**.

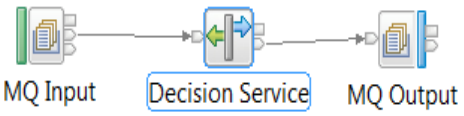
The Data location should now be set for the **bookOrder** and the error message should be gone.



__ 7. Connect the nodes.

__ a. Connect the MQ Input node **Out** terminal to Decision Service node **In** terminal.

__ b. Connect the Decision Service node **Out** terminal to the MQ Output node **In** terminal.



__ 8. Save the message flow.

__ 9. Verify that the **Problems** view does not contain any errors.

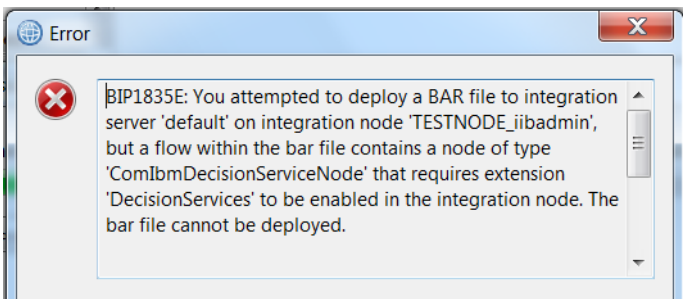
Part 3: Deploy and test the decision service

In this part of the exercise, you use the Integration Toolkit Flow exerciser to deploy and test the message flow that contains the decision service.

__ 1. Click the **Start Flow Exerciser** icon in the Message Flow editor toolbar.

__ 2. Deploy to integration server that is named **default** on the **TESTNODE_iibadmin** integration node.

__ 3. The deployment fails because Integration Bus is not enabled for use with IBM Operational Decision Manager.



__ 4. Acknowledge the error by clicking **OK** and then cancel the deployment.

__ 5. Enable DecisionServices mode in Integration Bus. In the IBM Integration Console, type:

```
mqsimode -x DecisionServices
```

Important

If you replicate this scenario on your own computer, you must ensure that you have the license to use the external Operational Decision Manager features within Integration Bus.

__ 6. Click the **Start Flow Exerciser** icon again.

__ 7. Deploy to integration server that is named **default** on the **TESTNODE_iibadmin** integration.

Verify that the deployment was successful.

__ 8. Import and send the test message `BookOrder1.xml` from the `C:\labfiles\Lab06-DecSvc\data` directory.

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

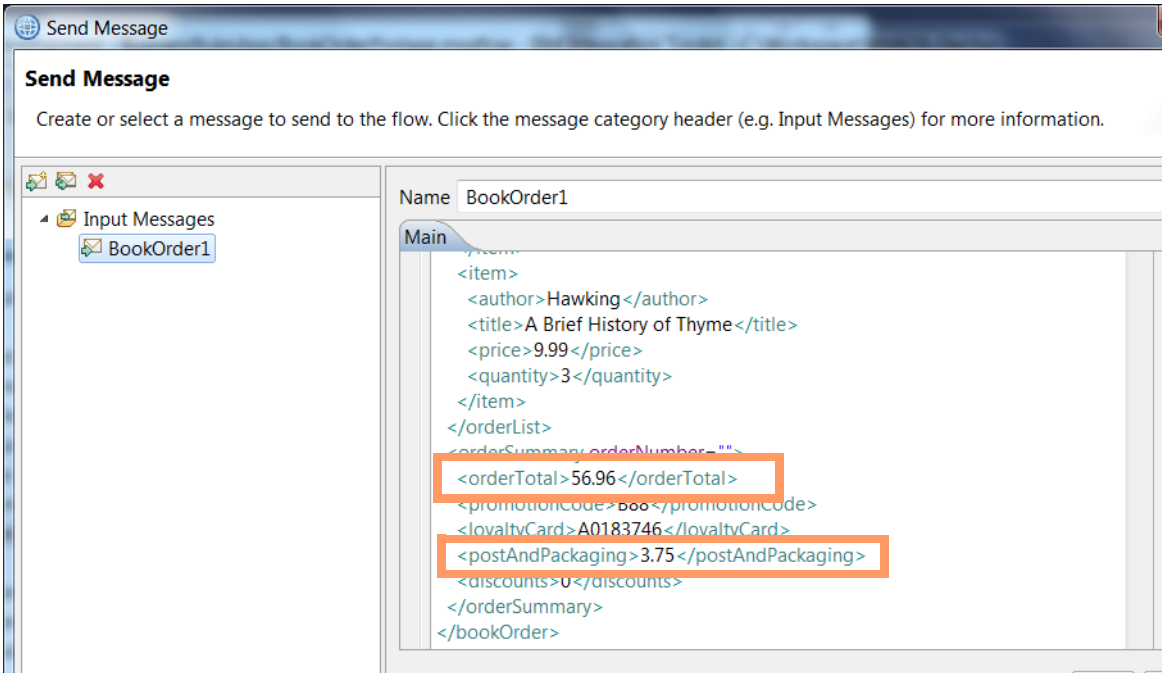
__ b. In the **Send Message** window, click the **New Message** icon.

__ c. For the message **Name**, type: `BookOrder1`

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

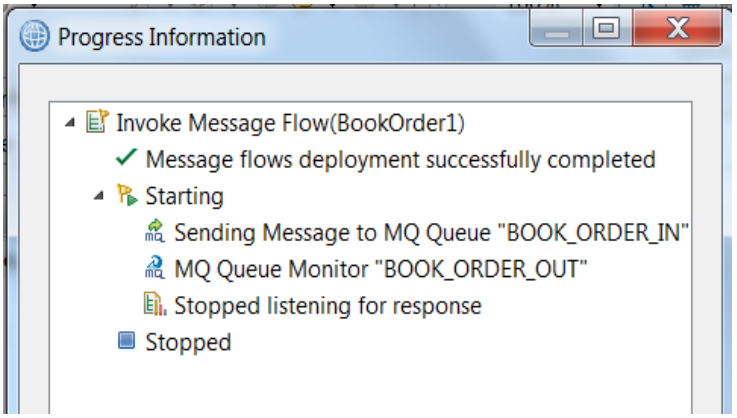
__ e. Import `BookOrder1.xml` from the `C:\labfiles\Lab06-DecSvc\data` directory.

__ f. Scroll to the bottom of the input data and verify that the <orderTotal> is 56.96, which is greater than the value specified in the decision service rule. Also, verify that the current <postAndPackaging> value is 3.75.



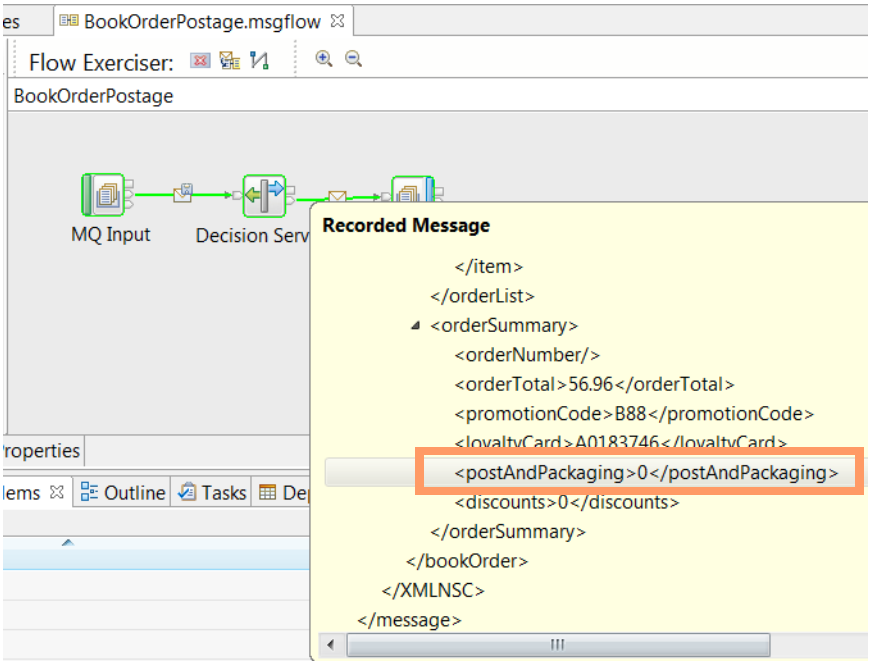
__ g. Click **Send**.

__ 9. On the **Progress Information** window, verify that the message is sent to the BOOK_ORDER_IN queue.



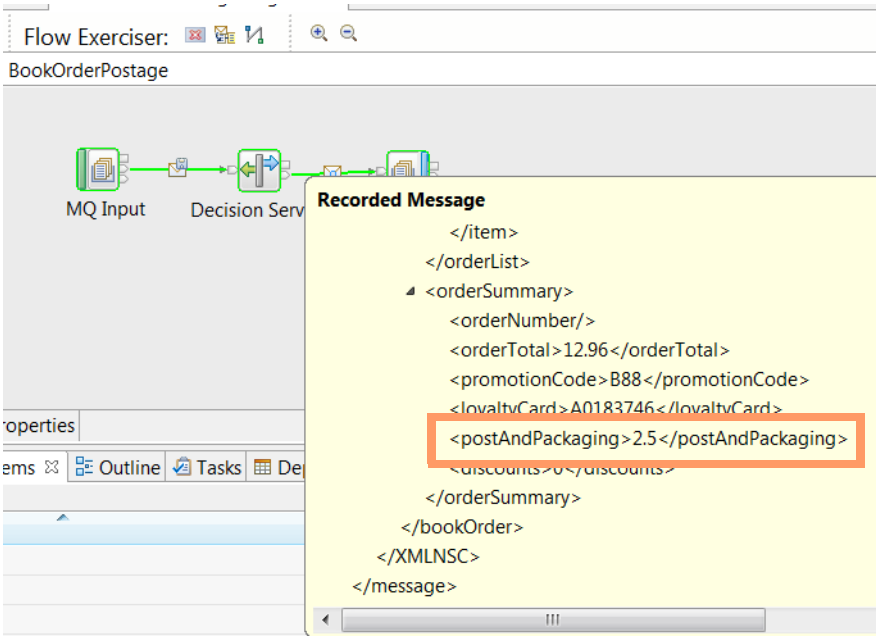
__ 10. Close the **Progress Information** window.

__ 11. In the Flow exerciser, click the letter icon between the Decision Service node and MQ Output node to view the output message. The <postAndPackaging> value in the output message should be 0 because the <orderTotal> is greater than 15.



__ 12. Run another test by using the Flow Exercise and the BookOrder2.xml file C:\labfiles\Lab06-DecSvc\data directory.

In this file, the <orderTotal> value is 12.96 and the current value for <postAndPackaging> is 3.75. The output message should have a <postAndPackaging> value of 2.5.



Exercise cleanup

__ 1. In the Message Flow editor toolbar, click the **Return flow to edit mode** icon to stop the Flow exerciser.

__ 2. Stop the Flow exercise recording mode.

In the **Integration Nodes** view, right-click the integration server that is named **default** that is running on the **TESTNODE_iibadmin** integration node, and then click **Stop recording**.

__ 3. Delete all flows and resources from the integration server that is named **default** that is running on the **TESTNODE_iibadmin** integration node.

In the **Integration Nodes** view, right-click the integration server that is named **default** that is running on the **TESTNODE_iibadmin** integration node, and then click **Delete > All flow and resources**.

__ 4. Close the open editor windows.

End of exercise

Exercise review and wrap-up

In the first part of this exercise, you created a library and then imported an XML schema that describes the data. You then created a decision service that references the XML schema. You wrote business rules by using the Integration Toolkit Rule Designer.

In the second part of the exercise, you implemented the decision service in a message flow application.

In the third part of the exercise, you tested the application by using the Integration Toolkit Flow exerciser.

Having completed this exercise, you should be able to:

- Create a decision service
- Write business rules in the IBM Integration Toolkit Rule Designer
- Test the decision service

.

Exercise 7. Implementing IBM Integration Bus runtime security

What this exercise is about

In this exercise, you implement the IBM Integration Bus runtime security to allow security credentials to be propagated in a message flow.

What you should be able to do

After completing this exercise, you should be able to:

- Propagate security credentials within a message flow
- Configure message flow nodes to enable secure message processing
- List the types of security tokens
- Use a Compute node to simulate the actions of an external security provider

Introduction

With IBM Integration Bus, you can implement security control on individual message flows. Without this capability, the message transports that deliver messages to message flows for processing are responsible for providing security. By using the IBM Integration Bus security functions, you can authenticate and authorize each message when it is presented for processing in a message flow. You can also propagate the security credentials to most output transports.

When you use the IBM Integration Bus security manager, you can:

- **Extract** security credentials from the message
- **Authenticate** the credentials
- **Map** the credentials to alternative credentials
- **Authorize** the original or alternative credentials to access the message flow
- **Propagate** the credentials through the message flow to the outbound message transport

To authenticate, map, and authorize, you must use an external security provider. The image for this course does not include an external security provider. In this exercise, you work with the message transport nodes and propagate the message. You also simulate the credential mapping operation of an external security provider by using a Compute node.

Message flow security does not require the integration node to have a default queue manager. The message flow in this exercise uses the integration node

HTTP listener, which requires that integration node use a default queue manager.

In the first part of this exercise, you review the message flows that this exercise uses. You also deploy the message flows.

In the remaining parts of this exercise, you use the Integration Toolkit Unit Test Client to test the message flow for three scenarios.

- Test 1: IBM MQ message contains security information in the message header
- Test 2: IBM MQ message contains security information in the message body
- Test 3: IBM MQ message requires mapped credentials

Requirements

- IBM Integration Bus V10
- IBM MQ V8 with a queue manager that is named IIBQM that contains the local application queues: SECURITYIDFROMMQIN, SECURITYIDFROMMQOUT, SECURITYIDFROMMSGIN, SECURITYIDFROMMSGOUT
- The integration node IIBNODE_WITHQM with the default the queue manager IIBQM that you created in Exercise 1, Part 1
- Lab files in the `C:\labfiles\Lab07-Security` directory

Exercise instructions

Exercise setup

__ 1. If it is not already running, start the IBM Integration Toolkit.

__ 2. Switch to a new workspace that is named: `C:\Workspace\security`

When the **Welcome** window is displayed, close it. The Integration Development perspective is displayed.

__ 3. Import the starting projects from the project interchange file.

__ a. From the menu bar, click **File > Import**.

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

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

__ d. Browse to `C:\labfiles\Lab07-Security\resources\SecurityImportPI.zip`, and then click **Open**. The **Import Projects** dialog box is displayed.

__ e. Ensure that all projects are selected, and then click **Finish**.

The projects are imported. Wait for the workspace build to complete.

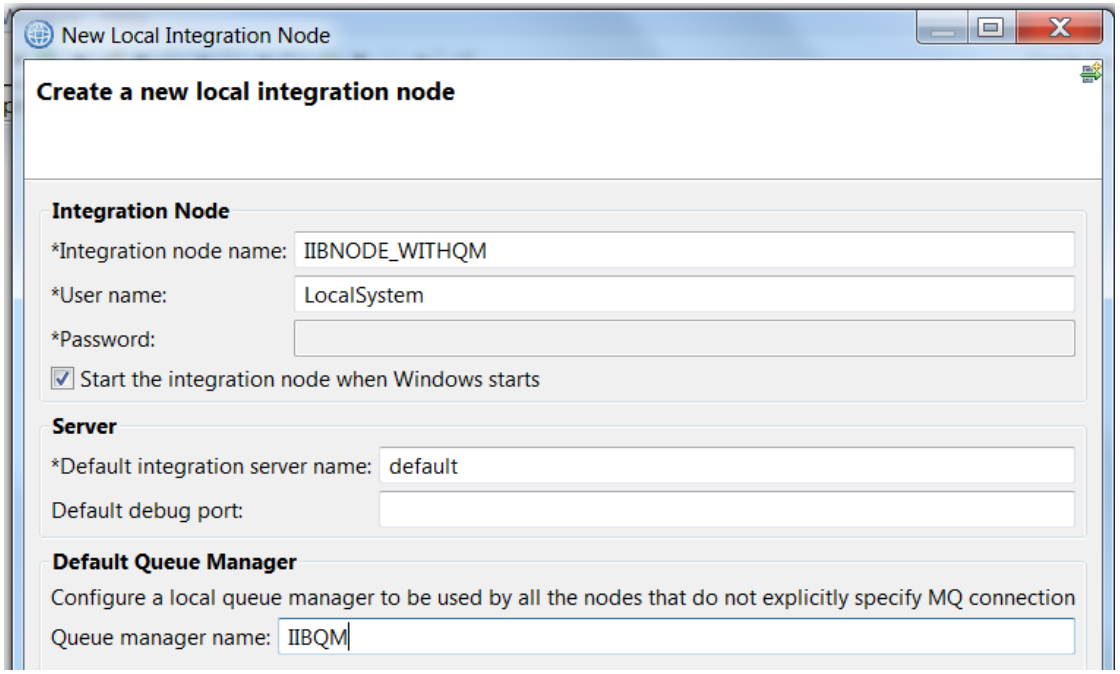
__ 4. The configuration of the IBM MQ nodes and the HTTP nodes in this exercise assume that the integration node has a default queue manager. The instructions for this exercise use the integration node `IIBWITHQM` and queue manager `IIBQM` that you created in Part 1 of Exercise 1.

If you already have an integration node that is named `IIBNODE_WITHQM` that has a default queue manager that is named `IIBQM`, go to step 5.

If do not have an integration node that is named `IIBNODE_WITHQM` that has a default queue manager that is named `IIBQM`, complete the next steps.

__ a. In IBM MQ Explorer, create a queue manager that is named `IIBQM` with a dead-letter queue that is named `DLQ`.

__ b. In the Integration Toolkit, create a local integration node that is named `IIBNODE_WITHQM` with a default integration server that is named `default` and a queue manager that is named `IIBQM`.



__ c. Create the SYSTEM.BROKERS queues on the integration node queue manager by running the `iib_createqueues.cmd` file.

In an IBM Integration Console as an Administrator, type the following commands.

```
cd server\sample\wmq
iib_createqueues.cmd IIBQM
```

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

In the IBM Integration Console, type:

```
runmqsc IIBQM < C:\labfiles\Lab07-Security\resources\Q_Defs.mqsc
```

__ 6. This exercise uses the Integration Toolkit Unit Test Client. Enable the Unit Test Client menus.

__ a. Click **Window > Preferences**.

__ b. Expand **Integration Development** and then click **Unit Test Client**.

__ c. Click **Enable menus for test client** and then click **OK**.

Part 1: Review the message flows

The message flow application contains three message flows. In this part of the exercise, you review each message flow.

__ 1. In the **Application Development** view, expand **Independent Resources > SecurityIdentitySampleFlowProject > Flows**.

Information

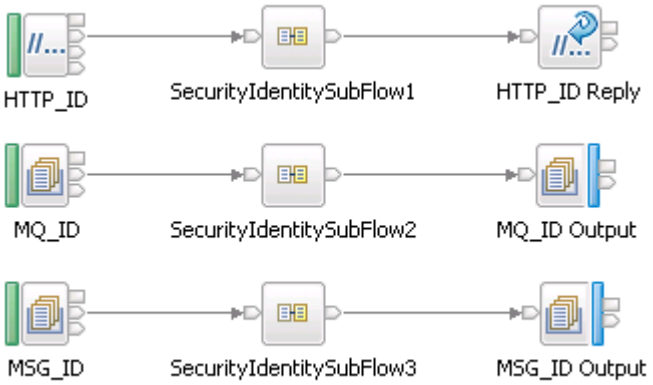
A Warning message in the **Problems** view indicates that the subflow is defined as a `.msgflow` file instead of a `.subflow` file, which requires special handling when creating a `.bar` file.

If you want to convert this file to a `.subflow` to eliminate this warning message, right-click the **SecuritySampleIdentity.msgflow** file in the Application Development view and then click **Convert to subflow**.

In this exercise, you do not create the BAR file but use the BAR file that is provided in the BARs folder.

__ 2. Double-click **SecurityIdentitySampleFlow.msgflow** to open it in the Message Flow editor.

This message flow file has three flows. Each flow does similar processing; the only difference is in the transport protocols that are used.



- The top flow receives a message from HTTP Input node **HTTP_ID**, calls the SecurityIdentitySubflow, and then writes the resulting message assembly to an HTTP Reply node.
- The middle flow receives a message from MQ Input node **MQ_ID**, calls the SecurityIdentitySubflow, and then writes the resulting message assembly to an MQ Output node.
- The bottom flow receives a message from MQ Input node **MSG_ID**, calls the SecurityIdentitySubflow, and then writes the resulting message assembly to an MQ Output node. This flow is similar to the second flow, except for the security parameters that are specified on the MQ Input node.

In the next steps, you review the transport properties that are associated with the three flows.

__ a. Review the node properties for the **HTTP_ID** node and answer the following questions.

Question 1: On the **Basic** tab, what is the **Path Suffix for URL** value?

_____

Note

The answers to the questions are listed in the last section of the exercise.

Question 2: On the **Security** tab, what is the **Identity token type** value?

_____

Review the other possible values for **Identity token type**.

__ b. Review the node properties for the **MQ_ID** node and answer the following questions.

Question 3: On the **Basic** tab, what is the input queue?

_____

Question 4: On the **Security** tab, what is the **Identity token type** value?

_____

__ c. Review the node properties for the **MQ_ID Output** node and answer the following question.

Question 5: On the **Basic** tab, what queue does the node write to?

_____

__ d. Review the properties for the **MSG_ID** node and answer the following questions.

Question 6: On the **Basic** tab, what is the input queue?

_____

__ e. **Question 7:** On the **Security** tab, what is the **Identity token type** value?

_____

Question 8: How does this compare to the **MQ_ID** node? What other differences do you see on this tab?

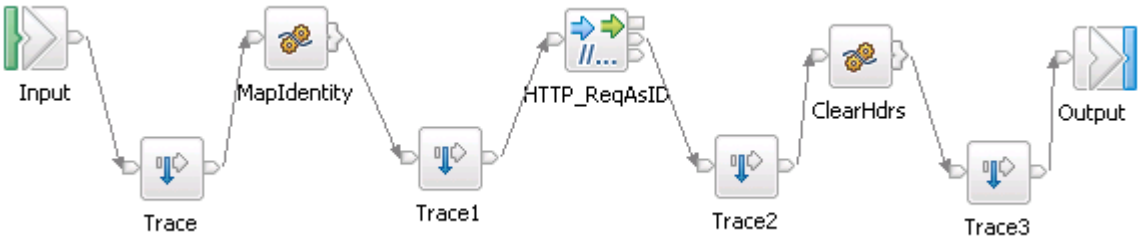
_____

Review the properties for the **MSG_ID Output** node and answer the following question.

__ 3. **Question 9:** On the **Basic** tab, what queue does the node write to?

_____

Double-click any of the **SecurityIdentitySubFlow** nodes to open the subflow in the Message Flow editor.



The subflow does most of the work in this exercise. It contains a **MapIdentity** Compute node, an **HTTP_ReqAsID** HTTP Request node, and a **ClearHdrs** Compute node. Between each of these nodes, and following the **ClearHdrs** node, are **Trace** nodes.

__ a. Double-click the **MapIdentity** node. This action opens the ESQL editor.

Question 10: Explain briefly what the **MapIdentity** node does.

_____

_____

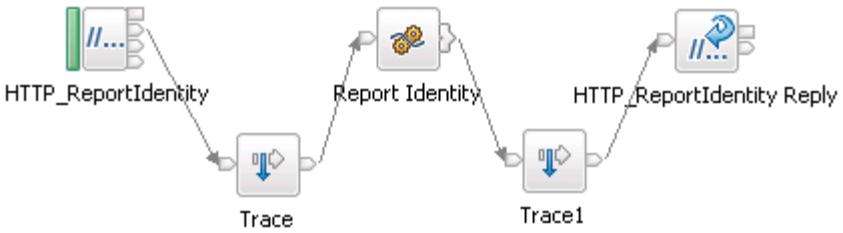
_____

Question 11: Examine the properties of the **HTTP_ReqAsID** node. It is an HTTP request node that calls the message flow that is contained in **SecurityIdentityReportFlow.msgflow**. How is this action accomplished?

_____

_____

__ 4. Open **SecurityIdentityReportFlow.msgflow** in the Message Flow editor.



This independent main flow is called as a service provider. It adds security report information to the message assembly. Trace nodes are embedded within this flow also.

__ a. Review the properties of the **HTTP_ReportIdentity** node

Question 12: On the **Security** tab, what is the **Identity token type** value?

_____

__ b. Double-click the **ReportIdentity** node. This action opens the ESQL editor.

Question 13: Explain briefly what this node does.

_____

_____

_____

__ 5. Deploy the components to the integration server that is named **default** on the integration node that is named **IIBNODE_WITHQM**.

__ a. Expand **BARs** in the **Application Development** view.

__ b. Right-click **SecurityIdentityPropagation.bar**, and then click **Deploy**. The Deploy dialog box is displayed.

__ c. Select the integration server that is named **default** on the integration node that is named **IIBNODE_WITHQM** and then click **Finish**. The BAR file is deployed.

Note

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

Test 1: IBM MQ message contains security information in the message header

In this part of the exercise, you use the Integration Toolkit Test Client to test the message flows and analyze the results.

In this test, you submit a simple message that contains an empty message body. The MQMD for the incoming message contains the following security items:

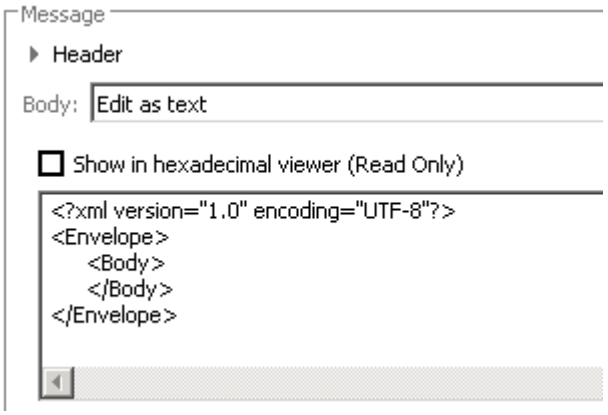
- A user name in the MQMD **User ID** field
- An “issued by” (the name of the application or user who sent the message) in the MQMD **Put application name** field

__ 1. Double-click the **Security_Identity_MQ_ID.mbtest** file under the **SecurityIdentityFlowProject > Flow Tests** folder in the **Application Development** view to open it in the Unit Test Client.

__ 2. Review the test conditions and configuration.

__ a. In the **Message Flow Test Events** section, click **Enqueue**.

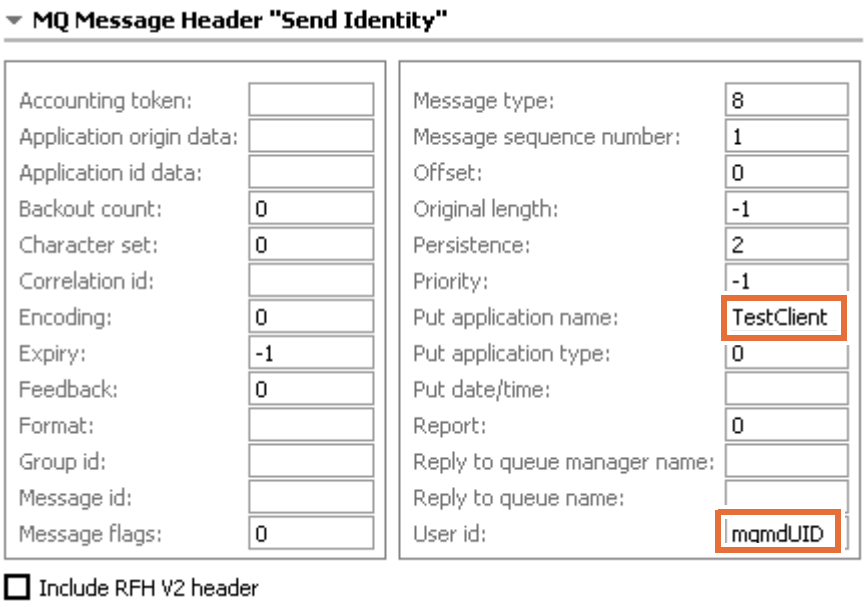
__ b. In the **Detailed Properties** section, review the message that you are going to send. You see that it contains an envelope and an empty message body.



__ c. Click the **Configuration** tab at the bottom of the Unit Test Client.

__ d. In the **Test Client Configuration** section, expand **MQ Message Headers**, and then click **MQ Message Header "Send Identity"**.

__ e. Review the **MQ message header** (MQMD). Observe that the **Put application name** and **User ID** elements of the MQMD are populated. It might be necessary to enlarge the window to see the values.



__ f. Click the **Events** tab at the bottom of the Test Client window.

__ g. To start the test, click **Send Message**.

__ h. After a few seconds, the message flow runs.

Open IBM MQ Explorer and verify that the message was enqueued to the **SECURITYIDFROMMQOUT** queue (the **Current queue depth** is now 1).

SECURITYIDFROMMQIN	Local	1	0	0
SECURITYIDFROMMQOUT	Local	0	1	1
SECURITYIDFROMMSGIN	Local	1	0	0
SECURITYIDFROMMSGOUT	Local	0	0	0

__ 3. Review the results of the test.

Hint

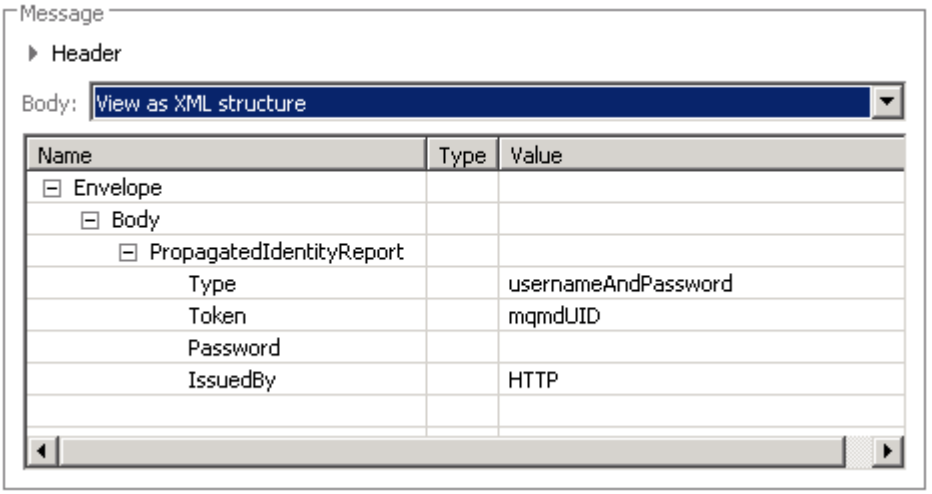
Use the output from the **Trace** nodes to verify the contents of the message assembly as the message flow was running. The Trace nodes write their output to `C:\securitytrace.txt.` You can verify the output destination by reviewing the properties for any of the Trace nodes.

Question 14: Did the **MapIdentity** Compute node in the subflow generate values for the **IdentityMapped** elements in the OutputRoot.Properties tree of the message assembly? Why or why not?

_____

__ a. Examine the message that was written to IBM MQ by the **MQ_ID Output** node in the main flow. Review the node properties to determine the queue that was used.

In the Unit Test Client, click **Dequeue** in the **Message Flow Test Events** section, and then click **Get Message** under the **Detailed Properties**.



Did you expect this result? _____

__ b. Use the trace output to compare the **IdentitySource** elements in the message assembly before **HTTP_ReqAsID** was started, with the elements after **HTTP_ReqAsID** was run. Before **HTTP_ReqAsID**:

```
(0x03000000:NameValue):IdentitySourceType     = 'username' (CHARACTER)
(0x03000000:NameValue):IdentitySourceToken    = 'mqmdUID' (CHARACTER)
(0x03000000:NameValue):IdentitySourcePassword = '' (CHARACTER)
(0x03000000:NameValue):IdentitySourceIssuedBy = 'TestClient' (CHARACTER)
```

After **HTTP_ReqAsID** (before **ReportIdentity** node in **ReportFlow** flow):

```
(0x03000000:NameValue):IdentitySourceType     = 'usernameAndPassword' (CHAR
(0x03000000:NameValue):IdentitySourceToken    = 'mqmdUID' (CHARACTER)
(0x03000000:NameValue):IdentitySourcePassword = '' (CHARACTER)
(0x03000000:NameValue):IdentitySourceIssuedBy = 'HTTP' (CHARACTER)
```

The **IdentitySourceType** changed from `username` to `usernameAndPassword`, and the **IdentitySourceIssuedBy** changed from `TestClient` to `HTTP`.

Question 15: Can you explain why this action happened?

_____

_____

_____

__ 4. Close the Unit Test Client window. Do not save changes in case you are prompted to do so.

__ 5. Delete or rename the trace output file.

__ a. Because the IBM Integration Bus runtime engine holds the trace file open, you must stop the integration server to release the file lock.

In the **Integration Nodes** view, right-click the integration server that is named **default** on the IIBNODE_WITHQM integration node and then click **Stop**. Wait for the integration server to end.

__ b. You can now use Windows Explorer to delete the trace output file `C:\securitytrace.txt`. If you want to keep it for comparison with subsequent test runs, rename the file instead of deleting it.

__ c. Restart the integration server by right-clicking it and then clicking **Start** in the **Integration Nodes** view. Wait for the integration server to start.

Test 2: IBM MQ message contains security information in the message body

In this test, you submit a message that contains security credentials within the message itself, instead of in the transport header. IBM MQ does not provide a location to include a password within the MQMD. Passing the user ID and password credentials as part of the message body, rather than

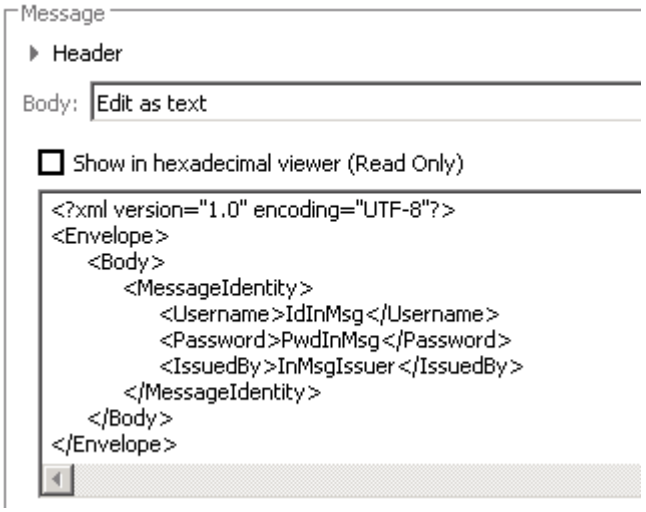
just the user ID in the MQMD, is one alternative approach. In this test, the user ID, password, and "issued by" values are supplied in the `Body.MessageIdentity` folder of the incoming message.

__ 1. Double-click the **Security_Identity_MSG_ID.mbtest** flow test file to open it in the Test Client.

__ 2. Review the test conditions and configuration.

__ a. In the **Message Flow Test Events** section, click **Enqueue**.

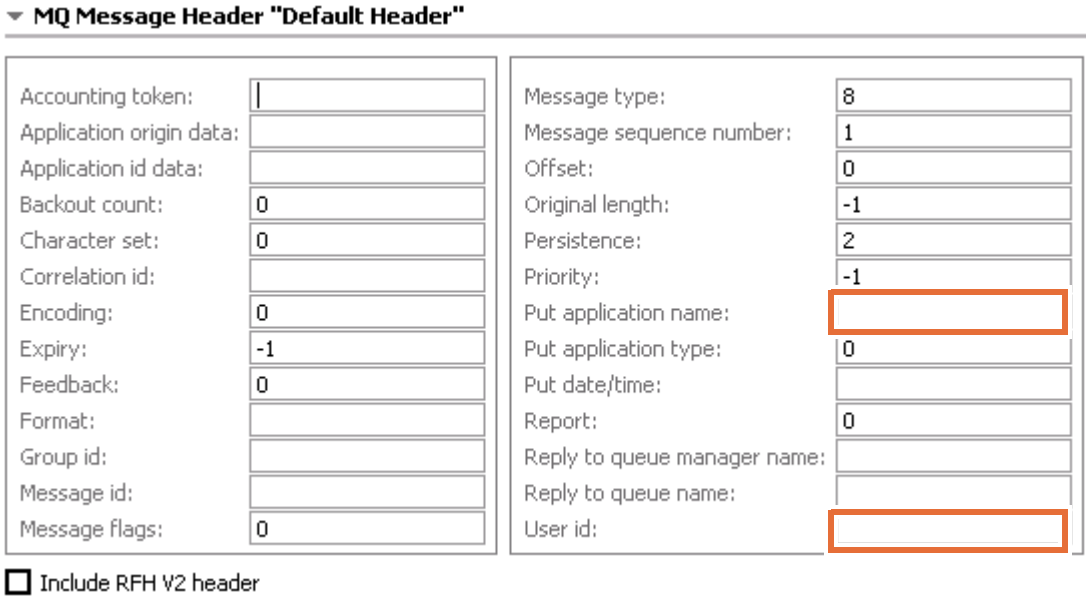
__ b. In the **Detailed Properties** section, review the message that you are going to send. You see that the message body contains a **MessageIdentity** tag that contains the security credentials.



__ c. Click the **Configuration** tab at the bottom of the Unit Test Client window.

__ d. In the **Test Client Configuration** section, expand **MQ Message Headers**, and then click **MQ Message Header "Default Header"**.

__ e. Review the **MQ message header** (MQMD). This time the **User ID** and **Put application name** properties are empty.



__ f. Click the **Events** tab at the bottom of the Unit Test Client window.

__ g. To start the test, click **Send Message**.

__ h. After a few seconds, the message flow runs. Again, you can verify that the message was enqueued on the **SECURITYIDFROMMSGOUT** queue by using IBM MQ Explorer.

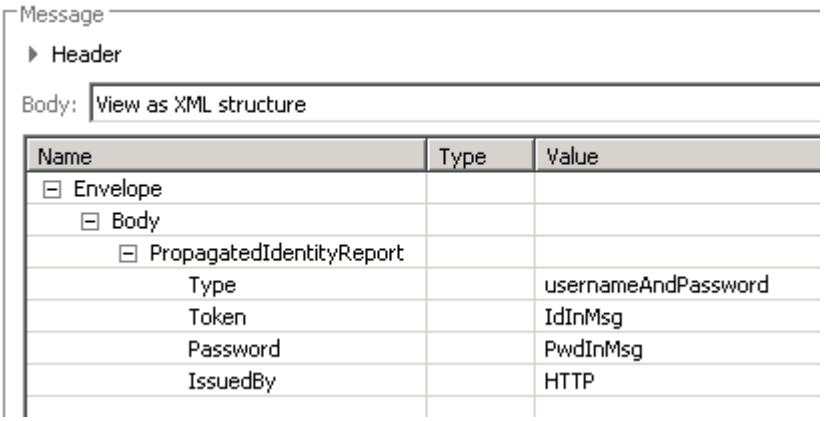
__ 3. Review the results of the test.

Question 16: Did the **MapIdentity** Compute node in the subflow generate values for the **IdentityMapped** elements in the `OutputRoot.Properties` tree of the message assembly? Why or why not?

_____

__ a. Review the message that was written to IBM MQ by the **MSG_ID Output** node in the main flow.

In the Unit Test Client, click **Dequeue** in the **Message Flow Test Events** section, and then click **Get Message** under **Detailed Properties**.



Did you expect this result? _____

__ b. Use the trace output to compare the **IdentitySource** elements in the message assembly before **HTTP_ReqAsID** was started, with the elements after **HTTP_ReqAsID** was run.

Before **HTTP_ReqAsID**:

```
(0x03000000:NameValue):IdentitySourceType     = 'usernameAndPassword' (CHAR
(0x03000000:NameValue):IdentitySourceToken    = 'IdInMsg' (CHARACTER)
(0x03000000:NameValue):IdentitySourcePassword = 'PwdInMsg' (CHARACTER)
(0x03000000:NameValue):IdentitySourceIssuedBy = 'InMsgIssuer' (CHARACTER)
```

After **HTTP_ReqAsID**:

```
(0x03000000:NameValue):IdentitySourceType     = 'usernameAndPassword'
(0x03000000:NameValue):IdentitySourceToken    = 'IdInMsg' (CHARACTER)
(0x03000000:NameValue):IdentitySourcePassword = 'PwdInMsg' (CHARACTER)
(0x03000000:NameValue):IdentitySourceIssuedBy = 'HTTP' (CHARACTER)
```

Because the same message flow logic ran for this test as for the first test, the results are similar, even though the message credentials were carried in the message body rather than the message header.

Again, you saw the **IdentitySourceIssuedBy** change (from `InMsgIssuer` to `HTTP`) because of the properties of the **HTTP_ReportIdentity** HTTP Input node in the **SecurityIdentityReportFlow**. The **IdentitySourceType** was already set to `usernameAndPassword`, so you cannot tell by the trace output whether the HTTP Input node also changed it (although it did).

__ 4. Close the Test Client window. Do not save changes in case you are prompted to do so.

__ 5. Delete or rename the trace output file, as you did in the previous test.

Remember to stop the integration server before renaming or deleting the file. Remember also to restart the integration server.

Test 3: IBM MQ message requires mapped credentials

In many cases, it is necessary to map, or transform, a set of credentials from one form to another. For example, it might be necessary to map a user ID and password that are used in one environment to a corresponding user ID and password for use in a different environment. Because a message flow can access multiple environments within the same message flow (such as a web service environment and an IBM MQ environment), mapping of identities between the environments is a necessary task.

Identity mapping (also known as *identity federation*) is often done with an external security provider. Credential information is sent to the security provider, and the corresponding alternative credentials are returned. The mapped credentials are stored (along with the original credentials) in the `Properties` folder of the message assembly.

As you saw in the previous two tests, the MQMD provides a location for a user ID only, not a password. Therefore, mapping is often used in message flows that use IBM MQ as the message transport, by using the user ID and “Issued by” values. In this test, the message flow simulates

credential mapping by using the **MapIdentity** Compute node in the **SecurityIdentitySubflow**, instead of using an external security provider.

In this test, an IBM MQ message is sent to the message flow. The MQMD contains a user ID and a “Put application ID” (which acts as the “Issued by”) which act as the message credentials. The body of the message contains a MapIdentity folder, which causes the **MapIdentity** Compute node to complete these mappings:

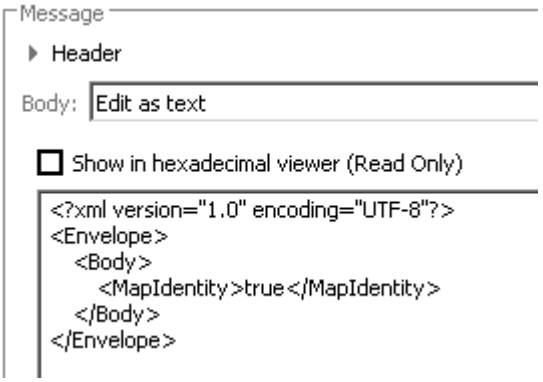
- The **IdentityMappedToken** (mapped user ID) is created by converting the user ID (from the MQMD) to lowercase, and then appending `@company.com`.
- The **IdentityMappedPassword** is created by concatenating “p_” to the front of the lowercase user ID and concatenating the four-digit year to the end.
- The **IdentityMappedType** is set to `usernameAndPassword`.
- The **IdentityMappedIssuedBy** is set to the constant “`SecurityIdentitySubFlow_MapIdentity`”.

__ 1. Double-click the **Security_Identity_Mapped.mbtest** file to open it in the Unit Test Client.

__ 2. Review the test conditions and configuration.

__ a. In the **Message Flow Test Events** section, click **Enqueue**.

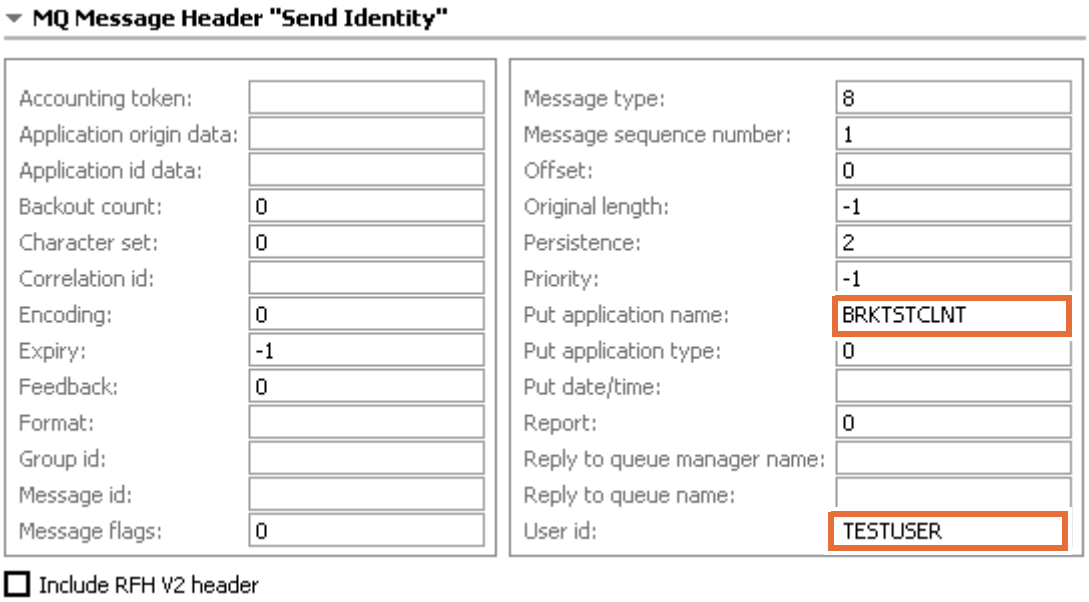
__ b. In the **Detailed Properties** section, review the message that you are going to send. You see that the message body contains a `MapIdentity` folder.



__ c. Click the **Configuration** tab at the bottom of the Unit Test Client window.

__ d. In the **Test Client Configuration** section, expand **MQ Message Headers**, and then click **MQ Message Header “Send Identity”**.

__ e. Review the **MQ message header** (MQMD). The **Put application name** and **User ID** elements of the MQMD are populated.



__ f. Click the **Events** tab at the bottom of the Unit Test Client window.

__ g. To start the test, click **Send Message**.

__ h. After a few seconds, the message flow runs.

__ i. Verify that the message was enqueued on the **SECURITYIDFROMQOUT** queue by using IBM MQ Explorer.

__ 3. Review the results of the test.

Question 17: Did the **MapIdentity** Compute node in the subflow generate values for the **IdentityMapped** elements in the OutputRoot.Properties tree of the message assembly? Why or why not?

_____

_____

_____

_____

__ a. Examine the trace output that the first **Trace** node in the SecurityIdentityReportFlow generated (the text in the trace output is labeled “Before Report Identity”).

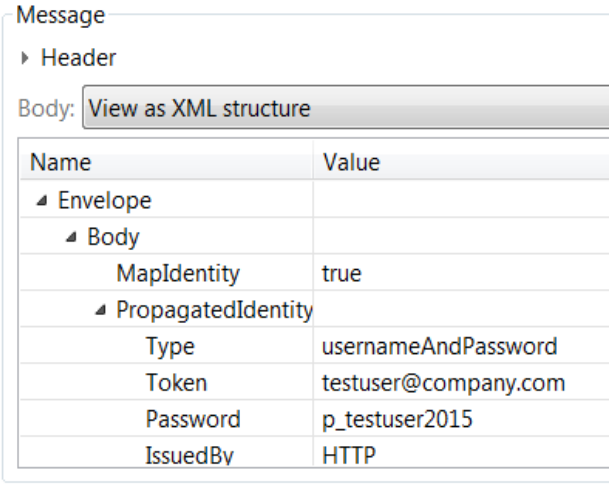
```
(0x03000000:NameValue):IdentitySourceType     = 'usernameAndPassword'
(0x03000000:NameValue):IdentitySourceToken    = 'testuser@company.com'
(0x03000000:NameValue):IdentitySourcePassword = 'p_testuser2012' (CHAR
(0x03000000:NameValue):IdentitySourceIssuedBy = 'HTTP' (CHARACTER)
(0x03000000:NameValue):IdentityMappedType     = '' (CHARACTER)
(0x03000000:NameValue):IdentityMappedToken    = '' (CHARACTER)
(0x03000000:NameValue):IdentityMappedPassword = '' (CHARACTER)
(0x03000000:NameValue):IdentityMappedIssuedBy = '' (CHARACTER)
```

Question 18: Compare that trace output to the output generated after the message exits the MapIdentity node (in the previous step). What changed between the two traces?

_____

__ b. Review the message that was written to IBM MQ by the **MQ_ID Output** node in the main flow.

In the Test Client, click **Dequeue** in the **Message Flow Test Events** section, and then click **Get Message** under the **Detailed Properties**.



Did you expect this result? _____

__ 4. Close the Unit Test Client. Do not save changes if you are prompted to do so.

Exercise cleanup

__ 1. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, in case you are prompted.

__ 2. In the **Integration Nodes** view, right-click the integration server that is named **default** on the IIBNODE_WITHQM integration node, and then click **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.

End of exercise

Answers to exercise questions

Question 1: On the **Basic** tab, what is the **Path Suffix for URL** value?

Answer: /Security/IdentityFromHttp

Question 2: On the **Security** tab, what is the **Identity token type** value?

Answer: Transport Default

Question 3: On the **Basic** tab, what is the input queue?

Answer: SECURITYIDFROMMQIN

Question 4: On the **Security** tab, what is the **Identity token type** value?

Answer: Transport Default

Question 5: On the **Basic** tab, what queue does the node write to?

Answer: SECURITYIDFROMMQOUT

Question 6: On the **Basic** tab, what is the input queue?

Answer: SECURITYIDFROMMSGIN

Question 7: On the **Security** tab, what is the **Identity token type** value?

Answer: Username + Password.

Question 8: How does this compare to the **MQ_ID** node? What other differences do you see on this tab?

Answer: The other properties on this tab are populated. Each specifies a location in the inbound message assembly where the security credentials (token location, password location, and IssuedBy location) are found.

Question 9: On the **Basic** tab, what queue does the node write to?

Answer: SECURITYIDFROMMSGOUT

Question 10: Explain briefly what the MapIdentity node does.

Answer: The node attempts to set a pointer to MapIdentity folder in the body of the InputRoot envelope (`InputRoot.XMLNSC.Envelope.Body.MapIdentity`). If it exists, the elements of the `OutputRoot.Properties` mapped identity fields are populated.

Question 11: Examine the properties of the **HTTP_ReqAsID** node. It is an HTTP request node that calls the message flow that is contained in **SecurityIdentityReportFlow.msgflow**. How is this action accomplished?

Answer: The value that is specified in the **Web service URL** property matches the **Path suffix for URL** property of the HTTP input node that starts the **SecurityIdentityReportFlow** message flow.

Question 12: Review the properties of the **HTTP_ReportIdentity** node. On the **Security** tab, what is the **Identity token type** value?

Answer: Transport Default

Question 13: Explain briefly what this node does.

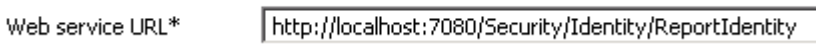
Answer: The node sets a pointer to the Envelope body in the Output root (`OutputRoot.XMLNSC.Envelope.Body`). It then creates a "report" tree that contains the values of the `Input Root.Properties` identity source properties. It also clears the HTTP input header and the MessageIdentity field in the `OutputRoot.Envelope.Body`.

Question 14: Did the **MapIdentity** Compute node in the subflow generate values for the **IdentityMapped** elements in the OutputRoot.Properties tree of the message assembly? Why or why not?

Answer: It did not generate values because no `InputRoot.XMLNSC.Envelope.Body.MapIdentity` element was found. Open the ESQL by double-clicking the **MapIdentity** node in the subflow to review the code. The **MapIdActionsRef** variable becomes FALSE, so the **LASTMOVE** test failed. The MapIdentity node only copied the InputRoot root to the OutputRoot.

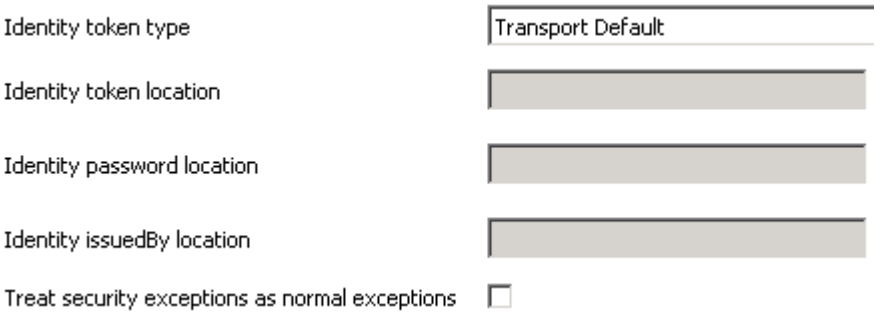
Question 15: The **IdentitySourceType** changed from `username` to `usernameAndPassword`, and the **IdentitySourceIssuedBy** changed from `TestClient` to `HTTP`. Can you explain why this action happened?

Answer: When the message reached the **HTTP_ReqAsID** HTTP request node in the SecurityIdentitySubflow, it started the SecurityIdentityReportFlow (based on the **Web service URL** property).



Examine the properties of the **HTTP_ReportIdentity** HTTP Input node in the **SecurityIdentityReportFlow**, specifically the **Security** tab.Because the **Identity token type** property is **Transport Default** (the default value for the HTTP Input node), the node sets the **IdentitySourceType** element in the message assembly to

Username+Password. It also retrieves the identity from the HTTP Basic Auth transport header.



- Because the **Identity issuedBy location** property is not set, the node sets the **IdentitySourceIssuedBy** element in the message to HTTP.

In this test, although the original message came in with a user name only, the behavior of some of the message processing nodes changed the content of the message. In this case, it changed the source type and source identity of the original message.

Question 16: Did the **MapIdentity** Compute node in the subflow generate values for the **IdentityMapped** elements in the OutputRoot.Properties tree of the message assembly? Why or why not?

Answer: As in the first test, it did not generate values because no InputRoot.XMLNSC.Envelope.Body.MapIdentity element was found. The MapIdentity node copied the InputRoot root to the OutputRoot.

Question 17: Did the **MapIdentity** Compute node in the subflow generate values for the **IdentityMapped** elements in the OutputRoot.Properties tree of the message assembly? Why or why not?

Answer: In this test, the message contained an InputRoot.XMLNSC.Envelope.Body.MapIdentity. Because this folder was present, the **MapIdentity** node mapped the credentials that were contained in the MQMD to mapped values, simulating the action of an external security provider. The **MapIdentity** node populated the IdentityMapped elements of the message assembly.

Before **MapIdentity**:

```
(0x03000000:NameValue):IdentitySourceType     = 'username' (CHARACTER)
(0x03000000:NameValue):IdentitySourceToken    = 'TESTUSER' (CHARACTER)
(0x03000000:NameValue):IdentitySourcePassword = '' (CHARACTER)
(0x03000000:NameValue):IdentitySourceIssuedBy = 'BRKTSTCLNT' (CHARACTER)
(0x03000000:NameValue):IdentityMappedType     = '' (CHARACTER)
(0x03000000:NameValue):IdentityMappedToken    = '' (CHARACTER)
(0x03000000:NameValue):IdentityMappedPassword = '' (CHARACTER)
(0x03000000:NameValue):IdentityMappedIssuedBy = '' (CHARACTER)
```

The **IdentitySource** elements (IdentitySourceType, IdentitySourceToken, IdentitySourcePassword, and IdentitySourceIssuedBy) are populated, but the corresponding **IdentityMapped** elements are not.

After **MapIdentity**:

```
:NameValue):IdentitySourceType      = 'username' (CHARACTER)
:NameValue):IdentitySourceToken     = 'TESTUSER' (CHARACTER)
:NameValue):IdentitySourcePassword  = '' (CHARACTER)
:NameValue):IdentitySourceIssuedBy  = 'BRKTSTCLNT' (CHARACTER)
:NameValue):IdentityMappedType      = 'usernameAndPassword' (CHARACTER)
:NameValue):IdentityMappedToken     = 'testuser@company.com' (CHARACTER)
:NameValue):IdentityMappedPassword  = 'p_testuser2012' (CHARACTER)
:NameValue):IdentityMappedIssuedBy  = 'SecurityIdentitySubFlow_MapIdentity'
```

Observe that the **IdentityMapped** elements are now populated. The **IdentitySource** elements retained their original values.

Question 18: Compare that trace output to the output generated after the message exits the MapIdentity node (in the previous step). What changed between the two traces?

Answer: In the trace output from the **SecurityIdentityReportFlow**, the **IdentitySource** elements are populated with the **IdentityMapped** elements that showed in the previous trace, and the **IdentityMapped** elements are empty. This action is the result of the **HTTP_ReqAsID** HTTP request node. When an output or request node propagates an identity, the *mapped* identity is used. If the mapped identity is not set, or if the node does not support the token type that is contained within the message, the *source* identity is used.

Exercise review and wrap-up

In the first part of this exercise, you review the message flows that this exercise uses. You also deploy the message flows.

In the remaining parts of this exercise, you use the Integration Toolkit Unit Test Client to test the message flow for three scenarios.

Having completed this exercise, you should be able to:

- Propagate security credentials within a message flow
- Configure message flow nodes to enable secure message processing
- List the types of security tokens
- Use a Compute node to simulate the actions of an external security provider

Exercise 8. Recording and replaying message flow data

What this exercise is about

In this exercise, you define monitoring events and then record and replay messages. You also learn how to capture and report failed events.

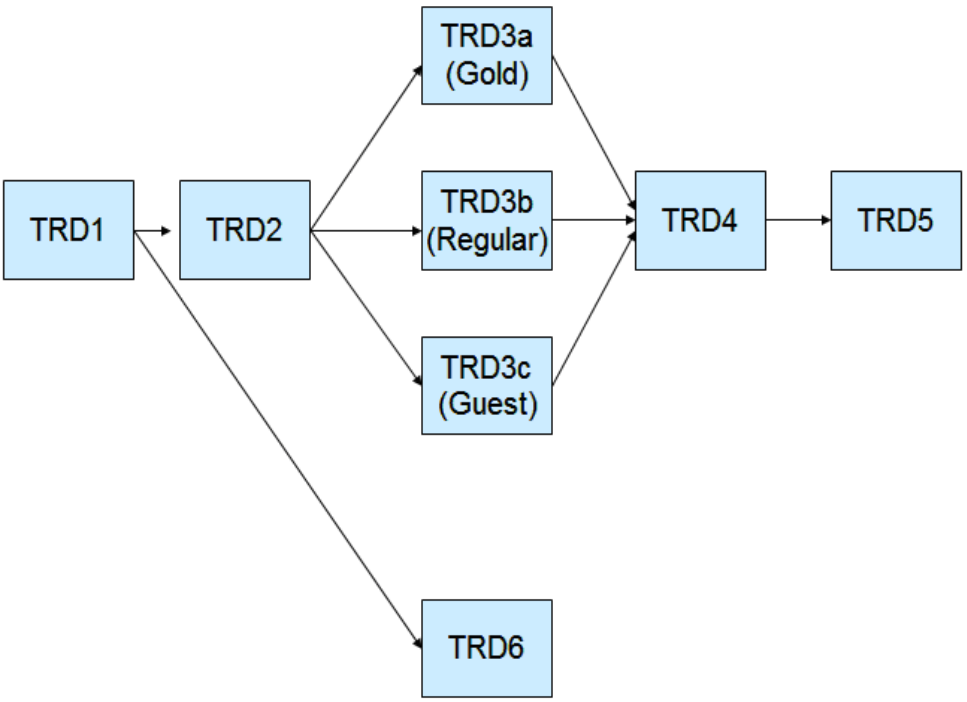
What you should be able to do

After completing this exercise, you should be able to:

- Define monitoring events
- Activate flow monitoring
- View event messages in the IBM Integration web user interface
- Replay messages
- Capture and report failed events

Introduction

This exercise uses an application that contains several message flows (summarized in the following figure).



The application processes stock trade requests, and each trade runs five message flows: TRD1, TRD2, TRD3*, TRD4, and TRD5. The TRD3 message flow is selected based on the customer type:

- If the customer type is “Gold”, the TRD3a message flow runs.
- If the customer type is “Regular”, the TRD3b message flow runs.
- If the customer type is “Guest”, the TRD3c message flow runs.

If a validation failure occurs in the first message flow, TRD1, then the TRD6 flow runs.

Several of the message flow nodes have monitoring points that are defined on them by using the **Monitoring** node properties. These monitoring points publish certain items of the message payload data. The **Data viewer** in the IBM Integration web user interface uses this data to access processed messages, and to resubmit (replay) the message for reprocessing.

Note

Flow monitoring can also be achieved noninvasively by using monitoring templates, which are not covered in this lab.

In this second part of this exercise, you set up a DB2 database for record and replay database. You also configure the JDBC connection to the database, and the IBM MQ queues that are required for the Trades application. Command and batch files are provided in the `C:\labfiles\Lab08-RandR\install` directory to automate this process.

Requirements

- IBM Integration Bus V10.
- IBM MQ V8 and a queue manager that is named IIBQM.
- IBM DB2.
- IBM MQ SupportPac IH03: RFHUtil in the `C:\software\ih03` directory.
- An IBM Integration Bus integration node with an IBM MQ queue manager specified on the integration node. The exercise instructions refer to IIBNODE_WITHQM that is created in Exercise 1, Part 1.
- Lab files in the `C:\labfiles\Lab08-RandR` directory.
- The IBM MQ queues (created in the exercise): TRADE.VALIDATE.IN, TRADE.CUST.TYPE.IN, TRADE.GOLD.IN, TRADE.REGULAR.IN, TRADE.VALIDATION.FAILURE.IN, TRADE.GUEST.IN, TRADE.RECONCILIATION.IN, TRADE.COMPLETE.IN, TRADE.COMPLETE.OUT, TRADE.FIX.IN, TRADE.FIX.OUT, TRADE.REPLAY.INPUT.

Exercise instructions

Part 1: Set up the environment for record and replay

In this part of the exercise, you set up the record and replay database, JDBC connections, and the queues that are required for the Trades application.

Command and batch files are provided in the `C:\labfiles\Lab08-RandR\install` directory to automate the setup process. You can examine any of the files by using Notepad.

Important

This lab exercise assumes that you already created an integration node that is named IIBNODE_WITHQM that specifies IIBQM as its queue manager. It also assumes that the SYSTEM.BROKER queues were created on the queue manager.

See Exercise 1, Part 1 for more detailed instructions for creating the queue manager, integration node, and SYSTEM.BROKER queues.

__ 1. Create the queues and subscriptions for the TRADES message flows on the queue manager that is specified on the integration node. This step assumes that you have a queue manager that is named IIBQM.

__ a. Start an IBM Integration Console as the administrator (right-click the IBM Integration Console shortcut on the desktop and then click **Run as administrator**).

__ b. In the IBM Integration Console, change directories to `C:\labfiles\Lab08-RandR\install\MQsetup`. Type:

```
cd C:\labfiles\Lab08-RandR\install\MQsetup
```

__ c. Enter the command to create the queues on the integration node queue manager IIBQM:

```
runmqsc IIBQM < TradeQueues.mqsc
```

__ d. In IBM MQ Explorer, verify that the following objects were created:

- The local queues TRADE.VALIDATE.IN, TRADE.CUST.TYPE.IN, TRADE.GOLD.IN, TRADE.REGULAR.IN, TRADE.VALIDATION.FAILURE.IN, TRADE.GUEST.IN, TRADE.RECONCILIATION.IN, TRADE.COMPLETE.IN, TRADE.COMPLETE.OUT, TRADE.FIX.IN, TRADE.FIX.OUT, TRADE.REPLAY.INPUT, and RECORD.REPLAY.SUB
- The subscription RECORD.REPLAY.SUB with the topic string of `IIBNODE_WITHQM/Monitoring/#`

__ 2. Create the record and replay database.

__ a. In the IBM Integration Console, type the following commands to create the DB2 database and tables:

```
cd ..\DBSetup
CreateTRADES_Tables.bat
```

This command opens a DB2 command window and creates the record and replay tables in the TRADES database. The tables are re-created so that this lab exercise starts with a clean display of monitor events, and so that new events are easily viewable in the IBM Integration web user interface.

It might take a few minutes for this batch file to complete.

Verify that the command ran successfully and then press Enter in the DB2 command window to close the window.

__ 3. Define the JDBC connections from the IIBNODE_WITHQM integration node to the TRADES database tables.

In the IBM Integration Console, type the following command:

```
ConfigureJDBC
```

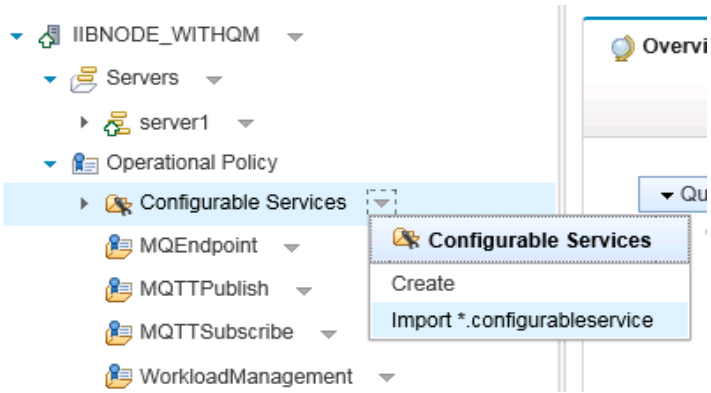
__ 4. Import the data store configurable service.

The record and replay function uses a data store, which represents the database that holds the captured monitoring events. The data store is defined to the integration node by using configurable services.

You can define the configurable services manually. For this lab, the configurable services are already created but they must be imported into the IIBNODE_WITHQM integration node.

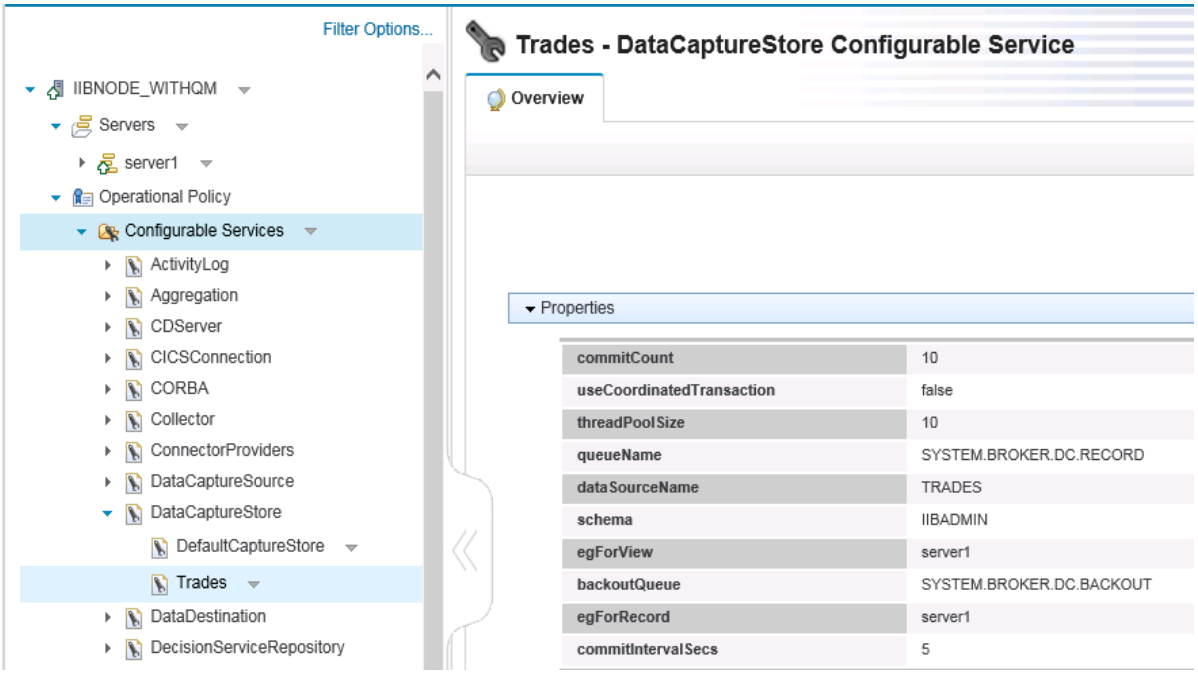
__ a. In the IBM Integration web interface for IIBNODE_WITHQM, expand the **Operational Policy** folder.

__ b. Click the down arrow to the right of the **Configurable Services** folder and then click **Import *.configurableservice**.



__ c. Browse to the `C:\labfiles\Lab08-RandR\configurable_services` directory and then select the **`Trades_data_capture_store.configurableservice`** file. Click **OK**.

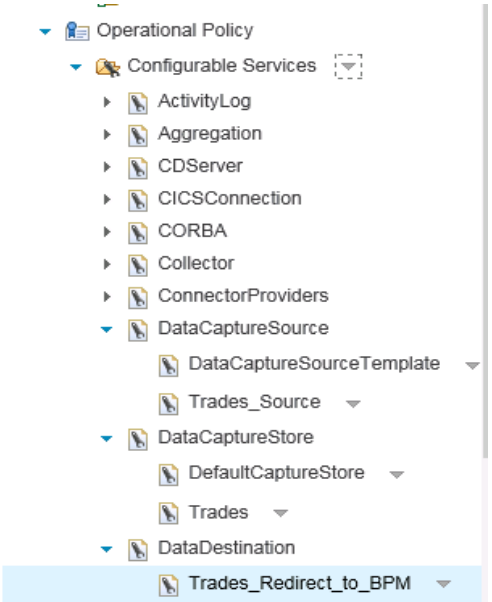
__ d. The **Trades** configurable service should now be listed under the **Configurable Services > DataCaptureStore** folder.



__ 5. Following the same process as in step 4, import `Trades_source.configurableservice` and `Trades_BPM_Data_Destination.configurableservice` from the `C:\labfiles\Lab12-RandR\configurable_services` directory.

You should now have three new configurable services for the Trades application:

- DataCaptureSource > Trades_Source
- DataCaptureStore > Trades
- DataDestination > Trades_Redirect_to_BPM



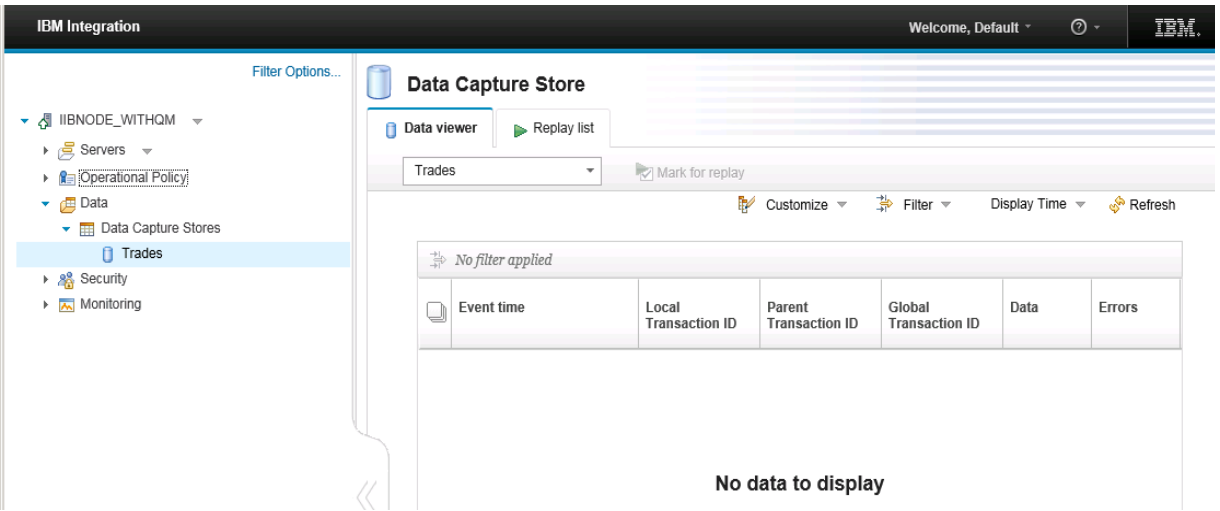
Note

The **Trades_Source** service subscribes to the topic `$SYS/Broker/IIBNODE_WITHQM/Monitoring/server1/#`. This data source collects all monitoring events that the applications in the **server1** integration server generate. This data source does not collect events that are emitted in other nodes or integration servers.

Part 2: View messages in the IBM Integration web interface

__ 1. In the IBM Integration web interface, expand **Data > Data Capture Stores** in the navigator and then select the **Trades** data capture store.

The events list for the **Trades** database is empty.

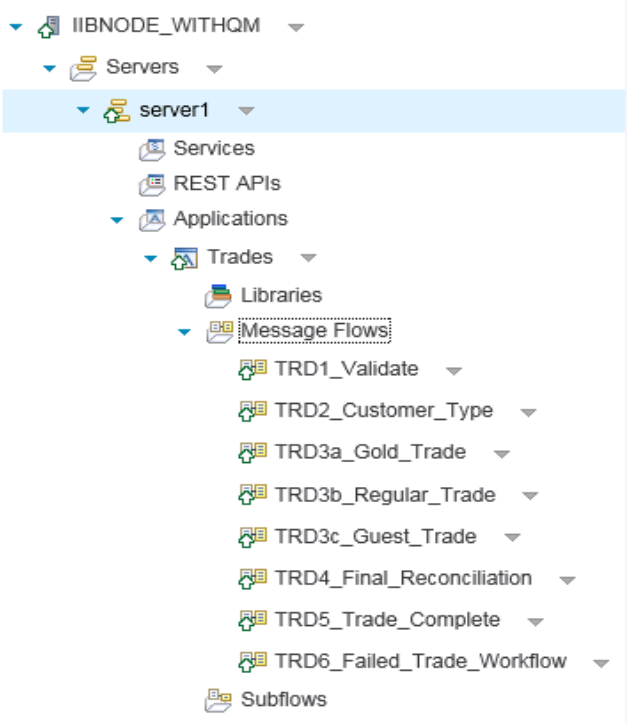


__ 2. In the IBM Integration web interface, deploy the BAR File that is named `Trades.bar` from the `C:\labfiles\Lab08-RandR\application` directory to an integration server on the IIBNODE_WITHQM integration node.

This exercise assumes that the integration server is named `server1`.

__ 3. Validate that the **Trades** application is deployed.

Expand the **Trades** application in the IBM Integration web user interface. Verify that the application contains eight message flows. The message flows are run in sequence, with just one of the TRD3* messages flows used, depending on the type of customer.



__ 4. Enable flow monitoring on the message flows in the **Trades** application.

Type the following command to enable monitoring on each of the message flows:

```
cd ..\..\monitoring
enableMonitoringTrades
```

Information

This command runs the integration node commands:

```
mqsichangeflowmonitoring IIBNODE_WITHQM -e server1 -k Trades
-f TRD1_Validate -c active
```

It also runs equivalent commands for the other message flows in the **Trades** application.

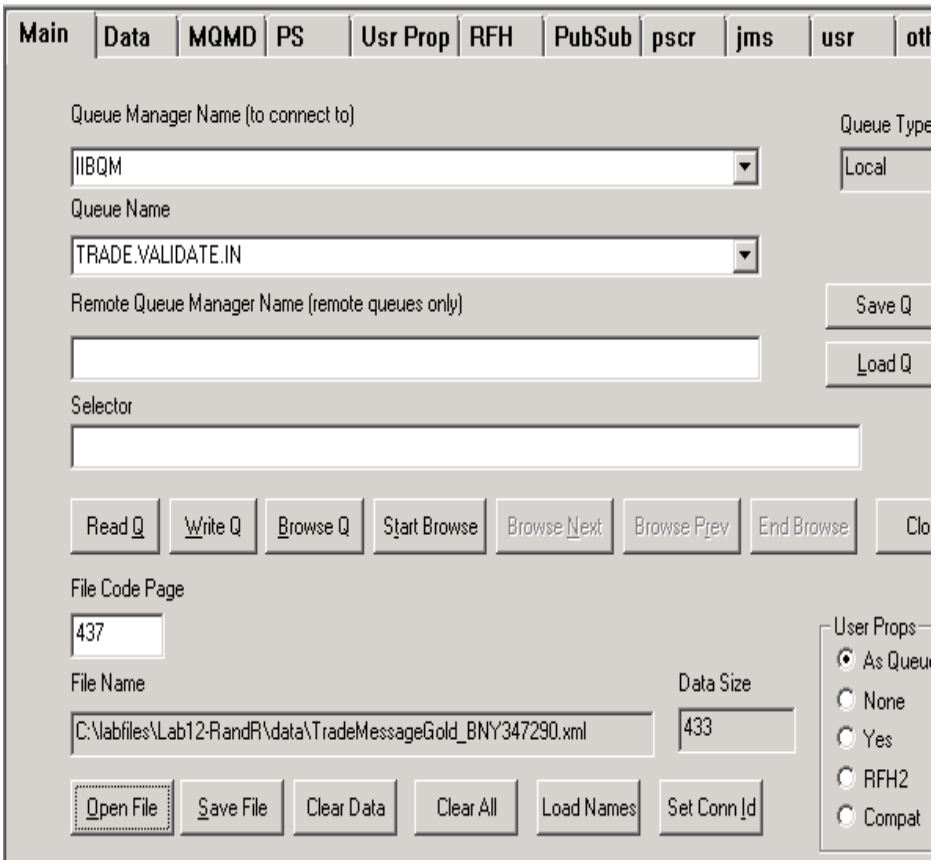
If you redeploy the **Trades** application, the flow monitoring status is reset. You must enter the `enableMonitoringTrades` command again to reactivate flow monitoring.

__ 5. Use RFHUtil to send some new events to the **Trades** application queue TRADE.VALIDATE.IN on the IIBQM queue manager.

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

__ b. In RFHUtil, open the file for a “Gold” customer:
`C:\labfiles\Lab08_RandR\data\TradeMessageGold_BNY347290.xml`

__ c. Send one instance of the data to the queue TRADE.VALIDATE.IN on IIBQM.



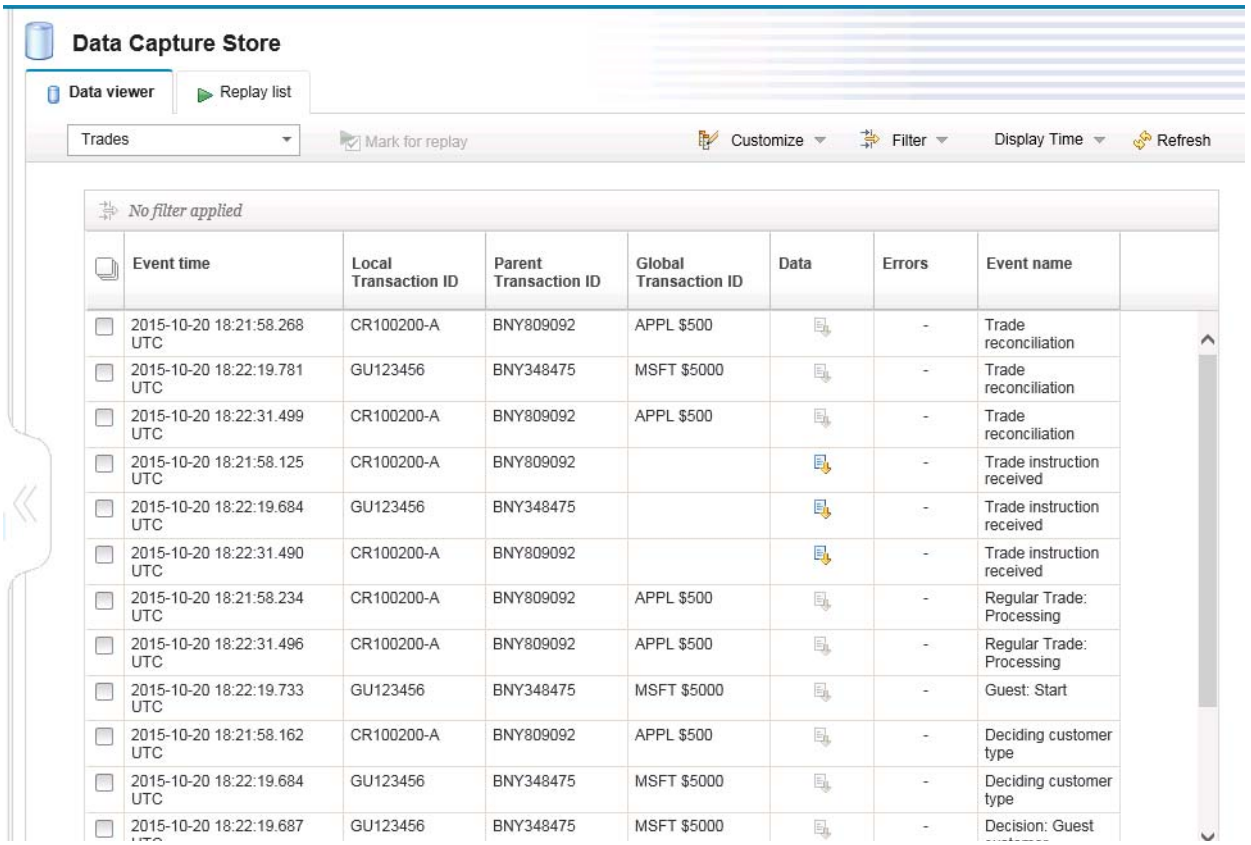
__ d. Send a message to TRADE.VALIDATE.IN for a “Regular” customer:
`C:\labfiles\Lab08-RandR\data\TradeMessageRegular_BNY809092.xml`

__ e. Send a message to TRADE.VALIDATE.IN for a “Guest” customer:
`C:\labfiles\Lab08-RandR\data\TradeMessageGuest_BNY348475.xml`

__ 6. Click **Refresh** on the **Data viewer** in the IBM Integration web interface to show the new events.

Note

If you receive an error when trying to access the Data Capture Store, restart the **server1** integration server and try to access the **Data viewer** again.



Event time	Local Transaction ID	Parent Transaction ID	Global Transaction ID	Data	Errors	Event name
2015-10-20 18:21:58.268 UTC	CR100200-A	BNY809092	APPL $500		-	Trade reconciliation
2015-10-20 18:22:19.781 UTC	GU123456	BNY348475	MSFT $5000		-	Trade reconciliation
2015-10-20 18:22:31.499 UTC	CR100200-A	BNY809092	APPL $500		-	Trade reconciliation
2015-10-20 18:21:58.125 UTC	CR100200-A	BNY809092			-	Trade instruction received
2015-10-20 18:22:19.684 UTC	GU123456	BNY348475			-	Trade instruction received
2015-10-20 18:22:31.490 UTC	CR100200-A	BNY809092			-	Trade instruction received
2015-10-20 18:21:58.234 UTC	CR100200-A	BNY809092	APPL $500		-	Regular Trade: Processing
2015-10-20 18:22:31.496 UTC	CR100200-A	BNY809092	APPL $500		-	Regular Trade: Processing
2015-10-20 18:22:19.733 UTC	GU123456	BNY348475	MSFT $5000		-	Guest: Start
2015-10-20 18:21:58.162 UTC	CR100200-A	BNY809092	APPL $500		-	Deciding customer type
2015-10-20 18:22:19.684 UTC	GU123456	BNY348475	MSFT $5000		-	Deciding customer type
2015-10-20 18:22:19.687	GU123456	BNY348475	MSFT $5000		-	Decision: Guest

__ 7. The **Data viewer** uses the standard column heading names. You can customize the headings so that they are more descriptive.

You have several options for customizing the column headings:

- You can change the display name of each column by double-clicking the required display name, and entering another name. These changes are stored in the Integration Registry, and are retained uniquely for each data capture store. All users who display data from the same data capture store see the changes that this user makes. If you want to record and view data with different headings, record the events in a separate data capture store.
- You can select or clear any of the recorded fields for display.
- You can override the width of the displayed column. The widths can also be overridden by using the divider bars.

Click **Customize** and change the Display Name for the column headings as follows:

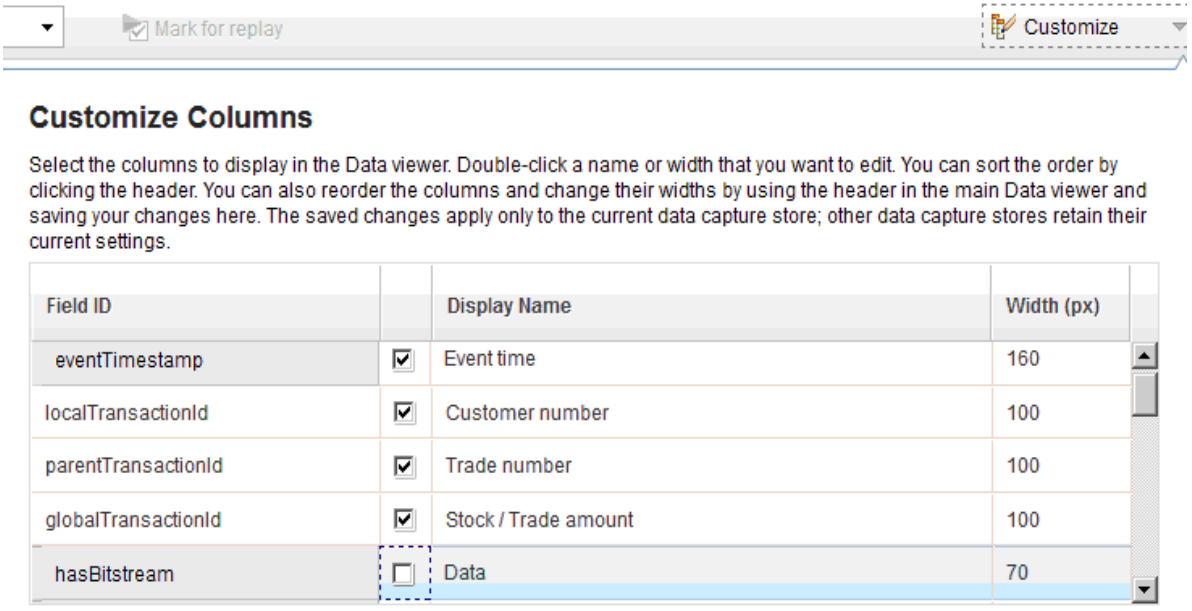
__ a. Change **Local Transaction ID** to: `Customer number`

__ b. Change **Parent Transaction ID** to: `Trade number`

__ c. Change **Global Transaction ID** to: `Stock/Trade amount`

__ d. Change **Event name** to `Trade processing stage` and change the field width to `160`.

__ e. Change **hasBitstream** (**Data**) so that it is not selected for display.



__ f. Click **Apply**. The column names are updated with the new values.

__ 8. You can click the column headers to change the order of the events in the **Data viewer**.

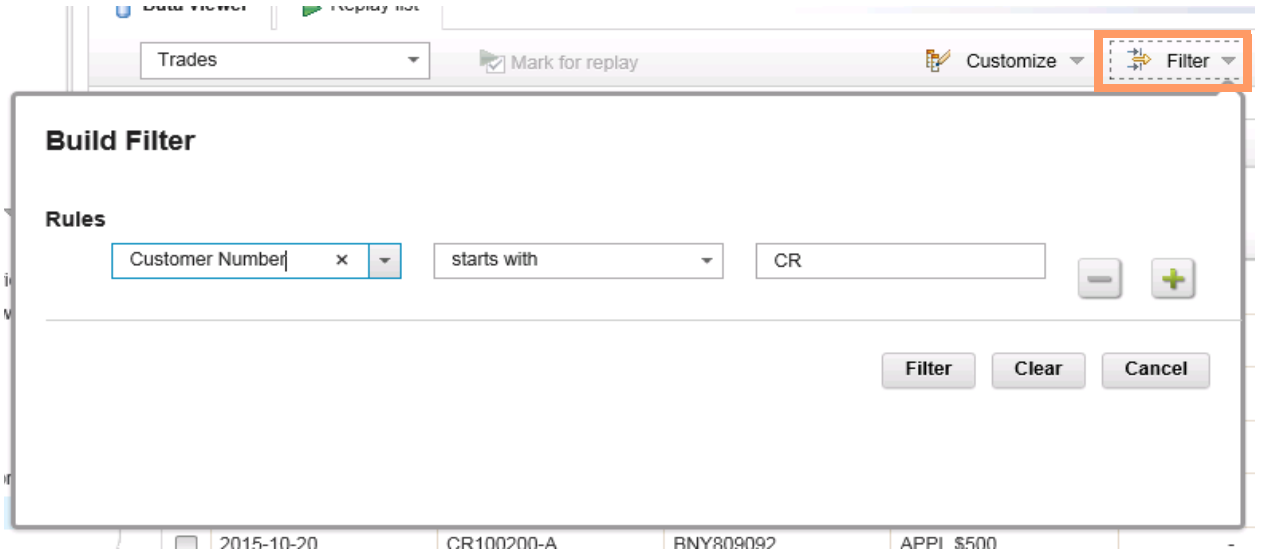
For example, click the **Event time** column to display the oldest events first.

__ 9. You can limit the data that is displayed in the **Data viewer** by using the **Filter** function.

__ a. Click **Filter**.

__ b. Specify the filter criterion: `Customer number starts with CR`

__ c. Click **Filter** to activate the defined filters.



__ 10. You can add more filters to the display.

__ a. Click **Filter** again, and click the green plus sign to add a second filter.

__ b. Specify the filter criterion: `Trade processing stage contains Complete`.

__ c. Click **Filter** to activate the new filter.

Note

The values for each filter are case-sensitive.

__ 11. Clear the filters by selecting **Filter** from the **Data viewer** tab and then clicking **Clear**.

Part 3: Replay messages

In this part of the exercise, you configure the IIBNODE_WITHQM integration node to replay messages. You can use replay to select messages that the IBM Integration web interface lists and send them to the same, or a different, message flow for further processing.

Depending on the message flow and the types of events that you want to replay, the replay queue might be the same input queue that the original message flow uses or a different queue. The replay might also be a separate (and different) message flow.

In this lab, you replay the message by sending it to a separate IBM MQ queue: another application does not process it.

__ 1. Review the configurable service that was defined to enable the replay.

__ a. In the IBM Integration web interface, expand the **Configurable Services** folder.

__ b. Select the **DataDestination > Trades_Redirect_to_BPM** configurable service.

This configurable service enables messages to be routed to the TRADE.FIX.IN queue on the IIBQM.

In this lab, a simple message flow in the **Trades** application processes this queue. In another scenario, a business process management application might process it and amend the message before sending it to the **Trades** application again.

__ 2. In the IBM Integration web interface **Data viewer**, enable the **Data** column.

__ a. Click **Customize**.

__ b. Enable the **hasBitstream** (**Data**) column.

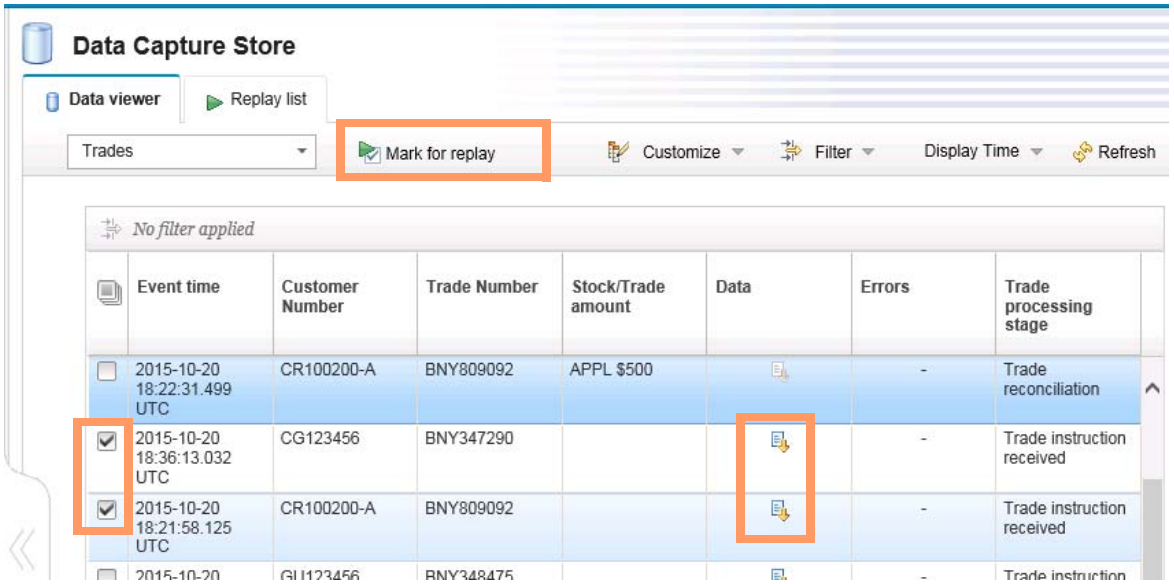
__ c. Click **Apply**.

__ 3. Select some of the messages for replay.

__ a. Select the check box for the message to enable it.

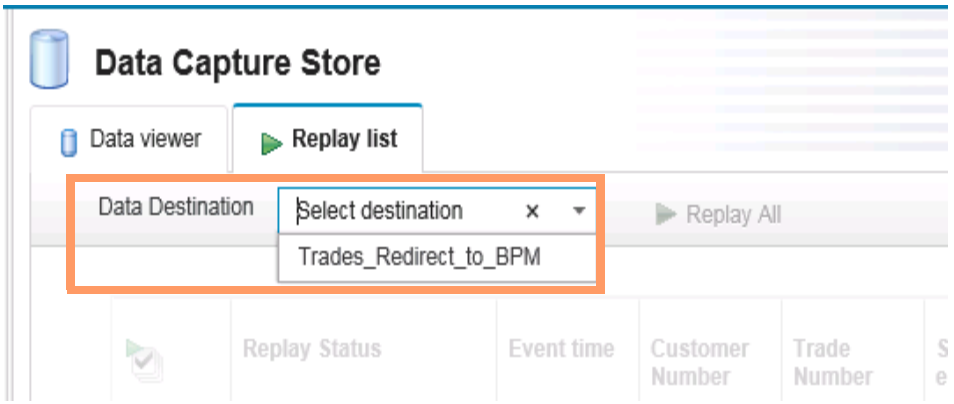
__ b. Make sure that at least one of the selected messages shows the colored bitstream icon in the **Data** column.

__ c. Click **Mark for replay**, which is now active.



__ 4. Clicking **Mark for replay** displays the **Replay list**. However, you still cannot start the **Replay** function. You must first select the **Data Destination**.

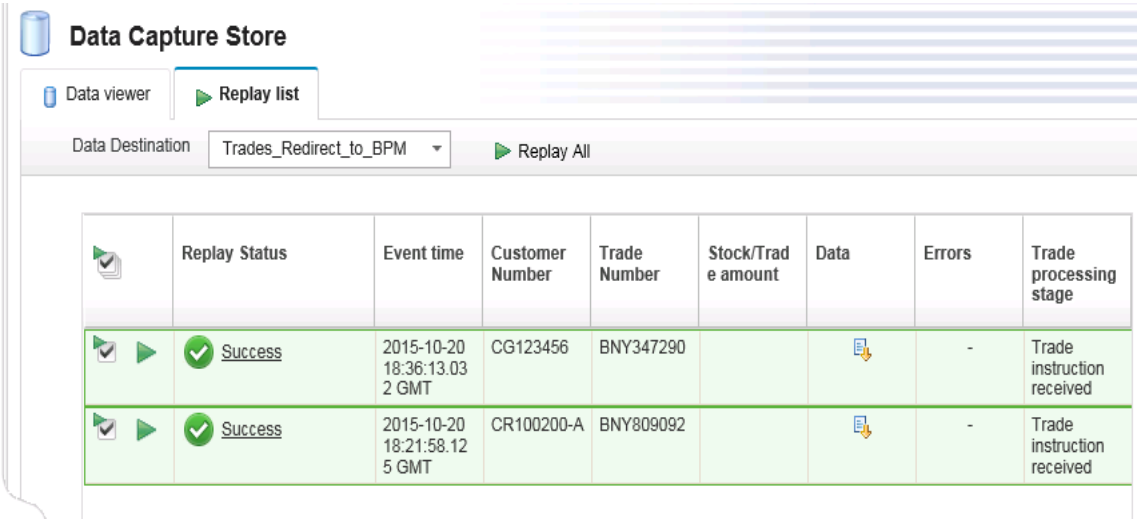
On the **Data Destination** menu, select the destination **Trades_Redirect_to_BPM**.



__ 5. On the **Replay list** view, click **Replay All** (or you can replay each item individually by clicking the green arrow next to each message).

You should see that the item that contained the data bitstream was successfully sent to the replay destination. It does not mean that the application successfully processed data; it was successfully sent to the receiving destination.

You cannot replay items that do not contain a data bitstream.



__ 6. Confirm that the messages were sent to the replay queue.

__ a. Open IBM MQ Explorer.

__ b. Select **Queues** under **IIBQM**.

__ c. The **Current queue depth** of TRADE.FIX.OUT should increase by 2 (or the number of messages that you sent for replay).

Part 4: Handle failed messages

If a message flow encounters an error during processing, it can be captured and reported by using the IBM Integration web interface.

To enable this feature, the monitoring point on the message flow node must include the `$ExceptionList` string in the monitoring event message.

__ 1. Using RFHUtil, send a bad message to the **Trades** application.

__ a. Open the file `C:\labfiles\Lab08-RandR\data\TradeMessage_BadMessage.xml`

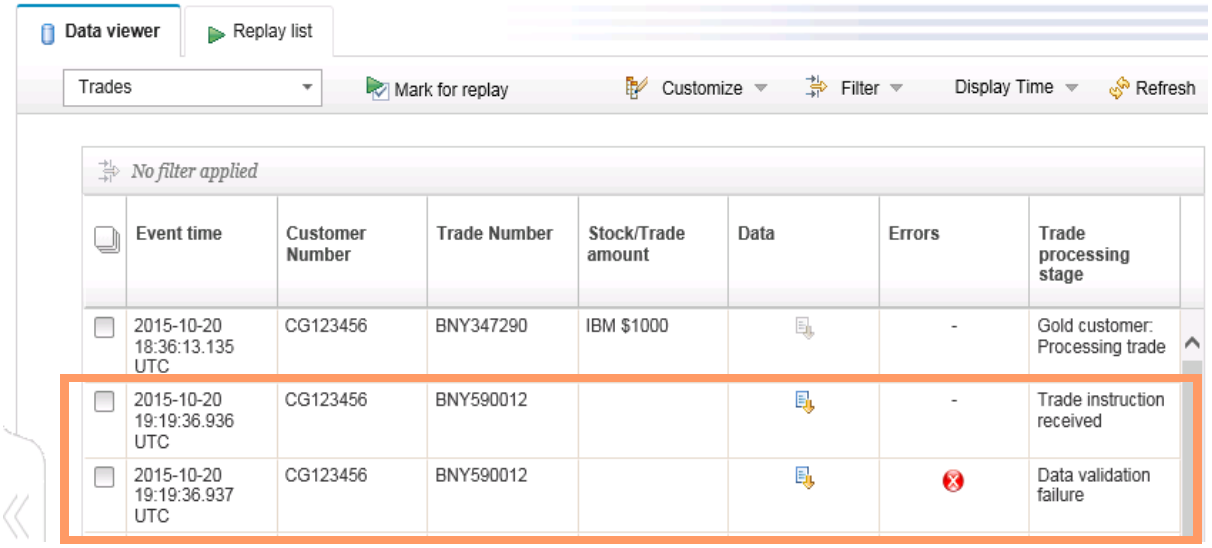
__ b. Send the message to TRADE.VALIDATE.IN queue on IIBQM.

Although this message is a valid XML message, it is missing a required XML element. It fails validation on the **ReceiveTrade** node because validation is set to **Content and Value**.

__ 2. In the IBM Integration web interface, you should see two new entries.

The first new entry has a **Trade processing stage** of *Trade instruction received*.

The second new entry has a **Trade processing stage** of *Data validation failure*.



__ 3. Failed events can be highlighted in the IBM Integration web interface **Data viewer** by customizing the displayed columns. Click **Customize**.

__ a. If it is not already selected, enable the **hasException** (**Errors**) column.

__ b. Click **Apply**.

The **Errors** (**hasException**) column shows a red cross for all monitoring events that contain the `$ExceptionList` data.

Exercise cleanup

__ 1. In the IBM Integration web interface, stop the running application and then delete it from the **server1** integration server.

__ 2. Close RFHUtil.

__ 3. Close the IBM Integration web interface.

End of exercise

Exercise review and wrap-up

In Part 1 of this exercise, you set up the environment for recording and replaying messages. Setup included building the database tables and importing the configurable services.

In Part 2 of this exercise, you viewed event messages in the IBM Integration web interface, customized the Data viewer, and defined filters to limit the data that is displayed.

In Part 3 of this exercise, you replayed messages by sending them to an alternative queue.

In Part 4 of this exercise, you configured the message flow to generate the `$ExceptionList` string and identified failed messages in the IBM Integration web interface.

Having completed this exercise, you should be able to:

- Activate message flow monitoring and store events in a database
- Enable an integration node to connect to a database with ODBC
- View event messages in the IBM Integration web user interface
- Replay a message to an IBM MQ queue by using the IBM Integration web interface

.